Optimisation d'ensembles de capteurs pour le suivi et la recherche de cibles

Florian Delavernhe

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

université angers

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ D'ANGERS
COMUE **U**NIVERSITÉ **B**RETAGNE **L**OIRE

École Doctorale N° 601
Mathématique et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Florian DELAVERNHE

Optimisation d'ensembles de capteurs pour le suivi et la recherche de cible

Thèse présentée et soutenue à UNIVERSITÉ D'ANGERS, le 14/09/2020
Unité de recherche : LERIA (EA 2645)
Thèse N° :

Rapporteurs avant soutenance :

Safia Kedad-Sidhoum	Professeur des universités, CNAM de Paris
Michael Poss	Chargé de recherche HDR, LIRMM CNRS, UMR 5506, Montpellier

Composition du Jury :

Président :	Christelle Jussien-Guéret	Professeur des universités, Université d'Angers
Examinateurs :	Christian Artigues	Directeur de recherche, LAAS CNRS, UPR 8001, Toulouse
	Adrien Goëffon	Maître de conférences HDR, Université d'Angers
	Safia Kedad-Sidhoum	Professeur des universités, CNAM de Paris
	Michael Poss	Chargé de recherche HDR, LIRMM CNRS, UMR 5506, Montpellier
Dir. de thèse :	André Rossi	Professeur des universités, Université Paris-Dauphine
Co-dir. de thèse :	Marc Sevaux	Professeur des universités, Université Bretagne Sud

Invité :

	Patrick Jaillet	Dugald C. Jackson Professor, Massachusetts Institute of Technology, États-Unis

REMERCIEMENTS

Je tiens tout d'abord à remercier Safia Kedad-Sidhoum et Michael Poss qui ont accepté d'être rapporteurs pour cette thèse. J'espère que sa relecture ne leur a pas été trop fastidieuse et qu'elle a éveillé leur intérêt.

Je remercie fortement mes encadrants, André Rossi et Marc Sevaux, qui m'ont accordé leur confiance pour cette thèse. Ils ont su me guider patiemment et avec brio le long de ces trois années et cela malgré la distance qui nous séparait. Leurs précieux conseils et l'expérience gagnée à leur coté resteront pour toujours avec moi.

Je remercie tout particulièrement Patrick Jaillet pour son accueil au MIT, un déplacement qui restera un évènement phare dans ma vie. Cette collaboration a été l'occasion pour moi de découvrir avec curiosité un nouveau domaine et surtout d'approfondir avec intérêt mes connaissances sur des éléments que j'avais délaissés auparavant.

Je remercie aussi, bien évidemment, tous les membres du département informatique de l'UFR Sciences de l'université d'Angers et du laboratoire LERIA pour leur accueil et conseils. Mes plus sincères remerciements aussi pour les secrétaires, Christine Bardaine et Catherine Pawlonski, avec qui c'est toujours un plaisir de discuter et dont l'aide et l'efficacité sont grandement appréciables.

Un grand remerciement aussi à tous les autres doctorants, ATER, stagiaires,.... A tous ces assidus de la pause midi, buveurs de thé ou café, ces distractions indispensables qui ont contribuées indirectement aux travaux de cette thèse, merci.

Enfin et surtout, je remercie ma famille et mes amis dont le support avant et pendant (et après ?) cette thèse ont été indispensable. Plus particulièrement, je remercie Ekaterina dont le support et la patience au quotidien m'ont permis de trouver les ressources nécessaires à ce travail.

TABLE DES MATIÈRES

INTRODUCTION

Motivations

Les ensembles de capteurs sans fil forment une technologie importante pour l'industrie actuelle et future [1]. Ce sont des appareils généralement peu chers, fonctionnant avec des batteries et qui peuvent récolter diverses données sur leur environnement et les transmettre à une station de base pour traitement. Typiquement, un capteur dispose de composantes pour acquérir des données, un processeur pour les traiter et un moyen de communication pour les transmettre. Cependant, ses composantes dépendent de sa mission et un capteur peut être plus ou moins complexe en fonction de sa conception. Ainsi, il peut être mobile ou statique, capter des images, du son ou des grandeurs physiques. Comme le montrent YICK, MUKHERJEE et GHOSAL [2], les nombreux types de capteurs permettent une grande variété d'applications différentes, dans les domaines militaires, industriels, médicaux ou environnementaux. Les avancées technologiques des dernières années ont eu pour effet notamment de diminuer les coûts de production des capteurs et d'en accroître l'autonomie énergétique. Cela a entraîné une croissance importante du marché économique des capteurs. Par exemple, [3] évalue le marché des capteurs sans fil à 29,06 milliards de dollars en 2016 et ses prévisions le voient atteindre 93,86 milliards de dollars en 2023, soit une croissance annuelle d'environ 18% qu'on retrouve dans beaucoup d'autres évaluations.

Les avantages conférés par les réseaux de capteurs sans fil sont multiples et leur permettent d'apparaître comme une solution avantageuse à des problèmes de surveillance de cibles. En effet, ces réseaux sont dits *ad hoc* et ne nécessitent donc aucune infrastructure préalable pour fonctionner. De plus, les capteurs sans fil statiques étant relativement peu coûteux, ils peuvent être déployés dans des zones sensibles où ils risquent d'être endommagés ou détruits, sans que cela n'aient de conséquences dramatiques pour le réseau. Ils sont donc parfaits pour un déploiement en urgence dans une zone difficile ou dangereuse d'accès pour un personnel humain, avec largage par avion par exemple. C'est le cas typiquement pour un déploiement dans des zones ravagées par des catastrophes naturelles ou dans des théâtres d'opérations militaires. La surveillance de cibles mobiles est une application bien adaptée aux réseaux de capteurs sans fil dans ce contexte, où la récolte des données sur les cibles et leurs envois peuvent être effectués tout au long de leurs trajets par les nombreux capteurs présents. Dans ce type d'application, la durée de vie du réseau, c'est-à-dire son aptitude à surveiller les cibles mobiles dans la zone spécifiée est souvent limitée par la capacité des batteries embar-

quées sur les capteurs, que l'on ne peut bien souvent ni remplacer, ni recharger.

Les récentes avancées de la technologie des capteurs ont aussi permis l'utilisation de véhicules autonomes pour constituer des flottes de capteurs mobiles qui étendent les possibilités des réseaux de capteurs, et posent de nouveaux problèmes d'optimisation. Un exemple typique est l'utilisation de capteurs dans le domaine de la théorie de la recherche bayésienne. Ce domaine est apparu pendant la seconde guerre mondiale [4] et part de l'application concrète de la recherche efficace de sous-marins mais qui trouve de nos jours de nouvelles applications utilisant notamment des ensembles de capteurs, ou véhicules autonomes. Les missions peuvent relever de la recherche et du sauvetage [5] ou de la recherche de débris d'avion. La théorie de la recherche est une approche probabiliste qui représente la réussite d'une mission par une probabilité de succès qui découle de décisions prises à partir de données d'entrées exprimées elles aussi sous la forme de probabilités.

Les travaux effectuées au cours de cette thèse abordent des problématiques de recherche et de surveillance de cibles. Dans un premier temps, l'objectif est de formaliser pour ces missions des problèmes d'optimisation qui représentent de manière réaliste les besoins et limitations. Pour cela, nous généralisons des problèmes particuliers abordés dans la littérature, où sont pris en considération les déplacements de capteurs, les communications et transfert de données, les cas multi-cibles... Dans un deuxième temps, nous proposons des méthodes de résolution pour chaque problème identifié. Le but de cette thèse est donc à la fois de généraliser des problèmes issus de la littérature pour en proposer des versions plus réalistes de problèmes industriels ou militaires, mais aussi proposer de nouveaux outils algorithmiques pour les résoudre efficacement, à l'échelle où ils se posent.

Le plan de cette thèse est le suivant. Dans la suite de cette introduction, avec la sous-section *Littérature*, nous présentons les travaux existants relatifs aux problématiques d'ensembles de capteurs. Ceux qui sont liés à la recherche et la surveillance de cibles sont abordés plus en détails. Le premier chapitre de cette thèse, chapitre 1, traite du problème d'ordonnancement robuste d'un réseau de capteurs pour la surveillance de cibles sous incertitude temporelle. On y propose une généralisation d'un problème présenté dans la section *Littérature* ainsi qu'une méthode de résolution exacte. Le chapitre 2 se place dans la suite du premier, et traite la même problématique mais cette fois-ci dans le cas dynamique. Les solutions trouvées dans le chapitre 1 sont alors dynamiquement modifiées par une nouvelle méthode, cette fois-ci heuristique, mais offrant des garanties des performances dynamiques. Le chapitre 3 est consacré au problème de recherche de cibles. On y présente un problème différent des précédents chapitres. Encore une fois, on généralise des travaux précédents afin de conférer davantage de réalisme au problème abordé, ce qui se traduit aussi par une complexité algorithmique accrue. La méthode de résolution proposée est différente, dans sa conception, des méthodes de la littérature, mais permet d'obtenir des résultats de bonne qualité en peu de temps. Enfin, la

conclusion résume les travaux effectués et révèle des perspectives ouvertes par ces travaux, dont le développement est en cours.

Littérature

Les problèmes d'optimisation dans les ensembles de capteurs

La littérature sur les ensembles de capteurs et leurs utilisations est très vaste [6]. Il y a tout autant de types de capteurs différents que d'applications possibles ou d'organisations à mettre en place. La recherche scientifique sur le sujet peut alors aborder des problématiques différentes qui font appels à des domaines variés. Par exemple, dans sa revue de la littérature sur les réseaux de capteurs, YICK, MUKHERJEE et GHOSAL [2] considèrent deux principaux thèmes : (i) la technologie "physique" sur laquelle repose les capteurs et (ii) leurs applications. Les problèmes traités dans la littérature peuvent être ainsi divisés en trois groupes qui abordent un thème ou les deux. On trouve donc des travaux sur le système physique d'un capteur (ses composantes telles que la batterie, son système d'exploitation,...), d'autres publications sur le transfert des données récoltées (protocoles de communication, organisation du réseau,...) et d'autres encore sur les services à mettre en place pour accomplir la mission du réseau (couverture de la zone, sécurité, localisation,...). Évidemment, ces trois parties sont intrinsèquement liées les unes avec les autres. Dans cette section consacrée à la littérature ainsi que dans la suite de cette thèse, nous nous focalisons sur l'aspect application d'un ensemble de capteurs, et n'aborderons pas la technologie propre au matériel. Nous présentons dans cette section notamment des problèmes d'optimisation ayant trait aux décisions à prendre pour exploiter au mieux un ensemble de capteurs.

En premier lieu, des capteurs mobiles peuvent être exploités pour des problèmes de couverture, où l'avantage de la mobilité des capteurs est utilisée pour remplir au mieux les zones non couvertes [7]. Ainsi, dans [8], le problème de k-couverture est abordé. L'objectif est de couvrir tous les points de l'espace avec au moins k capteurs. En effet, il peut être important de pouvoir garantir la surveillance d'un point par plusieurs capteurs pour pallier d'éventuelles défaillances ou lorsque la redondance des données de plusieurs capteurs est nécessaire pour confirmer les informations (par exemple, plusieurs capteurs doivent confirmer avoir détecté la cible). Dans un autre travail, LIU et LIANG [9] traite le problème de θ-couverture, où une couverture complète de la zone est impossible et donc l'objectif se réduit à la couverture d'au moins θ% de la zone. L'optimisation d'ensembles de capteurs ne se limite pas aux problèmes de couverture, et est aussi souvent liée à des problématiques de consommation d'énergie ou de durée de vie des capteurs. En effet, les capteurs fonctionnent souvent avec des batteries qui sont supposées non rechargeables car les capteurs sont déployés loin ou en zones dangereuses. Il existe une

abondante littérature sur ces problématiques. Par exemple, deux protocoles importants sont utilisés pour économiser de l'énergie lors de l'utilisation en réseaux des capteurs, LEACH [10] et HEED [11]. Dans ces deux protocoles, les capteurs sont partitionnés en clusters, et chaque cluster sélectionne un capteur comme tête, qui est chargée d'agréger les données collectées et de les transférer vers une station de base. Chaque protocole s'appuie sur une méthode différente pour sélectionner les têtes de cluster. La consommation d'énergie est équilibrée en faisant tourner la tête de chaque cluster, c'est-à-dire en changeant régulièrement le capteur de tête. De plus, LEACH [10] présente de nouvelles métriques pour définir la durée de vie des réseaux. Certains travaux ont associé les problématiques d'économie d'énergie et de couverture de cible comme dans [12]. Dans ce problème, la durée de vie du réseau est également maximisée, mais on considère que chaque capteur est affecté à une famille et chaque famille a une exigence de couverture. Dans [13], les auteurs utilisent le principe de tête de cluster et combinent économie d'énergie et besoin de couverture. Étant donné que les capteurs sont mobiles sur l'horizon temporel, la qualité (couverture) d'un emplacement fluctue. Par conséquent, le réseau est en mesure de déplacer plusieurs fois les têtes de cluster vers différents emplacements, ce qui induit un coût pour chaque relocalisation. Dans leur travail, les auteurs ont proposé une heuristique basée sur la génération de colonnes pour trouver un compromis entre la couverture des données et les coûts de relocalisation. L'utilisation de clusters de capteurs est symptomatique de contraintes de connectivité, c'est-à-dire des applications pour ensembles de capteurs où l'ensemble des données collectées doit être transmis à un nœud de traitement central (ou station de base). Les capteurs deviennent alors multi-rôles : leur consommation d'énergie à chaque instant dépend de leur rôle : inactif, transfert de données, réception de données ou surveillance, comme dans [12, 14, 15].

La surveillance de cibles

La collecte des données dans l'environnement des capteurs est souvent utilisée pour le suivi de cibles. Dans ce type de problème, l'objectif de l'ensemble de capteurs est d'assurer la surveillance continue d'un ensemble de cibles, et donc de récolter et de transmettre des données pendant toute la durée de la mission de surveillance. On y retrouve les problématiques phares des ensembles de capteurs, notamment l'économie d'énergie qui permet ainsi de surveiller plus longtemps les cibles, et la couverture de la zone de la mission. Par exemple, dans le cas de cibles statiques, CARDEI et DU [16] augmentent la durée de vie du réseau en organisant l'ensemble de capteurs en un nombre maximal de sous-ensembles disjoints de capteurs qui couvrent toutes les cibles. Les auteurs ont montré que si les ensembles sont activés séquentiellement, la durée de vie du réseau est améliorée. L'utilisation d'ordonnancements qui alternent les activités des capteurs se révèle souvent efficace pour maximiser la durée de vie d'un réseau de capteurs. CASTAÑO, ROSSI, SEVAUX et VELASCO [17] proposent une approche

de génération de colonnes pour calculer un tel ordonnancement sous contraintes énergétique et de connectivité. Dans [18], les auteurs présentent un algorithme génétique pour résoudre un problème avec un réseau de capteurs directionnels à portée réglable. Ici, un capteur ne peut surveiller que les cibles situées à sa portée, qui est réglable, et ajustable dans une direction d'activation. Plus la portée de détection est grande, plus l'énergie consommée est importante. Le capteur a plusieurs directions possibles pour la détection, mais il ne peut utiliser qu'une seule d'entre elles à la fois. L'approche heuristique proposée par les auteurs trouve une solution efficace pour surveiller la cible immobile tout en maximisant la durée de vie du réseau.

Le critère d'adaptabilité présenté dans [19, 20] est également important. Ce critère juge la capacité d'un protocole de suivi de cibles à s'adapter à un grand nombre de capteurs ou de cibles. En effet, un réseau dense augmente la compléxité du transfert des données et un nombre important de cibles peut augmenter considérablement la consommation énergétique. Les protocoles qui s'adaptent efficacement utilisent généralement les principes de clusters mentionnés précédemment. Le critère de précision évalue, quant à lui, dans quelle mesure les estimations sur les emplacements et les temps de passage des cibles sont proches des emplacements ou des temps réels [20, 21]. C'est-à-dire, à quel point les cibles peuvent s'éloigner de leurs comportements attendus tout en restantes suivies. La littérature couvre la probabilité de perdre une cible, le processus de récupération, la robustesse contre les retards et les avances ou la maximisation de la région de confiance pour l'emplacement de la cible, la sensibilité au bruit, etc. Enfin, un autre critère est celui de la détection des erreurs et défauts. Les algorithmes détectent les défauts du réseau ou les événements externes inattendus et reconfigurent le réseau aussi rapidement que possible, comme dans [22].

Dans les deux premiers chapitres de cette thèse, nous étendons les travaux de [21] où les auteurs ont abordé un problème de robustesse pour le suivi de cibles avec un réseau de capteurs sans fil. L'objectif est de trouver un ordonnancement qui couvre à tout moment une cible, à l'aide d'une trajectoire estimée sur l'ensemble de l'horizon temporel. Cependant, les cibles peuvent subir des retards ou des avances et l'ordonnancement fourni est protégé de ces perturbations par un rayon de stabilité ρ. En effet, quels que soient les retards ou avances des cibles à tout moment de leur trajectoire, l'ordonnancement reste réalisable (couvre la cible) tant que ces valeurs restent inférieures ou égales au rayon de stabilité. La manière d'appréhender l'incertain dans [21] s'inspire de celle qui est présentée dans [23] dans le contexte des chaînes de montage. Dans ces travaux, le but est de planifier des opérations sur un nombre minimum de machines avec des variations possibles dans les temps de traitement, dans un horizon de temps cyclique. La méthode présentée dans [21] commence par une phase de discrétisation, c'est-à-dire la transformation du problème dont les données d'entrée sont géométriques, en données utilisables pour le modéliser comme un problème d'optimisation combinatoire. L'espace couvert est partitionné en faces (ce terme est défini dans la section 1.4). La cible se

déplace alors dans différentes faces, chaque transition entre deux faces étant appelée *tick*. La trajectoire géométrique de la cible n'est plus nécessaire et est transformée en une succession de fenêtres de temps associées à des faces. Ainsi, chaque fenêtre de temps peut être associée à un ensemble de capteurs disponibles pour la surveillance de la cible. Les auteurs ont proposé un algorithme pseudo-polynomial en deux étapes. Ils ont remarqué que l'augmentation du rayon de stabilité a un impact sur les fenêtres de temps, et leur approche de résolution repose sur une méthode dichotomique visant à déterminer un ensemble réalisable de fenêtres de temps avec le rayon de stabilité le plus élevé possible. Chaque étape de la méthode dichotomique nécessite de résoudre une instance du problème de transport. Après la dichotomie, un programme linéaire est résolu pour maximiser le rayon de stabilité avec les fenêtres de temps obtenues. Le chapitre 1 s'appuie également sur la phase de discrétisation et la dichotomie, mais étend le problème en considérant plusieurs cibles et contraintes de communication (transfert des données à la station de base). Il faut donc dès lors trouver des routes pour les données collectées, qui s'appuient sur des capteurs servant de relais, jusqu'à une station de base. La communication impacte fortement les batteries des capteurs et donc les solutions produites. Elle augmente aussi beaucoup la complexité du problème. De plus, nous avons ajouté différents objectifs, visant à proposer des solutions économes en énergie pour préparer le réseau à des futures missions, ce qui est encore une extension de [21].

La recherche de cibles

La recherche de cibles est aussi un domaine qui s'applique aux ensembles de capteurs. Ce sujet de la recherche opérationnelle, plutôt ancien, se retrouve maintenant accessible aux capteurs grâce aux récents progrès technologiques. La recherche de cibles est souvent mise en œuvre, dans la littérature, par des *Unmanned Aerial Vehicle* (UAV), ou véhicules aériens autonomes ou drones en français. Cependant, les UAV peuvent aussi être considérés comme des capteurs mobiles et certains travaux décrivent le même problème en utilisant l'un ou l'autre des termes. Ainsi, on considère, dans cette thèse, les UAV comme des capteurs mobiles et donc la recherche de cibles par UAV comme un sous-domaine de la littérature sur les ensembles de capteurs.

Le premier travail sur la théorie de la recherche date de 1946 avec le rapport *Search and screening* de B.O. Koopman (avec [4], la version étendue publiée en 1980). Ces travaux, effectués pendant la Seconde Guerre mondiale, visaient à rendre plus efficace la localisation de sous-marins ennemis. Depuis lors, le sujet et les applications ont évolués et sortent du domaine purement militaire (par exemple, recherche et sauvetage en mer). La théorie de la recherche est un sujet qui attire beaucoup l'attention de la communauté scientifique ainsi que le montrent les nombreuses études bibliographiques renouvelées toutes les décennies environ (par exemple, [24-27]). Dans cette sous-section, nous donnons un aperçu rapide de la littéra-

ture sur la recherche de cibles, en commençant par la recherche de cibles avec UAV, suivie par des travaux récents sur les jeux de recherche (*Search games* en anglais). Enfin, nous abordons les travaux précédents sur l'allocation des ressources pour la recherche de cibles multicapteurs, pour des cibles statiques puis mobiles.

Les UAV sont de plus en plus accessibles et de plus en plus utilisés dans des applications comme la recherche de cibles. À cette fin, de nombreux protocoles ont été développés pour coordonner dynamiquement un essaim d'UAV à la recherche d'une ou de plusieurs cibles. La mission peut éventuellement être suivie d'une autre tâche lorsqu'une cible est trouvée [28] (par exemple, détruire la cible dans [29] ou lui porter secours). Généralement, contrairement au problème que nous présentons dans le chapitre 3, il n'y a pas de planification à l'avance des activités de recherche ou d'allocation des ressources. Le problème revient souvent à trouver une méthode permettant de prendre des décisions dynamiques sur la direction et la vitesse que doit prendre un drone à un instant donné. Le chemin suivi par les UAV est alors calculé dynamiquement et la probabilité de détection de la cible est directement liée à la position des drones par rapport à la cible. Par exemple, dans [30], l'objectif est de trouver la cible en un temps minimum à l'aide de décisions dynamiques qui sont prises en temps réel. Ces décisions sont des vecteurs d'actions qui vont générer les déplacements futurs des drones. La probabilité d'une détection de cible par un drone est modélisée par une fonction exponentielle décroissante de la distance séparant le capteur, des positions estimées des cibles. Dans [30], les auteurs ont proposé une nouvelle approche, basée sur une récompense heuristique attendue pour réduire le caractère court-terme des décisions prises. La coordination des drones pour la recherche de cibles a également été abordée à l'aide de métaheuristiques d'inspiration biologique qui imitent le comportement des essaims, comme indiqué dans [31]. Par exemple, ALFEO, CIMINO et VAGLINI [32] imposent aux drones une stratégie inspirée par la biologie : les drones déposent des phéromones en des lieux où des observations jugées intéressantes ont été faites [33]. Les mouvements des drones sont régis par la présence de phéromones à leur portée, et par les déplacements de leurs voisins, mimant ainsi le comportement d'essaims naturels qui a inspiré de nombreux algorithmes d'optimisation [34]. Cette stratégie fournit d'abord un aperçu rapide de la zone, suivi d'une meilleure exploration des parties les plus prometteuses.

Les jeux de recherche [27] sont une application de la théorie des jeux [35] à la théorie de la recherche. Dans ce contexte, l'objectif de la cible est d'échapper à ses poursuivants : elle tentera de se dissimuler pendant que les chercheurs la recherchent. Les jeux de recherche sur les graphes et les régions bidimensionnelles sont présentés dans [36]. Comme le montrent ces travaux, la cible peut être soit immobile, soit mobile. Lorsqu'elle est immobile, elle sélectionne un emplacement arbitraire dans l'espace (ou nœud dans un graphe) et s'y cache. L'objectif du chercheur est souvent d'atteindre le plus rapidement l'endroit, où se cache la cible. De nombreux travaux ont été réalisés sur le sujet et le problème est toujours l'objet de publications de

la communauté scientifique. À titre d'exemple récent, GARREC et SCARSINI [37] proposent un problème de jeu de recherche où la cible est immobile dans un réseau stochastique : à tout moment, chaque arête du graphe peut être active ou inactive, suivant une loi de probabilité connue. Les auteurs étudient les valeurs et les stratégies disponibles pour des graphes spécifiques, en utilisant la théorie des jeux de recherche déterministes. Un autre travail récent [38] traite le problème avec une cible immobile, située dans un sous-ensemble connu du réseau. Les problèmes de jeu de recherche sont proches de ceux de "gendarmes et voleurs" de la littérature [39]. Ce sont des problèmes dans les graphes, où l'on cherche à minimiser le nombre de chercheurs pour assurer qu'un fugitif soit capturé (le fugitif peut être trouvé dans un nœud du graphe).

Le problème de la distribution optimale d'un effort de recherche pour détecter une cible statique provient de [4]. Il a été traité il y a quelques années [40, 41] dans le cadre de la division en sous-zones. Dans ce problème, la zone recherchée est divisée en un ensemble de sous-zones disjointes, elles-mêmes divisées en cellules disjointes. L'objectif est de minimiser la probabilité de ne pas avoir détecté la cible à la fin la mission. Le problème de recherche est hiérarchique : le niveau supérieur vise à déterminer la meilleure attribution des capteurs aux sous-zones. Chacun des capteurs est attribué à une sous-zone unique et ne peut ensuite rechercher la cible qu'à l'intérieur de sa sous-zone. Notons que plusieurs capteurs peuvent être attribués à la même sous-zone, et les auteurs de [41] considèrent une attribution injective (1x1) plus facile à résoudre. Au niveau inférieur du problème hiérarchique, on recherche la distribution optimale des ressources pour chaque capteur. Les autres caractéristiques du problème (la fonction objectif, la distribution de la cible dans l'espace,...) sont similaires à celles utilisées dans notre travail (voir la section 3.2). Le niveau inférieur est résolu en utilisant une méthode itérative : à chaque itération, sur la base de [42], on optimise l'allocation des ressources pour un capteur, tandis que l'allocation des autres capteurs est fixée. La méthode itère jusqu'à ce qu'elle converge. Le niveau supérieur est résolu en utilisant une méthode d'entropie croisée (CE pour *cross entropy* en anglais) [43]. À chaque itération, la méthode CE affecte aléatoirement les capteurs à des sous-zones, puis elle évalue l'affectation en utilisant le niveau inférieur de leur algorithme. Ensuite, la méthode met à jour la loi de distribution des affectations des capteurs aux sous-zones et l'itération suivante commence. Ainsi, le niveau supérieur du problème est résolu à l'aide d'une affectation aléatoire dont la distribution est ajustée par les performances des solutions obtenues lors des itérations précédentes. Cette méthode heuristique répète ces étapes jusqu'à convergence de la fonction objectif. LE THI, NGUYEN et DINH [40] abordent le même problème avec une nouvelle formulation mathématique, qui conduit finalement à une reformulation en un programme DC (*Difference of Convex functions*, ou différence de fonctions convexes en français) [44] en utilisant une technique de pénalité exacte. Ensuite, le problème est résolu en utilisant un algorithme DC (DCA) [44]. Des expériences numériques sont présen-

tées sur une instance tirée de [41] sur laquelle de meilleurs résultats sont obtenus.

La répartition optimale de l'effort de recherche est beaucoup plus complexe pour une cible en mouvement. Nous considérons ici le problème en temps discret, où l'horizon du temps est divisé en périodes de même durée. BROWN [45] présente un algorithme progressif-rétrogressif (*forward-backward* en anglais) pour résoudre ce problème. C'est un algorithme qui itère de nombreuses fois jusqu'à convergence, et chaque itération est la résolution du problème période par période (de la première à la dernière). Pour chaque période, l'algorithme utilise des probabilités de détection pour les périodes précédentes et suivantes qui sont mises à jour à chaque itération. Initialement, les probabilités pour les périodes suivantes sont arbitraires et peuvent mal refléter la réelle probabilité de détection qui découlera des décisions qui seront prises dans ces périodes. Mais, après de nombreuses itérations, elles sont affinées à l'aide des solutions calculées et convergent vers les valeurs réelles qui seront obtenues pour les périodes suivantes. Ce type d'algorithme est très répandu dans la littérature pour ces problèmes. Théoriquement, avec un seul capteur, les conditions nécessaires et suffisantes pour une recherche optimale ont été résolues dans [46]. Ces conditions ont été étendues à des contraintes à double couche (une contrainte de ressource pour l'ensemble de l'horizon temporel et des contraintes de ressource pour chaque période) dans [47]. Ce problème, avec déplacement de la cible, a également été abordé dans [41], alors que LE THI, NGUYEN et DINH [40] ont étendu leur travail à ce cas dans [48]. Dans ce nouveau problème, les auteurs recherchent alors une allocation des capteurs aux sous-zones pour chaque période. Un même capteur peut donc chercher dans deux sous-zones différentes mais lors de périodes différentes. Dans les deux travaux, la méthode pour la cible statique a été étendue au cas où la cible est mobile, à l'aide de la méthode progressif-rétrogressif où la prise de décision à chaque période est faite à l'aide de l'algorithme dédié aux cibles statiques. Les algorithmes génèrent donc de nombreuses solutions, où les allocations des capteurs aux sous-zones de chaque période sont obtenues à l'aide de l'algorithme utilisé pour les cibles statiques, et en considérant des distributions de probabilités de détection de la cible dans les autres périodes. Ces probabilités sont mises à jour à chaque itération. Avec suffisamment d'itération effectuées, l'algorithme converge et s'approche de la solution optimale.

Recherche et suivi de cibles

On trouve également dans la littérature des problèmes d'optimisation abordant conjointement la recherche et le suivi de cibles. Cependant, ces travaux relèvent davantage de la thématique de recherche de cibles avec UAV et abordent peu les problématiques du suivi de cibles présentés plus haut dans cette section.

Une partie des travaux de ce domaine est consacré à la maximisation de deux objectifs contradictoires [49]. Le premier objectif consiste à maximiser le nombre de cibles trouvées et

le deuxième vise à maximiser la durée de surveillance de ces cibles (c'est-à-dire à collecter le plus de données possibles sur ces dernières). Ainsi, il faut trouver un compromis entre l'exploration de la zone pour trouver plus de cibles, et le suivi des cibles déjà détectées. LI, PITRE, JILKOV et CHEN [49] présentent notamment une nouvelle métrique pour évaluer les performances d'une solution, à l'aide d'une somme pondérée des objectifs obtenus en sommant les données récoltées sur les cibles trouvées. L'horizon de temps de la mission est discrétisé, et les auteurs ajoutent, dans leur fonction objectif, des coefficients qui permettent de donner un ordre d'importance entre les cibles et entre les instants où les cibles sont surveillées. Par exemple, il est supposé que surveiller une cible plus tôt est plus intéressant que plus tard. Les auteurs ajoutent aussi à leur métrique un facteur permettant de prendre en compte la survivabilité de leurs UAV, c'est-à-dire, le risque que les appareils soient détruis en suivant des chemins trop dangereux. Les auteurs ont ensuite étendu leur travail à l'aide de simulations réalistes [50] et d'une méthode de résolution utilisant les principes d'optimisation en essaim [51]. Pour résumer, c'est un problème qui est focalisé sur le calcul des déplacements des drones et dont la contribution à la fonction objectif dépend de ce que les drones captent durant leur vol.

Dans des travaux comme [52], la mission requiert qu'une fois trouvée, une cible doit être suivie par plusieurs capteurs. Ici, l'objectif du suivi pour un UAV est de garder un déplacement synchronisé et de minimiser sa distance relative avec la cible. Les travaux de [52] se focalisent sur les algorithmes implémentés dans les UAV qui permettent leur navigation autonome et l'accomplissement de leur mission. Les algorithmes traitent donc le décollage, le vol vers la zone de recherche, la recherche et le suivi. Encore une fois, ces problèmes de recherche et de suivi sont focalisés sur le calcul des déplacements (directions, points de passage,. . .) des UAV, et se rapprochent donc plutôt de la littérature sur la recherche de cible.

Publications

Le contenu de cette thèse été sujet aux communications et publications suivantes :

— Delavernhe, F., Lersteau, C., Rossi, A., & Sevaux, M. (2018, Février). Ordonnancement robuste d'un réseau de capteurs sans fils pour le suivi de cibles mobiles avec communications. Dans ROADEF : Recherche Opérationnelle et d'Aide à la Décision.

— Delavernhe, F., Lersteau, C., Rossi, A., & Sevaux, M. (2019, Février). Ordonnancement réactif pour le suivi de cibles mobiles : de la robustesse à la garantie de performance en ligne. Dans ROADEF : Recherche Opérationnelle et d'Aide à la Décision.

— Delavernhe, F., Jaillet, P., Rossi, A., & Sevaux, M. (2020, Février). Planification de la recherche d'une cible par plusieurs capteurs avec considération du coût de déplacement. Dans ROADEF : Recherche Opérationnelle et d'Aide à la Décision.

— Delavernhe, F., Lersteau, C., Rossi, A., & Sevaux, M. (2020). Robust scheduling for

target tracking using wireless sensor networks. Computers & Operations Research, 104873.

Plan de la thèse

Le plan de cette thèse est le suivant. Dans le chapitre 1, nous abordons le problème de surveillance de cible comme présenté précédemment dans la section *Littérature* et dans [21]. Nous étendons le problème de robustesse au cas multi-cibles avec transfert des données vers une station de base. Toujours dans le chapitre 1, nous proposons une méthode pour produire des ordonnancements qui optimisent la robustesse et des considérations énergétiques. Les performances de cette méthode sont étudiées dans ce chapitre. Dans le chapitre 2, nous étendons le problème présenté dans le chapitre 1 en proposant une méthode qui ajuste l'ordonnancement robuste décrit auparavant aux cas où cette robustesse n'est plus suffisante. Cette nouvelle méthode est dynamique et gère la détection de retards et d'avances et l'adaptation des activités des capteurs pour permettre une surveillance continue. Les performances de cette méthode sont garanties par un rayon de stabilité dynamique qui évolue avec le temps et son temps de calcul a été étudié. Dans le chapitre 3, nous abordons le problème de recherche de cibles, présenté plus tôt dans la littérature. Ici, une nouvelle formulation du problème est présentée. Elle est plus générale et plus réaliste mais complexifie sa résolution. Nous y proposons une méthode de résolution qui construit pas à pas une solution, qui n'est pas basée sur un algorithme progressif-rétrogressif. Le chapitre final est la conclusion, où sont présentées des perspectives avancées de nos travaux. Certaines perspectives, sur la surveillance de cibles, font déjà l'objet de premiers résultats et sont présentées avec plus de détails.

CHAPITRE 1

ORDONNANCEMENT ROBUSTE POUR LE SUIVI DE CIBLES MOBILES AVEC COMMUNICATIONS

1.1 Introduction

Nous considérons dans ce chapitre, la mission de surveillance d'un ensemble de cibles à l'aide d'un ensemble de capteurs statiques [9]. Dans cette mission, l'ensemble de capteurs est aléatoirement réparti dans une zone sans infrastructure, afin de surveiller (c'est-à-dire enregistrer des données et les transmettre à une station de base) un ensemble de cibles mobiles. Le regroupement et traitement des données collectées étant important, les capteurs sont organisés sous la forme d'un réseau où chaque nœud est un capteur qui peut collecter, transmettre et recevoir de l'information.

Dans le réseau, une station de base est déployée et représente le « puits », c'est-à-dire la destination des données collectées par les capteurs. La station de base a ensuite la possibilité de transmettre les données récoltées en dehors de la zone de déploiement pour traitement ultérieur par des décideurs. Ce type de mission peut se retrouver, par exemple, dans le contexte militaire où un réseau de capteurs sans fil est déployé sur un champ de bataille, et où plusieurs cibles, ennemies ou alliées, traversant le champ de bataille doivent être surveillées.

Une fois activé en mode surveillance, un capteur ne peut collecter de données que sur les cibles qui se situent à l'intérieur de son rayon de surveillance, et enregistre des données en continu. La portée de surveillance, ou portée de détection, est la distance maximale entre le capteur et une cible qu'il peut surveiller. Ainsi, l'objectif fondamental de ce problème est de trouver un ordonnancement d'activation, qui alternera les états actifs et inactifs des capteurs (pour des raisons d'économie d'énergie), et qui permettra une surveillance continue des cibles. L'utilisation de ce type d'ordonnancement prolongera fortement la durée de vie du réseau par rapport à une activation continue de tous les capteurs [53].

En outre l'ordonnancement doit aussi garantir le transfert des données collectées vers la station de base. Il doit donc prévoir toutes les activités de transfert et réception de données des

capteurs en plus de la surveillance complète et continue des cibles. Si à un instant, une cible n'est pas surveillée par un capteur, avec transfert des données à la station de base, alors on considère que la mission a échoué. On remarque que la faisabilité de la mission et la durée de vie du réseau sont limitées par les portées de surveillance, de communication ainsi que par la capacité des batteries des capteurs.

L'activité de détection d'un ensemble donné de cibles est appelée une mission. À notre connaissance, les travaux de la littérature consacrés à l'optimisation de la durée de vie des réseaux de capteurs sans fil sont restreints aux missions de surveillance d'une seule cible en mouvement. Le problème abordé dans ce chapitre vise à produire un ordonnancement qui alterne les activités des capteurs pour surveiller plusieurs cibles. L'objectif principal est de maximiser la capacité de l'ordonnancement résultant à rester réalisable malgré l'incertitude affectant la vitesse des cibles le long de leurs trajectoires. L'objectif secondaire est de maximiser la capacité du réseau de capteurs sans fil à surveiller un ensemble donné de zones d'intérêt pour de futures missions. Le dernier objectif consiste à minimiser la quantité totale d'énergie requise par la mission de surveillance courante. Ces trois objectifs sont abordés dans un ordre lexicographique, la solution est donc produite d'abord par une approche robuste puis ensuite dans une optique d'économie d'énergie.

Dans le cas où les cibles sont des véhicules terrestres circulant sur des routes ou des voies, leur trajectoire géographique peut être prédite tout naturellement (par exemple, un train ne quittera pas ses rails). Pour les bateaux ou les avions, la trajectoire peut être estimée au moins pour une courte durée. Dans ce travail, nous supposons qu'une telle prédiction de trajectoire est disponible, mais l'incertitude considérée concerne la vitesse à laquelle les cibles se déplacent le long de leur trajectoire.

Les principales contributions de ce chapitre sont les suivantes. Tout d'abord, un modèle plus réaliste que celui présenté dans [21] est envisagé pour le réseau de capteurs sans fil, où la communication des données produites par les capteurs est prise en compte dans le nouveau modèle. Dans la plupart des applications de réseau de capteurs, la communication des données collectées est obligatoire et comme la consommation électrique due à la communication est beaucoup plus importante que celle qui est imputable à la détection [54], le nouveau modèle proposé dans cet article est beaucoup plus réaliste. En outre, le travail précédent [21] est également étendu pour prendre en compte plusieurs cibles dans le problème. En conséquence, nous introduisons une nouvelle borne supérieure sur le rayon de stabilité défini à la section 1.5, et les bornes précédentes introduites dans [21] sont naturellement étendues pour inclure la communication et le suivi de plusieurs cibles. Le rayon de stabilité [55] est une mesure de la capacité d'un ordonnancement à demeurer réalisable malgré l'incertitude. Deuxièmement, ce chapitre propose une approche étendue et globale pour produire des solutions robustes, affinées avec deux critères supplémentaires : les résultats de [21] sont étendus à

plusieurs cibles, et la communication *multi-hop* (communication entre capteurs pour transférer les données collectées vers une station de base) est prise en compte. Enfin, cette approche prend en compte des considérations énergétiques en plus de la robustesse, avec des zones prioritaires, et plusieurs niveaux de priorité.

Ce chapitre est organisé comme suit : une définition formelle du rayon de stabilité est donnée à la section 1.2. La section 1.3 définit le problème. Ensuite, nous présentons la phase de discrétisation, utilisée en préambule à la méthode de résolution, à la section 1.4 et proposons une borne supérieure pour le rayon de stabilité à la section 1.5. La section 1.6 présente la méthode de résolution. La section 1.7 présente les expériences et leurs résultats et enfin, la section 1.8 conclut ce chapitre.

1.2 Rayon de stabilité

Une définition formelle du rayon de stabilité d'un programme linéaire peut être trouvée dans [55]. Dans le problème de planification, le rayon de stabilité est un indicateur de la plus grande variation des temps de traitement des travaux pour lesquels l'ordonnancement optimal conserve son caractère optimal. Dans de tels problèmes, chaque tâche i de l'ensemble de tâches Q a un temps de traitement p_i et, dans un ordonnancement s, une date de fin $c(s)_i$. Des événements incertains peuvent affecter le temps de traitement d'une tâche i avec une variation de ϵ_i de telle sorte que le nouveau temps de traitement soit égal à $p_i \pm \epsilon_i$, ce qui affectera également la date de fin. Un ordonnancement a un rayon de stabilité de ϱ si et seulement si l'ordonnancement reste optimal pour tous les vecteurs de temps de traitement $p' \in O_\varrho(p)$, avec $O_\varrho(p)$ la boule fermée centrée sur $p = \{p_1, p_2, \ldots, p_{|Q|}\}$ les temps de traitement prévisionnels et avec un rayon égal à ϱ. Autrement dit, cela signifie que l'ordonnancement reste optimal si pour chaque tâche i, $\epsilon_i \leq \varrho$.

Après avoir introduit le problème dans la section suivante, la définition originale du rayon de stabilité est adaptée au problème de suivi de cibles à la section 1.4.

1.3 Définition du problème

Cette section introduit le problème du suivi de plusieurs cibles par un réseau de capteurs sans fil, avec prise en compte de l'acheminement des données vers une station de base. La communication par saut (*hop communication*) signifie qu'un capteur qui ne peut pas communiquer directement avec la station de base peut lui faire parvenir des données via des capteurs intermédiaires, qui servent de relais. Ce problème est noté P_c^n où n est le nombre de cibles et c signifie « communication ». Dans ce problème, pendant un horizon temporel donné T, un ensemble J de n cibles dont la trajectoire spatiale est connue, traverse une zone surveillée par

un ensemble I de m capteurs. La fonction $\tau_j(t)$, connue, estime la position de la cible j à tout instant t.

Dans ce problème, le réseau doit garantir un suivi constant de chaque cible. Si la trajectoire d'une cible n'est pas en permanence à la portée de surveillance d'au moins un capteur actif, la mission de surveillance est mise en échec. Notons que si ces cibles hors de portées de surveillance sont ignorées, la mission réduite aux cibles restantes est réalisable. Deuxièmement, le réseau doit transférer toutes les données collectées vers une station de base connectée à une source d'énergie permanente et capable de transmettre les données à un centre de contrôle à distance. À cet effet, les capteurs sont équipés de modules de communication pour transmettre et recevoir des données. Une fois activés, ils peuvent former un chemin entre les capteurs de surveillance et la station de base, avec plusieurs capteurs utilisés comme relais si nécessaire.

Les capteurs ne peuvent communiquer avec d'autres capteurs ou avec la station de base que s'ils se trouvent sous leur portée de communication R_c. Un capteur peut surveiller uniquement les cibles qui se trouvent sous sa portée de détection R_s, c'est-à-dire qu'à un instant t, les cibles qui sont à plus de R_s mètres d'un capteur i ne peuvent pas être surveillées par ce capteur. $N(i)$ est l'ensemble de tous les capteurs à portée de communication du capteur i, c'est-à-dire que cet ensemble comprend tous les capteurs qui peuvent envoyer et recevoir des données du capteur i. Chaque capteur $i \in I$ a une batterie limitée avec une énergie de E_i joules.

Les capteurs sont multi-rôles [56] et nous considérons donc trois types de consommation d'énergie, associés à ces rôles :

— La surveillance d'une cible qui nécessite une puissance de p^S watts (un watt est un joule par seconde)
— L'émission de données qui nécessite p^T watts.
— La réception de données émises par un autre capteur qui nécessite p^R watts.

Toute activité de surveillance opérée par un capteur génère des données qui doivent être transmises à la station de base. Ainsi, si un capteur i surveille une cible pendant s secondes, i transmettra également s secondes de données, donc i consommera $(p^S + p^T) \times s$ joules. De même, dans une activité de relais, si le capteur i reçoit s secondes de données, i retransmettra s secondes de données, et consommera donc $(p^R + p^T) \times s$ joules. Par conséquent, la surveillance d'une cible pendant s secondes, extrait $(p^S + p^T) \times s$ joules de la batterie du capteur qui opère la surveillance et $(p^R + p^T) \times s$ joules de chaque batterie des capteurs utilisés pour transmettre les données collectées à la station de base. Ces activités (détection, réception et transmission) peuvent être opérées simultanément par un capteur. De plus, si un capteur surveille simultanément $x \geq 1$ cibles, il consomme également x fois sa puissance de surveillance. Par exemple, à un instant $t \in T$, si un capteur surveille deux cibles et reçoit des

données d'un autre capteur, il transmettra les données de ces trois activités. Par conséquent la consommation d'énergie instantanée de ce capteur est de $2p^S + p^R + 3p^T$ watts. C'est-à-dire qu'il consomme de l'énergie pour la surveillance des deux cibles, pour recevoir des données et pour transmettre les données de trois cibles puisqu'il transmet à la fois les données qu'il collecte directement et les données reçues d'un autre capteur. Par conséquent, un capteur qui ne capte, ni ne transmet, ni ne reçoit, ne consomme pas d'énergie. Des modèles de consommation d'énergie plus détaillés et complexes sont présentés dans [57, 58].

Une solution à long terme du problème devrait préserver la capacité du réseau à répondre aux futures missions de surveillance. Par conséquent, dans la suite, nous n'activons qu'un seul capteur à la fois pour surveiller une même cible, car l'activation de plusieurs capteurs ne fera que collecter des données redondantes et gaspiller de l'énergie. Cependant, plusieurs capteurs peuvent être activés au même instant pour la tâche de transmission, ou pour surveiller différentes cibles. En outre, nous considérons également des zones prioritaires, également appelés points chauds dans la littérature relative à la couverture [59]. L'objectif est de préserver et d'équilibrer les capacités résiduelles des batteries pour les futures missions de suivi de cibles [60]. Les décideurs définissent de multiples zones prioritaires où l'ordonnancement doit conserver autant de capacité de surveillance que possible. De plus, parce que toutes les zones peuvent ne pas être considérées d'égale importance, les gestionnaires du réseau définissent un rang $\ell \in C$ pour chaque zone, où C est l'ensemble des rangs et r est le rang maximum. Le rang d'une zone est fixe, il n'évolue pas pendant la mission courante, et plus le rang est élevé, plus les capacités de surveillance du réseau dans la zone doivent être préservées. Le classement exprime une préférence pour la préservation de l'énergie dans les zones considérées. Nous traitons le problème comme un problème multi-objectif, avec un ordre lexicographique des objectifs.

L'objectif principal est de trouver un ordonnancement d'activation des capteurs qui satisfait toutes les contraintes (surveiller en permanence les cibles, transférer les données, respecter les contraintes de limitation de l'énergie disponible induite par les batteries des capteurs) et qui maximise le rayon de stabilité. La notion générale de rayon de stabilité est définie et adaptée au problème à la section 1.4. Ainsi, l'ordonnancement reste réalisable même si les cibles sont en retard ou en avance à un point quelconque de leur trajectoire, à condition que l'avance ou le retard reste inférieur ou égal au rayon de stabilité.

La figure 1.1 donne un exemple d'ordonnancement pour la surveillance de deux cibles. La cible 1 est couverte par deux activités, la première étant opérée par le capteur 1 activé à l'instant 0 et se terminant à l'instant 8. La deuxième activité, opérée par le capteur 3, commence dès la fin de la première activité et se termine à l'instant 15. En plus d'un tel ordonnancement, la solution doit prévoir le transfert des données qui seront collectées au cours de ces activités, soit le transfert de $15 + 15 = 30$ secondes de données captées sur les deux cibles.

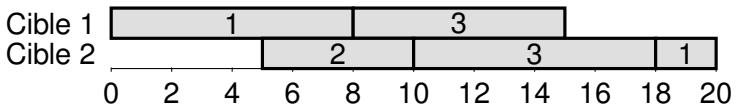

FIGURE 1.1 – Exemple d'ordonnancement pour deux cibles

$I = \{1, \ldots, m\}$	Ensemble de capteurs
$J = \{1, \ldots, n\}$	Ensemble de cibles
E_i	Énergie initiale de la batterie du capteur i
p^S	Puissance consommée en watts pour la surveillance
p^R	Puissance consommée en watts pour la réception de données
p^T	Puissance consommée en watts pour la transmission de données
R_c	Portée de la communication des capteurs
R_s	Portée de la surveillance des capteurs
T	Horizon de temps
$C = \{1, \ldots, r\}$	Ensemble des rangs des zones
$\tau_j(t)$	Position de la cible $j \in J$ à l'instant $t \in [0, T]$
$N(i)$	Ensemble de tous les capteurs à portée de communication du capteur i

Tableau 1.1 – Résumé de toutes les notations

Le deuxième objectif consiste à maximiser la durée minimale de couverture (surveillance + transmission des données) garantie dans les zones prioritaires pour chaque rang $\ell \in C$, tout en respectant la priorité exprimée par les rangs des zones. Cela signifie que le temps de surveillance garanti disponible n'importe où dans une zone prioritaire de rang r est maximisé, puis, parmi toutes les solutions qui offrent cette couverture garantie, nous maximisons la garantie de couverture partout dans les zones prioritaires de rang $r - 1$, et ainsi de suite jusqu'au rang 1.

Enfin, le troisième objectif vise à minimiser la quantité totale d'énergie requise par la mission courante, tout en maintenant tous les objectifs précédents à leurs valeur optimale.

Le tableau 1.1 résume les notations qui seront utilisées dans tout le manuscrit.

1.4 Discrétisation

Les données d'entrées du problème sont essentiellement de nature géographique. Les capteurs avec leurs caractéristiques et leur position, sont déployés dans une zone, avec les zones prioritaires, les cibles et leur itinéraire. Cependant, pour déterminer l'ordonnancement de l'activité des capteurs et le routage des données collectées, l'instance du problème doit être discrétisée. La discrétisation est basée sur [21], et a été étendue au cas de cibles multiples, avec des communications et des zones prioritaires. Prenons l'exemple de la figure 1.2, où les disques jaunes représentent la portée de surveillance de trois capteurs, la flèche noire l'itinéraire d'une cible, B la station de base et les disques gris modélisent les portées de communication. Deux

zones prioritaires sont représentées par des polygones verts, elles ont deux rangs différents, la plus petite zone étant la plus prioritaire.

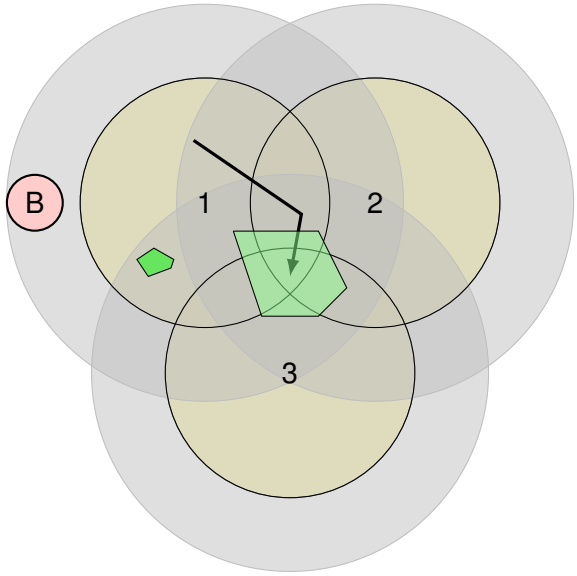

FIGURE 1.2 – Un exemple à trois capteurs

Une *face* est définie comme le lieu des points du plan surveillés par le même sous-ensemble de capteurs (dans la figure 1.2, la face $\{1, 2, 3\}$ est l'ensemble de tous les points dans l'espace qui se trouvent à portée des capteurs 1, 2 et 3). Chaque face f est associée à un ensemble de capteurs candidats $S(f)$. Autrement dit, tout capteur pouvant surveiller f est dans $S(f)$. La phase de discrétisation transforme la trajectoire de chaque cible en une séquence de faces visitées, car cela permet de faciliter la prise de décisions. Il suffit de sélectionner un ou plusieurs capteurs dans l'ensemble des capteurs candidats d'une face f pour surveiller la cible lorsqu'elle traverse f. L'instant où une cible passe d'une face à une autre s'appelle un tick. Lorsqu'une cible entre à portée d'un capteur, elle définit un tick *entrant* ; et un tick *sortant* lorsqu'elle quitte la portée d'un capteur. t_k^j est le k-ième tick de la cible j qui indique également sa date d'apparition, et σ_k^j est une valeur entière qui est égale à $+1$ si le tick est *entrant*, et -1 s'il s'agit d'un tick *sortant*. Par convention, le premier et dernier tick sont des ticks sortant et entrant respectivement [21]. Une fenêtre de temps Δ_k^j est la durée entre deux ticks successifs t_k^j et t_{k+1}^j.

Dans l'exemple de la figure 1.2, seules trois faces sont prises en compte (les autres ne sont pas visitées par la cible) : $\{1\}$, $\{1, 2\}$ et $\{1, 2, 3\}$. Par conséquent, cela définit quatre ticks comme présenté dans le tableau 1.2.

Pour les zones prioritaires et leurs rangs donnés, le problème est discrétisé comme suit. Soit un rang, dont la priorité est représentée par un entier $\ell \in \{1, \ldots, r\}$, où r est le nombre de rangs différents. Partout où une cible potentielle apparaît dans une zone prioritaire dont le

Face	{1}	{1,2}	{1,2,3}	
Taille fenêtre de temps	$\Delta_0^1 = 5$	$\Delta_1^1 = 10$	$\Delta_2^1 = 2.5$	
Tick	$t_0^1 = 0$ $\sigma_0^1 = -1$	$t_1^1 = 5$ σ_1^1 = +1	$t_2^1 = 15$ σ_2^1 = +1	$t_3^1 = 17.5$ σ_3^1 = +1

Tableau 1.2 – Les quatre ticks de la figure 1.2

rang est ℓ, le réseau doit fournir le même temps de surveillance garanti. Pour chaque rang $\ell \in \{1, \ldots, r\}$, $\mathcal{F}(\ell)$ est l'ensemble de toutes les faces qui ont une intersection non vide avec une zone de rang ℓ, et aucune intersection avec une zone de rang supérieur. En effet, si une face a une intersection non vide avec deux zones de rangs ℓ et ℓ' avec $\ell < \ell'$, cette face fait partie de $\mathcal{F}(\ell')$ uniquement car ℓ' est le rang le plus élevé. L'ensemble de tous les capteurs pouvant surveiller au moins une face dans $\mathcal{F}(\ell)$ est désigné par $T(\ell)$. Plus formellement, $T(\ell) = \cup_{f \in \mathcal{F}(\ell)} S(f)$ pour tous $\ell \in \{1, \ldots, r\}$. Afin de garantir la même durée de surveillance partout où une cible potentielle pourrait se trouver dans une zone de rang ℓ, nous garantissons la même durée de surveillance pour chaque face $f \in \mathcal{F}(\ell)$. Par conséquent, les zones prioritaires et rangs sont discrétisés en ensembles de faces $\mathcal{F}(\ell)$ pour chaque rang ℓ. Par exemple, dans la figure 1.2, si la plus petite zone a le rang 2 et l'autre a le rang 1, alors $\mathcal{F}(1) = \{\{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}, \}$ et $\mathcal{F}(2) = \{\{1\}\}$. La raison pour laquelle la face $\{1\}$ n'est pas dans $\mathcal{F}(1)$ est qu'elle appartient à $\mathcal{F}(2)$. Les capteurs candidats pour les deux rangs sont $T(1) = \{1,2,3\}$ et $T(2) = \{1\}$, on peut voir que le même capteur peut apparaître dans différents ensembles $T(\ell)$. Enfin, pour un rang ℓ donné, nous pouvons réduire $\mathcal{F}(\ell)$ car un temps garanti dans une face est également garanti dans toutes les faces pour lesquelles les capteurs candidats sont des sur-ensembles de l'ensemble des capteurs candidats dans cette face. Donc, chaque face f dans $\mathcal{F}(\ell)$ n'est conservée que si il n'existe pas de face f' initialement dans $\mathcal{F}(\ell)$, tel que $S(f') \subset S(f)$. En effet, si un temps de couverture t est garanti avec un ensemble de capteurs s, alors chaque face couverte par un sur-ensemble s' de s peut être couverte au moins t unités de temps. Dans l'exemple de la figure 1.2, nous avons $\mathcal{F}(1) = \{\{2\}, \{3\}\}$ et $\mathcal{F}(2) = \{\{1\}\}$ après réduction. $\mathcal{F}(1)$ est réduit à seulement deux faces car, par exemple, la face $\{1,2,3\}$ en a été exclue car elle peut être couverte au moins aussi longtemps que la face $\{2\}$.

La communication entre les capteurs est représentée par le graphe orienté de communication $\vec{G}_c = (I \cup \{B\}, A)$ avec B la station de base et A les paires de capteurs qui peuvent communiquer directement. Pour toutes les données collectées, un flot du capteur de surveillance vers la station de base est calculé dans $\vec{G}_c$. Avec l'exemple de la figure 1.2, nous obtenons le graphe $\vec{G}_c$ de la figure 1.3.

Le rayon de stabilité d'un ordonnancement réalisable de l'activité des capteurs est une mesure de sa capacité à maintenir une couverture complète des cibles malgré leurs avances

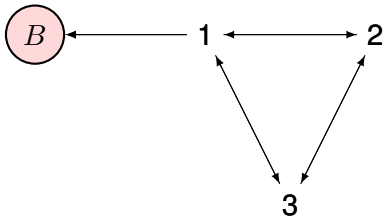

FIGURE 1.3 – Le graphe orienté de communication $\vec{G}_c$ associé à l'exemple de la Figure 1.2

et leurs retards. Plus précisément, nous supposons que la phase de discrétisation transforme la trajectoire de la cible $j \in J$ en une suite de $K_j \geq 1$ fenêtres de temps, et que la cible doit passer Δ_k^j unités de temps dans la fenêtre de temps k, pour tous $k \in \{1, \ldots, K_j\}$.

— $\vec{\zeta} = (\zeta_{jk} \in \mathbb{R})_{j \in J, k \in K_j}$ est l'ensemble des avances et retards possibles, *i.e.*, ζ_{jk} est l'écart entre le temps réellement passé par la cible j dans la fenêtre de temps k, et le temps de séjour estimé Δ_k^j. $\zeta_{jk} < 0$ correspond à une avance et $\zeta_{jk} > 0$ à un retard. Ainsi $\zeta_{jk} \geq -\Delta_k^j$ pour tous $j \in J$ et pour tout $k \in K_j$, car une cible ne peut pas passer un temps strictement négatif dans une fenêtre de temps.

— $\Omega(\Lambda)$ est l'ensemble des ordonnancements réalisables pour Λ, où Λ est l'ensemble de tous les λ_{jk} tels que la cible j séjourne λ_{jk} unités de temps dans la fenêtre de temps k, pour tout $j \in J$ et pour tout $k \in K_j$.

Soient $B(\varepsilon) = \{\zeta \in \vec{\zeta} |\ ||\zeta||_\infty \leq \varepsilon\}$, et Θ l'ensemble de tous Δ_k^j. Le rayon de stabilité de l'ordonnancement $\mathcal{S}$ peut être écrit comme

$$\rho(\mathcal{S}, \Theta) = \max\{\varepsilon > 0 | \forall \zeta \in B(\varepsilon), \mathcal{S} \in \Omega(\Theta + \zeta)\}$$

Par conséquent, le rayon de stabilité $\rho(\mathcal{S}, \Theta)$ de l'ordonnancement $\mathcal{S}$ est la quantité maximale de retards ou d'avances que les cibles peuvent avoir dans chaque fenêtre de temps, sans compromettre la capacité de $\mathcal{S}$ à assurer une couverture complète de toutes les cibles.

Les nouvelles notations introduites dans cette section sont rappelées dans le Tableau 1.3.

1.5 Bornes supérieures

Dans cette section, nous présentons différentes bornes supérieures du rayon de stabilité. La borne supérieure qui sera utilisée dans la méthode de résolution, notée UB, est la plus petite borne parmi trois bornes supérieures (c'est-à-dire que $\mathrm{UB} = \min(\mathrm{UB}_1, \mathrm{UB}_2, \mathrm{UB}_3)$). Deux d'entre elles sont des extensions naturelles de [21] et sont présentées à la section 1.5.1, la dernière est spécifique aux communications, elle est introduite à la section 1.5.2.

t_k^j	Instance du tick k de la cible j
σ_k^j	$\begin{cases} +1 & \text{si le tick } k \text{ de la cible } j \text{ est entrant,} \\ -1 & \text{sinon.} \end{cases}$
$T(\ell)$	Sous-ensemble de capteurs candidats qui peuvent surveiller une cible dans une face de rang ℓ
r	Nombre total de rang pour les zones
$\mathcal{F}(\ell)$	Ensemble réduit de toutes les faces qui ont une intersection non vide avec une zone de rang ℓ et une intersection vide avec les autres rangs $\ell' > \ell$
$S(f)$	Ensembles de capteurs couvrant la face f
K_j	Nombre de fenêtres de temps pour la cible j
Δ_k^j	Temps passé par la cible j dans sa $k$ème face

Tableau 1.3 – Résumé des notations introduites dans la discrétisation

1.5.1 Définition de deux bornes supérieures générales sur le rayon de stabilité

Nous visons à produire un ordonnancement de l'activité des capteurs qui maximise la valeur du rayon de stabilité, afin d'offrir une protection maximale contre les retards et les avances. Les bornes supérieures sont utiles pour atteindre cet objectif et font partie de l'approche de résolution introduite à la section 1.6.

Les deux bornes présentées dans [21] sont toutes deux construites en considérant ce qui peut limiter l'extension de la durée des fenêtres de temps. La première, notée UB_1, est basée l'ensemble des capteurs candidats communs à deux fenêtres de temps, et la seconde, UB_2, exploite les limites introduites par la capacité des batteries des capteurs. Ces deux bornes supérieures du rayon de stabilité, qui ont été initialement introduites pour une seule cible sans communication, sont étendues au cas des cibles multiples et prennent désormais également en compte la consommation d'énergie pour la communication. Ainsi, nous obtenons deux bornes plus serrées :

$$\mathrm{UB}_1 = \min_{\substack{j \in J \\ (k,k') \in K_j^2}} \left\{ \frac{1}{2} \left(\frac{\sum_{i \in S_j(k) \cap S_j(k')} E_i}{p^S + p^T} + t_{k'}^j - t_{k+1}^j \right) | k < k' \right\}$$

où $S_j(k)$ est l'ensemble des capteurs candidats pour la cible j pendant la fenêtre de temps k. $t_{k'}^j$ est le tick délimitant le début de la fenêtre de temps k' et t_{k+1}^j le tick de fin associé à la fenêtre de temps k. L'idée de cette borne est de limiter, pour chaque paire de fenêtre de temps, la valeur du rayon de stabilité par le temps de surveillance ($p^S + p^T$) disponible dans les capteurs qui sont communs à ces fenêtres de temps.

$$\mathrm{UB}_2 = \min_{f \in F} \left(\frac{\sum\limits_{i \in S(f)} E_i}{2 \times (p^S + p^T) \times n_f} \right)$$

où F est l'ensemble des faces visitées par les cibles, et n_f le nombre de fois où la face f est visitée (c'est-à-dire, le nombre de fois où une cible entre dans la face f). L'idée de cette borne est que le rayon de stabilité est limité par la somme des capacités des capteurs. En effet, en augmentant le rayon de stabilité, chaque face traversée est couverte plus longtemps et nécessite donc plus d'énergie pour sa couverture.

1.5.2 Borne supérieure basée sur le graphe de communication

Dans cette section, nous introduisons une nouvelle borne supérieure sur le rayon de stabilité basée sur la consommation énergétique imputable aux activités de communication, et de détection. En effet, les bornes précédentes ont été introduites dans un contexte où la communication n'était pas prise en compte. Les extensions apportées à ces bornes dans ce travail ne prennent pas en compte les tâches de relais de données, c'est-à-dire, la réception et le transfert de données à partir d'autres capteurs. Ces extensions ne prennent en compte que le transfert des données par les capteurs qui les ont collectées. Cependant, lorsque des capteurs intermédiaires sont utilisés, ce rôle peut avoir un impact significatif sur leur consommation. Étant donné que les capteurs peuvent avoir besoin de plusieurs relais pour envoyer les données à la station de base et considérant que les puissances d'émission et de réception ne sont pas négligeables par rapport à la surveillance, les performances de l'ordonnancement dépendront fortement des routes disponibles pour transférer les données.

Dans le contexte de l'application, certains sous-ensembles de capteurs devront être des points de passage obligés pour les données qui doivent atteindre la station de base, généralement les capteurs entourant la station de base. Pour chacun de ces ensembles de capteurs, l'augmentation de la durée de la mission du réseau (augmentation du rayon de stabilité) induit plus de données à transmettre, donc plus de consommation d'énergie. Ainsi, il est possible que l'un de ces ensembles de capteurs limite la valeur maximale atteignable par le rayon de stabilité. Ainsi, la somme des capacités des batteries de tous les capteurs de cet ensemble induit une borne supérieure sur le valeur du rayon de stabilité pour tout ordonnancement réalisable. Par conséquent, pour chaque face f, nous déterminons l'ensemble des capteurs pour lesquels toutes les données collectées nécessiteront d'être relayées, cet ensemble est appelé ensemble *restrictif* de f.

De plus, nous notons que tout ordonnancement réalisable couvre au moins le cas dans lequel aucune cible n'est en avance ni en retard. Dans un tel cas, le calcul de la quantité minimale de données qui doivent être collectées dans une face est facile. En effet, cette quantité est

égale au temps passé par les cibles dans la face en question. Ainsi, nous calculons également la quantité minimale de données collectées qui est ensuite transmise par tous les ensembles de restrictifs. Nous obtenons donc une borne supérieure sur le rayon de stabilité en fonction du niveau de la batterie de tous les ensembles restrictifs et de la quantité minimale de données collectée.

Nous rappelons que le graphe de communication $\vec{G}_c = (I \cup \{B\}, A)$ représente toutes les possibilités pour la transmission de données entre les capteurs et la station de base (voir section 1.4). Soit f une face visitée par au moins une cible pendant une durée non nulle, ainsi les données collectées dans f nécessitent des capteurs pour recevoir et transmettre ces données pour $\sum_{j \in J} \left(2\nu_f^j \rho + \sum_{\substack{k \in K_j \\ S_j(k) = S(f)}} \Delta_k^j \right)$ unités de temps, où ν_f^j est égal à un si et seulement si la cible j visite la face f, et zéro sinon. En effet, si deux cibles différentes visitent la même face, elles pourraient toutes deux passer 2ρ d'unités de temps supplémentaire sur cette face, avec ρ le rayon de stabilité. 2ρ est le temps supplémentaire maximal qu'une cible peut passer dans une face tout en étant couverte par un ordonnancement avec un rayon de stabilité ρ. Cette transmission de données est toujours possible à condition que pour tout ensemble d'articulation du graphe $V_f \subset I$ de $\vec{G}_c$ qui partitionne $I \cup \{B\} \backslash V_f$ en deux composantes connexes (la première contenant B, et la seconde contenant $S(f)$, les capteurs de f), l'énergie des capteurs de V_f soit suffisante pour assurer la réception et la transmission des données, ce qui peut être formulé comme suit :

$$\sum_{j \in J} \left(2\nu_f^j \rho + \sum_{\substack{k \in K_j \\ S_j(k) = S(f)}} \Delta_k^j \right) (p^R + p^T) \leq \sum_{i \in V_f} E_i$$

Afin d'obtenir la borne supérieure la plus serrée possible sur ρ, l'ensemble d'articulation V_f peut être tel que la somme de l'énergie des batteries soit minimale. De plus, l'inégalité ci-dessus peut être renforcée en considérant que V_f divise les sommets de $I \cup \{B\} \backslash V_f$ en deux ensembles X_B et X_f. X_B est l'ensemble de sommets qui contient B, et X_f celui qui contient $S(f)$. Par conséquent, l'ensemble d'articulation V_f sépare B de toutes les faces dont les capteurs candidats sont un sous-ensemble de X_f. Ainsi, les capteurs de V_f devraient avoir suffisamment d'énergie pour :

- recevoir et transmettre toutes les données collectées dans les faces ayant des capteurs candidats dans X_f et aucun capteur candidat dans V_f,
- recevoir ou surveiller, puis transmettre les données des faces avec des capteurs candidats dans X_f et avec certains capteurs candidats dans V_f mais pas tous,
- surveiller et transmettre les données des faces dont tous les capteurs candidats sont

inclus dans X_f et V_f.

On obtient :

$$\sum_{j\in J}\left(2\mu^j_{V_f}\rho+\sum_{\substack{k\in K_j\\ S_j(k)\subseteq V_f}}\Delta^j_k\right)p^S+\sum_{j\in J}\left(2\gamma^j_{V_f}\rho+\sum_{\substack{k\in K_j\\ S_j(k)\not\subseteq V_f\\ S_j(k)\cap V_f\neq\emptyset}}\Delta^j_k\right)\min(p^S,p^R)+$$

$$\sum_{j\in J}\left(2\varepsilon^j_{X_f\setminus V_f}\rho+\sum_{\substack{k\in K_j\\ S_j(k)\subseteq X_f\setminus V_f}}\Delta^j_k\right)p^R+\sum_{j\in J}\left(2\nu^j_{X_f}\rho+\sum_{\substack{k\in K_j\\ S_j(k)\subseteq X_f}}\Delta^j_k\right)p^T\leq\sum_{i\in V_f}E_i$$

où pour tout $j\in J$, les constantes suivantes $\mu^j_{V_f}$, $\gamma^j_{V_f}$ et $\varepsilon^j_{X_f\setminus V_f}$ sont définies par :

- $\mu^j_{V_f}$ est égal à 1 si et seulement si la cible j passe un temps non nul dans au moins une face dont les capteurs candidats sont un sous-ensemble de V_f, sinon $\mu^j_{V_f}=0$,
- $\gamma^j_{V_f}$ est égal à 1 si et seulement si la cible j passe un temps non nul dans au moins une face dont les capteurs candidats ont une intersection non vide avec V_f, mais ne constituent pas un sous-ensemble de V_f, sinon $\gamma^j_{V_f}=0$,
- $\varepsilon^j_{X_f\setminus V_f}$ est égal à 1 si et seulement si la cible j passe un temps non nul dans au moins une face dont les capteurs candidats forment un sous-ensemble de $X_f\setminus V_f$, sinon $\varepsilon^j_{X_f\setminus V_f}=0$.

Par conséquent, UB_3 peut être définie comme :

$$\rho\leq \mathrm{UB}_3=\min_{f\in F^\star}\left(\frac{\sum_{i\in V_f}E_i-\sum_{j\in J}\left(\sum_{\substack{k\in K_j\\ S_j(k)\subseteq V_f}}\Delta^j_k p^S+\sum_{\substack{k\in K_j\\ S_j(k)\not\subseteq V_f\\ S_j(k)\cap V_f\neq\emptyset}}\Delta^j_k\min(p^S,p^R)+\sum_{\substack{k\in K_j\\ S_j(k)\subseteq X_f\setminus V_f}}\Delta^j_k p^R+\sum_{\substack{k\in K_j\\ S_j(k)\subseteq X_f}}\Delta^j_k p^T\right)}{2\sum_{j\in J}\left(\mu^j_{V_f}p^S+\gamma^j_{V_f}\min(p^S,p^R)+\varepsilon^j_{X_f\setminus V_f}p^R+\nu^j_{X_f}p^T\right)}\right)$$

1.6 Méthode de résolution

Le problème ayant trois objectifs traités dans un ordre lexicographique, nous le séparons en trois problèmes successifs : P1 pour maximiser le rayon de stabilité, P2 pour maximiser le temps garanti dans les zones prioritaires, et P3 pour minimiser la consommation d'énergie. Ils sont résolus séquentiellement, de P1 à P3, où l'objectif d'un problème devient une contrainte dans le suivant. Par conséquent, la méthode de résolution comporte trois étapes successives.

Tout d'abord, un ordonnancement avec un rayon de stabilité maximum est recherché (P1 est présenté en détail à la section 1.6.1). Puisque P1 ne peut pas être résolu en calculant simplement la solution d'un seul programme linéaire car l'ensemble des faces à couvrir dépend du

rayon de stabilité, nous résolvons ce problème en utilisant d'abord une méthode de dichotomie. Chaque étape crée et résout un nouveau problème de décision pour une valeur fixe du rayon de stabilité Δ avec un programme linéaire appelé LP_Δ. Une fois que la valeur maximale de Δ, appelée Δ_{opt}, est trouvée dans un ensemble discret de valeurs potentielles avec une méthode de dichotomie, un programme linéaire final, appelé LP_ρ, est résolu. Il trouve la valeur optimale du rayon de stabilité $\rho = \Delta_{opt} + \delta$, avec δ la valeur de la fonction objectif de la solution optimale de LP_ρ.

Le problème P2 est ensuite abordé. Pour chaque rang ℓ, du plus élevé au plus bas, nous calculons une solution maximisant le temps pendant lequel on peut garantir la surveillance de toute cible dans une zone prioritaire de rang ℓ. Pour atteindre cet objectif, on résout un programme linéaire noté LP_ℓ relatif à la surveillance future de toutes les faces de rang ℓ.

La troisième étape consiste à résoudre P3 : la consommation d'énergie pour surveiller toutes les cibles est minimisée. Cela se fait en résolvant un programme linéaire noté LP_E.

L'algorithme 1 résume l'approche globale.

Algorithme 1: Résolution du problème P_c^n

```
// P1 - Maximisation de la robustesse
while (Δ ← prochaineValeurDichotomie()) ≠ null do
  Construire(LP_Δ)
  Δ = Résoudre(LP_Δ)
  if EstRéalisable(LP_Δ) then
    Δ_opt ← Δ
  end if
end while
Construire(LP_ρ, Δ_opt )
ρ = Résoudre(LP_ρ)
// P2 - Maximisation des garanties dans les zones prioritaires
for all ℓ ∈ C do
  Construire(LP_ℓ,ℓ)
  Résoudre(LP_ℓ)
end for
// P3 - Minimisation de la consommation d'énergie
Construire(LP_E)
return Résoudre(LP_E)
```

1.6.1 P1 : Maximisation du rayon de stabilité

La maximisation du rayon de stabilité est obtenue en étendant la méthode de dichotomie présentée dans [21]. Dans cette section, nous donnons un aperçu de cette méthode tout en fournissant plus de détails sur son adaptation au problème de ce chapitre. L'observation

principale est que l'augmentation du rayon de stabilité conduit à retarder les ticks entrants et à avancer les ticks sortants. Chaque fois que deux ticks de fenêtres de temps différentes échangent leur ordre d'apparition, une fenêtre de temps disparaît et une nouvelle est créée, avec un ensemble réduit de capteurs candidats. En conséquence, les contraintes du problème doivent être mises à jour. Il convient de noter qu'il existe un ensemble discret de valeurs pour le rayon de stabilité qui provoque de telles mises à jour, ces valeurs étant les distances entre les ticks entrants et sortants appartenant à des fenêtres de temps différentes. Nous utilisons la borne supérieure du rayon de stabilité, présentée à la section 1.5, pour réduire cet ensemble de valeurs en supprimant toutes les valeurs strictement plus grandes que la borne supérieure.

La première partie de la résolution de P1 consiste à trouver la valeur maximale du rayon de stabilité dans cet ensemble discret pour lequel le problème a une solution. Cela se fait avec une méthode de dichotomie qui vérifie l'existence d'un ordonnancement réalisable pour une valeur donnée dans cet ensemble, jusqu'à ce que la plus grande soit trouvée. La première valeur testée est toujours la borne supérieure du rayon de stabilité. En effet, si un ordonnancement réalisable avec un tel rayon de stabilité est trouvé, alors il est optimal pour P1. Si cette valeur n'est pas réalisable, un rayon de stabilité de 0 est testé. Si aucune solution n'est trouvée dans ce cas, alors il n'y a pas d'ordonnancement réalisable pour P1 et l'algorithme s'arrête. Cela se produit généralement lorsqu'une cible sort de la portée de tous les capteurs ou lorsque le réseau a si peu d'énergie que les données collectées ne peuvent pas être envoyées à la station de base. Si une solution est trouvée avec un rayon de stabilité non négatif, alors la méthode de dichotomie est utilisée pour trouver le plus grand rayon de stabilité dans l'ensemble discret. Cette méthode diffère de celle introduite dans [21] par la recherche d'un ordonnancement réalisable. En effet, en raison des exigences de transfert de données, nous devons résoudre un programme linéaire LP_Δ au lieu d'un problème de transport. Ce programme linéaire n'a pas de fonction objectif. Il est donné dans le modèle 1.1, et résolu avec un solveur.

Les variables de décision sont x_{jik}, le temps de surveillance de la cible j pendant sa fenêtre de temps k par le capteur i et $f_{ii'}$ la quantité de données transférées du capteur i au capteur i'. Nous introduisons H_j^i comme l'ensemble de toutes les fenêtres de temps de la cible j pour laquelle i est un capteur candidat, c'est-à-dire, l'ensemble de tous les $k \in K_j$ tels que i appartient à $S_j(k)$. Le programme linéaire LP_Δ (sans fonction objectif) est le modèle 1.1.

Les contraintes (1) représentent la limitation de la batterie pour chaque capteur, (2) modélisent le transfert de toutes les données vers la station de base (c'est une équation de conservation du flot). Enfin (3) assurent la satisfaction des contraintes de couverture, ce qui signifie que chaque cible est couverte en permanence par un capteur à tout moment.

À la fin de la méthode de dichotomie, on obtient la valeur maximale Δ_{opt} pour laquelle le problème obtenu est réalisable. Il ne reste ensuite plus qu'à maximiser le rayon de stabilité

$$p^S \sum_{j\in J} \sum_{k\in H_j^i} x_{jik} + p^R \sum_{i'\in N(i)} f_{i'i} + p^T \sum_{i'\in N(i)} f_{ii'} \leq E_i \qquad \forall i \in I \quad (1)$$

$$\sum_{j\in J} \sum_{k\in H_j^i} x_{jik} + \sum_{i'\in N(i)} f_{i'i} - \sum_{i'\in N(i)} f_{ii'} = 0 \qquad \forall i \in I \quad (2)$$

$$\sum_{i\in S_j(k)} x_{jik} = \Delta_k^j \qquad \forall j \in J, k \in K_j \quad (3)$$

$$x_{jik} \geq 0 \qquad \forall j \in J, k \in K_j, i \in S_j(k) \quad (4)$$

$$f_{ii'} \geq 0 \qquad \forall i \in I, i' \in N(i) \quad (5)$$

Modèle 1.1 – LP_Δ, le programme linéaire résolu à chaque itération de la méthode dichotomique

avec les fenêtres de temps et les ensembles de capteurs candidats de Δ_{opt}. Ainsi, tout en considérant ces fenêtres et capteurs candidats, nous maximisons l'augmentation du rayon de stabilité δ, en résolvant un programme linéaire similaire à celui utilisé pour déterminer Δ_{opt} dans LP_Δ. Le nouveau programme linéaire s'appelle LP_ρ, présenté dans le modèle 1.2. Il est identique à celui utilisé dans la méthode dichotomique, avec en plus une fonction objectif (6), une contrainte (7) et des contraintes mises à jour (3').

δ_{opt} = Maximiser δ (6)

$$\sum_{i\in S_j(k)} x_{jik} = \Delta_k^j + (\sigma_{k+1}^j - \sigma_k^j)\delta \qquad \forall j \in J, k \in K_j \quad (3')$$

$$\delta \geq 0 \quad (7)$$

Sous les contraintes (1), (2), (4), (5)

Modèle 1.2 – LP_ρ pour maximiser le rayon de stabilité après la dichotomie

Comme δ est l'augmentation du rayon de stabilité par rapport à Δ_{opt}, il a un impact sur la durée des fenêtres de temps dans (3'). σ_k^j est la valeur du k-ième tick de la cible j (-1 si sortant, 1 si entrant). La valeur numérique optimale du rayon de stabilité est alors $\rho = \Delta_{opt} + \delta$.

La figure 1.4 illustre la méthode pour résoudre P1.

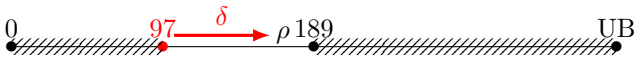

FIGURE 1.4 – Illustration de la méthode de résolution de P1

Dans cet exemple, avec différentes valeurs de ρ, les fenêtres de temps ne sont plus les mêmes ainsi que leurs ensembles de capteurs candidats. Le rayon de stabilité ρ peut être dans l'intervalle $[0, 97[$, ou dans $[97, 189[$, ou dans $[189, \mathrm{UB}[$, ou être égal à la borne supérieure. Au cours de la méthode dichotomique, nous trouvons $\Delta_{opt} = 97$, ce qui signifie qu'il existe un ordonnancement réalisable avec un rayon de stabilité de 97. Par conséquent, tous les ordonnancements avec un rayon de stabilité strictement inférieur à 97 ne sont plus pris en

compte. Ce résultat implique également qu'il n'y a pas de solution avec une valeur supérieure ou égale à 189. La deuxième étape de la résolution de P1 consiste alors à maximiser δ avec $0 \leq \delta < (189 - 97)$. La résolution de LP_ρ donne la valeur optimale de ρ, qui est égale à $97 + \delta$ dans cette illustration.

1.6.2 P2 : Maximiser les garanties de couverture dans les zones prioritaires

Maintenant que la valeur optimale de ρ a été identifiée, nous recherchons un ordonnancement qui maximise la garantie de couverture dans les zones prioritaires, tout en conservant la valeur optimale de ρ. Rappelons qu'un rang correspond à un ensemble de faces, et qu'une face ne peut avoir qu'un seul rang.

Dans cette phase, pour chaque rang du plus haut au plus bas, nous créons un nouveau problème, où nous maximisons T_ℓ, le temps de surveillance garanti dans toutes les zones dont le rang est ℓ, après la mission en cours. Pour chaque rang, un programme linéaire noté LP_ℓ est résolu. Il est rappelé que lors de la résolution de LP_ℓ pour un rang ℓ donné, les valeurs optimales de la fonction objectif des problèmes précédents résolus sont fixées (il s'agit du rayon de stabilité et des $T_{\ell'}$ pour tout $\ell' > \ell$), et l'objectif de LP_ℓ est la maximisation de T_ℓ. Par conséquent, les programmes linéaires résolus sont construits de manière incrémentale, en ajoutant et en modifiant les contraintes et les variables des programmes linéaires précédents résolus ($LP_{\ell+1}$ si $\ell < r$, LP_ρ sinon).

A partir du programme linéaire précédent résolu, nous fixons d'abord la valeur optimale de sa fonction objectif dans une nouvelle contrainte pour conserver la robustesse et les garanties obtenues aux rangs supérieurs. Nous modifions ensuite la fonction objectif et maximisons T_ℓ, la couverture garantie dans les zones de rang ℓ. Pour maximiser cet objectif, nous ajoutons des cibles virtuelles à surveiller dans toutes les faces incluses dans $\mathcal{F}(\ell)$. Ces cibles passent toutes le même temps, T_ℓ, dans leurs faces respectives. Ceci permet de modéliser l'exigence de couverture induite dans les faces de rang ℓ par la maximisation de T_ℓ, ce qui est équivalent à un temps de surveillance garanti. De plus, un routage virtuel des données (pris en compte par un nouvel ensemble de contraintes de flot) est ajouté pour router les données potentielles enregistrées à partir des cibles virtuelles jusqu'à la station de base. Ainsi, lorsque la surveillance de ces faces sera nécessaire, l'ensemble des capteurs candidats aura suffisamment d'énergie résiduelle après la mission courante pour suivre les cibles dans ces faces pendant T_ℓ unités de temps, et il y aura un chemin de capteurs avec suffisamment d'énergie restante pour transmettre les données à la station de base.

Les nouvelles variables de décision sont les suivantes :

— $x_{if\ell}$ le temps pendant lequel le capteur $i \in S(f)$ surveille une cible fictive dans une face f ayant le rang ℓ.

— $f^1_{ii'}$ la quantité de données générées par une cible fictive, transmises de i à i'.

Ces nouvelles variables permettent de considérer d'établir des garanties de couverture au-delà de la mission courante.

Le programme linéaire associé au rang ℓ est présenté dans le modèle 1.3.

Maximiser T_ℓ (6')

$$\sum_{j\in J}\sum_{k\in H_j^i} x_{jik}p^S + p^R \sum_{i'\in N(i)} f_{i'i} + p^T \sum_{i'\in N(i)} f_{ii'}$$
$$+p^S \sum_{f\in \bigcup_{\ell\le\ell'\le r}} \sum_{\mathcal{F}(\ell')|i\in S(f)} x_{if\ell'} + p^R \sum_{i'\in N(i)} f^1_{i'i} + p^T \sum_{i'\in N(i)} f^1_{ii'} \le E_i \quad \forall i\in I \quad (1')$$

$$\delta = \delta_{opt} \quad (8)$$

$$\sum_{i\in S(f)} x_{if\ell'} = T_{\ell'} \quad \forall \ell'\in C, \ell'\ge\ell, \ell'\le r, \forall f\in\mathcal{F}(\ell') \quad (9)$$

$$\sum_{f\in \bigcup_{\ell\le\ell'\le r}} \sum_{\mathcal{F}(\ell')|i\in S(f)} x_{if\ell'} + \sum_{i'\in N(i)} f^1_{i'i} - \sum_{i'\in N(i)} f^1_{ii'} = 0 \quad \forall i\in I \quad (10)$$

$$f^1_{ii'} \ge 0 \quad \forall i\in I, i'\in N(i) \quad (11)$$

$$T_{\ell'} = T^{opt}_{\ell'} \quad \forall \ell'\in C, \ell' > \ell \quad (12)$$

$$x_{if\ell'} \ge 0 \quad \forall \ell'\in C, \ell'\ge\ell, \ell' < r, \forall f\in\mathcal{F}(\ell'), i\in S(f) \quad (13)$$

sous les contraintes (2), (3'), (4), (5)

Modèle 1.3 – LP_ℓ, modèle résolu pour un rang ℓ

Les contraintes (1') sont les contraintes (1) modifiées pour prendre en compte les cibles fictives qui sont ajoutées et le flot virtuel qui en découle (transfert des données des cibles fictives à la station de base).
Les contraintes (2), (3'), (4), (5) sont les mêmes que dans P1.
La contrainte (8) fixe δ à sa valeur optimale, δ_{opt}, obtenue après la résolution de LP_ρ.
Les contraintes (9) assurent le suivi des cibles fictives pour un rang ℓ' pour une durée de $T_{\ell'}$ unités de temps.
Les contraintes (10) assurent l'équilibre des flux des données collectées générées par les cibles fictives, pour chaque capteur i.
Les contraintes (12) fixent les valeurs optimales des $T_{\ell'}$ pour tous les rangs précédents $\ell' > \ell$.

1.6.3 P3 : Minimisation de l'énergie consommée

Le problème P3 minimise f_3, l'énergie totale consommée pour accomplir la mission courante. Pour le résoudre, le programme linéaire LP_E est défini à partir du dernier programme linéaire construit lors de la résolution de P2, en fixant la valeur de sa fonction objectif à l'aide d'une nouvelle contrainte, et en changeant la fonction objectif comme indiqué ci-après.

Lors de la résolution de P3, l'ordonnancement et le routage peuvent être modifiés, mais la valeur du rayon de stabilité et les garanties de couverture obtenues en résolvant P1 et P2 sont maintenues à leurs valeurs optimales respectives.

L'énergie consommée par un capteur est représentée par le premier membre des contraintes (1), cette quantité peut s'écrire :

$$\sum_{i\in I}\left(\sum_{j\in J}\sum_{k\in H_j^i} x_{jik}p^S + p^R\sum_{i'\in N(i)} f_{i'i} + p^T\sum_{i'\in N(i)} f_{ii'}\right.$$

$$\left. + p^S \sum_{f\in \bigcup\limits_{\ell\le\ell'\le r}\mathcal{F}(\ell')|i\in S(f)} x_{if\ell} + p^R\sum_{i'\in N(i)} f^1_{i'i} + p^T\sum_{i'\in N(i)} f^1_{ii'}\right)$$

Le flot virtuel et les cibles fictives introduites lors de la résolution de P2 n'induisent pas de consommation réelle car ils ne sont pas utilisés pendant la mission en cours, mais sont épargnés pour de futures missions, ils sont donc soustraits de la quantité précédente, ce qui conduit à :

$$\sum_{i\in I}\left(\sum_{j\in J}\sum_{k\in H_j^i} x_{jik}p^S + p^R\sum_{i'\in N(i)} f_{i'i} + p^T\sum_{i'\in N(i)} f_{ii'}\right)$$

Enfin, comme la durée totale d'enregistrement des cibles est constante (le rayon de stabilité est fixe), on ne peut que minimiser le trafic des données transférées au cours de la mission courante, donc f_3, la fonction objectif de P3 est :

$$f_3 = \text{Minimiser} \sum_{i\in I, i'\in N(i)} f_{i'i}$$

1.6.4 Analyse de complexité

Dans cette section, on analyse la complexité de la discrétisation, et de la résolution des problèmes P1, P2 et P3.

La discrétisation peut être accomplie en utilisant un algorithme pseudo-polynomial. Une analyse précise de la complexité de la discrétisation peut être trouvée dans [21], qui montre que le nombre de faces est borné par une fonction polynomiale en le nombre de capteurs. Ainsi, les modifications apportées à ce travail (communication, zones prioritaires et cibles multiples), qui reposent toujours sur le nombre de capteurs et de faces, ne changent pas la complexité de la discrétisation.

Pour résoudre P1, on recourt à une dichotomie qui nécessite un nombre logarithmique d'itérations, avec à chaque itération, un programme linéaire à résoudre. La dichotomie se fait sur un ensemble de valeurs discrètes, qui dans le pire des cas est égal à toutes les intersections possibles entre les ticks. Et, si les cibles ont des trajectoires polygonales, le nombre de ticks ne peut pas dépasser $2qm$, avec q le nombre de segments dans les trajectoires et m le nombre

de capteurs. Pour résoudre P2, nous résolvons autant de programmes linéaires que le nombre de rangs, et la résolution de P3 nécessite la résolution d'un seul programme linéaire.

Le problème peut être résolu en temps pseudo-polynomial en raison de la discrétisation.

1.7 Expérimentations numériques

1.7.1 Description du protocole expérimental

Dans cette dernière partie, nous présentons nos expériences, leurs résultats et leur analyse. Nous étudions le comportement de la méthode de résolution proposée et l'impact de différents paramètres comme le nombre de capteurs, de cibles et de rangs. De plus, nous évaluons l'efficacité de la borne supérieure introduite dans ce travail par rapport aux deux premières, qui ont été étendues à partir de [21]. À cette fin, nous concevons quatre expérimentations sur différents ensembles d'instances, chacune d'entre elles étudie l'impact sur le problème des éléments suivants :

— Impact des puissances de détection et de communication. Nous les faisons varier afin de comparer UB_3 aux deux autres bornes supérieures. Dans cette expérience, nous étudions l'efficacité de cette nouvelle borne supérieure sur le rayon de stabilité.
— Impact de la densité des capteurs. Dans l'expérience, seul le nombre de capteurs varie.
— Impact du nombre de cibles. Plus de cibles à surveiller induit plus de données, envoyées ou reçues.
— Impact du nombre de rangs et de zones.
— Impact de la communication.

Bien que chaque expérimentation soit conduite sur un ensemble d'instances spécifique, un même générateur d'instances, présenté dans la section 1.7.2 est utilisé.

La méthode de résolution est codée en C++ et tous les calculs ont été exécutés sur un ordinateur avec Ubuntu 16.04 et un processeur Intel Core i7-6700HQ cadencé à 2.60 GHz $\times$ 8 cœurs et 16 Go de RAM. Nous utilisons la version 12.7 d'IBM CPLEX pour résoudre les programmes linéaires. Les temps CPU rapportés sont exprimés en secondes.

1.7.2 Générateur d'instances

Toutes les instances utilisées sont produites par un générateur dont les paramètres d'entrée sont :

— Le nombre de capteurs m
— La surface de la zone rectangulaire $L_1 \times L_2$ où la mission se déroule
— Le nombre de cibles n à surveiller

— Le niveau minimum E_{min} et maximum E_{max} d'énergie initialement disponible dans les batteries des capteurs
— Les puissances associées aux différents rôles des capteurs (p^S, p^T et p^R)
— La portée de communication R_c et la portée de détection R_s
— Le nombre de zones prioritaires q et le nombre de rangs r
— Le rayon maximum des zones prioritaires R_a (voir description plus bas)

Tout d'abord, les capteurs sont déployés avec une répartition uniforme dans la zone. Chaque capteur a un niveau uniforme de batterie tiré aléatoirement entre les deux valeurs données en paramètres. Deuxièmement, les trajets des cibles sont également dessinés uniformément dans la zone. Leurs chemins sont de simples itinéraires composés de trois segments de droites contigus. Troisièmement, les zones prioritaires sont déployées aléatoirement selon une distribution uniforme et leur rang est également choisi de la même manière, avec au moins une zone prioritaire par rang. Chaque zone prioritaire est un disque dont le rayon est sélectionné de façon aléatoire entre 50% et 100% de la valeur maximale donnée comme paramètre. Enfin, la position de la station de base est tirée aléatoirement dans la zone.

Les paramètres par défaut des instances sont présentés dans le tableau 1.4.

Paramètre	Valeur	Paramètre	Valeur
Nombre de capteurs m	400	Énergie dans les batteries $[E_{min}, E_{max}]$	$[350, 400]$
Nombre de cibles n	2	p^S	2.8
Dimension de la zone de recherche $L_1 \times L_2$	300×300	p^R	1
Portée de surveillance R_s	35	p^T	1
Portée de communication R_c	70	Nombre de zones prioritaires q	5
Rayon maximum des zones prioritaires R_a	50	Nombre de rangs r	2

Tableau 1.4 – Valeurs par défaut du générateur d'instance

1.7.3 Résultats et analyses

Nous présentons dans cette section les résultats et l'analyse des différentes expérimentations.

Impacts de la consommation due à la communication et à la surveillance

Dans cette première expérimentation, nous étudions la qualité de la nouvelle borne supérieure. L'objectif est de la comparer aux bornes supérieures précédentes et de juger de son utilité pour la résolution du problème. En effet, les deux bornes précédentes sont toujours valables dans notre problème de robustesse et ont été étendues pour prendre en compte plusieurs cibles et les coûts de communication. Cependant, elles ne considèrent que la consommation induite par la couverture des cibles à l'intérieur d'une face (c'est-à-dire, la consommation

résultant de la détection des cibles à l'intérieur de la face et de la transmission des données correspondantes). Elles ne prennent pas en compte la consommation induite par la transmission des données collectées reposant sur les capteurs servant de relais jusqu'à la station de base. Par conséquent, l'ajout de la nouvelle borne supérieure UB_3 basée sur un tel principe semble pouvoir compléter les deux bornes existantes et permettre d'accélérer la résolution. En effet, la valeur de la borne supérieure est importante car elle peut réduire drastiquement le nombre d'itérations dans la méthode dichotomique en P_1, donc le nombre de programmes linéaires à résoudre. De toute évidence, UB_3 devrait être d'autant plus efficace que la consommation due à la communication est importante par rapport à la consommation imputable aux activités de détection. Dans cette expérimentation, nous étudions l'efficacité de la nouvelle borne par rapport aux deux bornes précédentes étendues lorsque nous travaillons sur un réseau de capteurs sans fil avec plusieurs rapports significatifs entre la puissance de surveillance p^S et les puissances de communication p^R et p^T.

Nous testons trois ensembles de puissances différents. Dans chacun d'eux, $p^R = p^T$ et $p^S = 3$. Tout d'abord, nous fixons $p^R = p^T = 0.5$. Dans le deuxième ensemble, nous avons $p^R = p^T = p^S = 3$. Et dans le troisième ensemble, $p^R = p^T = 5$. Nous générons un ensemble de cinquante instances, et nous utilisons les trois ensembles de puissances différents sur chacune d'elles. Tous les autres paramètres sont fixés à leur valeur par défaut, indiquées au tableau 1.4, à l'exception du nombre de capteurs et de cibles fixés à 300 et 4 respectivement.

Nous rapportons le nombre de fois où UB_3 est meilleure (inférieure) que les deux autres bornes dans le tableau 1.5.

p^S	$p^R = p^T$	#{ min(UB_1,UB_2) $> \mathrm{UB}_3$ }
3	0.5	2 / 50
3	3	20 / 50
3	5	26 / 50

Tableau 1.5 – Domination de UB_3 sur UB_1 et UB_2

Le tableau 1.5 montre que lorsqu'il y a une faible consommation d'énergie induite par le transfert des données, UB_3 est dominée. En effet, cette borne n'atteint la meilleure valeur que dans 2 cas sur 50. Clairement, UB_1 et UB_2 sont moins impactées par les faibles coûts de communication, contrairement à UB_3 qui est principalement basée sur ces coûts. Cependant, pour une consommation moyenne, avec $p^R = p^T = p^S = 3$, UB_3 est presque aussi efficace que les deux autres bornes supérieures combinées. Dans un tel cas, dans 20 cas sur 50, UB_3 est strictement la meilleure borne supérieure sur le rayon de stabilité, et est clairement utile dans la méthode de dichotomie. Enfin, nous observons que lorsque la consommation d'énergie due à la communication est beaucoup plus importante que la consommation d'énergie liée à la surveillance, qui est la situation la plus réaliste [54], notre nouvelle borne supérieure do-

mine les deux autres dans 26 cas sur 50, et est donc nettement plus efficace. Naturellement, cette domination devient plus forte lorsque la consommation d'énergie due à la communication augmente.

Cependant, les deux autres bornes doivent toujours être prises en compte car elles sont utiles dans près de la moitié des cas (notons que pour les coûts de communication les plus élevés, UB_1 était la meilleure borne dans 12 cas, de même pour UB_2).

Pour conclure, la nouvelle borne introduite, UB_3, a de bonnes performances par rapport aux deux autres et elle apporte une contribution significative à l'approche, en réduisant le nombre de programmes linéaires à résoudre dans la méthode de dichotomie. Cependant, elle est moins utile lorsque l'on considère de faibles coûts de communication.

Impact de la densité des capteurs

Dans cette deuxième expérience, nous étudions l'impact de la densité des capteurs sur la méthode et le rayon de stabilité obtenu. Pour cela, nous générons un ensemble de cinquante instances réalisables en utilisant les paramètres par défaut (tableau 1.4), à l'exception du nombre de capteurs qui est fixé à 200. Ensuite, nous ajoutons quelques capteurs à chaque instance afin d'étudier l'impact de la densité des capteurs dans le réseau. Chaque capteur ajouté est généré de la même manière que les capteurs initiaux, ce qui signifie qu'ils ont des positions aléatoires dans la zone $L_1 \times L_2$ et un niveau aléatoire de batterie initial dans $[E_{min}, E_{max}]$. La méthode est exécutée sur les instances lorsque le nombre de capteurs m est dans $\{200, 250, 300, 350, 400, 700\}$.

Dans le tableau 1.6, nous rapportons le nombre moyen de fenêtres de temps et le nombre de faces incluses dans une zone prioritaire. Le tableau 1.7 rapporte l'effort de calcul requis par chaque étape de la méthode de résolution avec le nombre moyen de programmes linéaires résolus pour P1. La figure 1.5 illustre, avec un graphique, ces temps de calcul avec en abscisse les différents nombres de capteurs. Enfin, dans le tableau 1.8, nous rapportons les valeurs moyennes des objectifs. Les temps de calcul sont exprimés en secondes.

#Capteurs	#Fenêtres	#Faces dans les zones prioritaires
200	103.48	85.94
250	128.94	133.58
300	154.58	188.26
350	179.94	254.40
400	204.94	324.80
700	357.38	973.34

Tableau 1.6 – Nombre moyen de fenêtres de temps et de faces dans les zones prioritaires avec différents nombres de capteurs

Le tableau 1.6 montre que lorsque la densité de capteurs augmente, le nombre de fenêtres de temps augmente également. Par conséquent, chaque étape de la méthode de résolution prend plus de temps, avec plus de données à traiter, plus de contraintes et plus de variables dans les modèles. S'ajoute à cela également une augmentation rapide du nombre de faces à l'intérieur des zones prioritaires. Ainsi, les modèles pour P2 et P3 deviennent encore plus complexes avec plus de contraintes et plus de variables.

#Capteurs	CPU (Discrétisation)	CPU (UB)	CPU (P1)	#LP résolus dans P1	CPU (P2)	CPU (P3)	Temps CPU global
200	0.02	0.04	0.28	3.44	1.22	0.49	2.05
250	0.02	0.06	0.16	1.22	2.38	1.14	3.75
300	0.04	0.13	0.41	2.30	4.01	1.86	6.45
350	0.06	0.20	0.35	1.22	6.39	2.62	9.62
400	0.10	0.33	0.53	1.48	8.94	3.63	13.54
700	0.86	2.47	1.03	1.00	59.65	19.52	83.53

Tableau 1.7 – Effort de calcul moyen avec différents nombres de capteurs

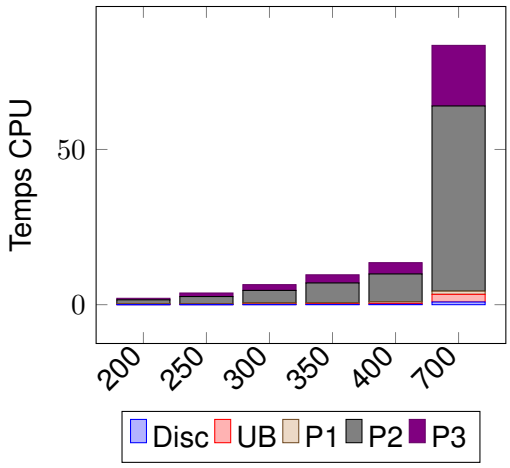

FIGURE 1.5 – Représentation de l'effort de calcul moyen avec différents nombres de capteurs

Chaque ligne du tableau 1.7 présente les résultats moyens sur cinquante instances. La première colonne est le nombre de capteurs. La seconde est le temps CPU (en secondes) passé dans l'étape de discrétisation, CPU (UB) est le temps moyen (en secondes) pour calculer la borne supérieure (section 1.5). Les colonnes 4, 6 et 7 représentent les temps CPU pour la résolution de P1, P2 et P3 respectivement. La colonne 5 représente le nombre moyen de programmes linéaires lors de la résolution de P1. La dernière colonne est le temps CPU moyen total requis pour résoudre une instance.

Comme prévu, la densité du réseau impacte beaucoup le temps de résolution. En effet, les résultats montrent que, lorsque le nombre de capteurs par unité de surface augmente, l'effort de calcul nécessaire à chaque étape et donc le temps CPU global augmentent rapidement. On peut l'expliquer avec le tableau 1.6, qui montre que le nombre de fenêtres de temps augmente. Par conséquent, les temps CPU requis par la discrétisation et le calcul de la borne supérieure

augmentent rapidement, car ils dépendent principalement du nombre de fenêtres de temps. Le temps de résolution de P1 dépend également du nombre de programmes linéaires à résoudre. C'est pourquoi le temps CPU pour P1 n'augmente pas toujours lorsque la densité augmente, car il y a souvent moins de programmes linéaires résolus dans P1 (parce que la borne supérieure est plus souvent une solution réalisable par exemple) bien qu'ils soient plus complexes à résoudre. Les processus de résolution de P2 et P3 sont fortement impactés par le nombre de faces dans les zones prioritaires, donc leurs temps de calcul augmentent également de manière significative lorsque la densité des capteurs est plus élevée.

#Capteurs	Rayon de stabilité	T_2	T_1	f_3
200	79.26	40.24	5.84	10816.16
250	102.98	43.74	4.87	10667.43
300	113.56	45.58	3.33	10509.07
350	119.75	47.58	1.56	10342.56
400	124.12	45.30	1.35	10312.58
700	143.20	32.01	1.23	10179.60

Tableau 1.8 – Évolution des objectifs avec le nombre de capteurs

Le tableau 1.8 montre que le rayon de stabilité augmente avec le nombre de capteurs par unité de surface. Le deuxième objectif, T_2, qui est le temps de surveillance garanti dans les zones les plus prioritaires après la mission courante, semble augmenter dans un premier temps mais avec plus de 350 il diminue. L'objectif suivant, T_1, relatif aux zones de moindre priorité, est impacté négativement par l'augmentation du nombre de capteurs, la garantie de couverture dans les zones les moins prioritaires diminuant. Enfin, le dernier objectif (la minimisation de la consommation liée à la communication) s'améliore faiblement aussi quand la densité de capteurs augmente.

Comme prévu, l'ajout de capteurs à un réseau permet d'augmenter la robustesse offerte. Cela ajoute plus d'opportunités pour le réseau de surveiller les cibles et de transférer les données collectées. Ainsi, il est moins contraint par les limites énergétiques des batteries et par la contrainte de n'avoir qu'un seul capteur surveillant une cible à la fois. Cette observation explique également la diminution du dernier objectif. Avec plus de capteurs, le réseau a plus de possibilités pour des itinéraires plus courts et plus directs vers la station de base. Il induit moins de transferts de données entre les capteurs et donc moins de consommation d'énergie. Les objectifs sur les temps garantis dans les zones prioritaires sont affectés différemment. En effet, avec plus de capteurs, il y a plus d'énergie dans le réseau et il semble que cela donne plus de possibilités d'augmenter ces objectifs. Cependant, pour chaque rang ℓ dans C, T_ℓ est contraint par le temps garanti dans chaque face f dans $\mathcal{F}(\ell)$, donc T_ℓ est contraint par la face la moins bien couverte. L'ajout de capteurs dans le réseau n'apporte pas nécessairement plus de capteurs dans chaque face déjà dans $\mathcal{F}(\ell)$. Ainsi, il peut toujours s'agir de la même face (exac-

tement le même ensemble de capteurs candidats) qui restreint la valeur de T_ℓ après l'ajout de nouveaux capteurs au réseau. Cependant, il ajoute encore plus de faces à couvrir dans les zones prioritaires (Tableau 1.6). Par conséquent, le réseau a moins de temps de couverture garanti dans les zones prioritaires.

Impact du nombre de cibles

Dans cette troisième expérience, nous analysons l'impact du nombre de cibles. Comme dans la deuxième expérience, tous les paramètres sont fixes (voir tableau 1.4) sauf celui que nous étudions. Le nombre de cibles varie dans l'ensemble $\{1, \ldots, 7\}$. Un premier ensemble d'instances réalisables est généré et résolu avec 7 cibles. Ensuite, nous supprimons simplement les cibles une par une, dans ces instances pour observer l'impact de différents nombres de cibles sur les performances.

Nous rapportons dans nos résultats les temps d'exécution moyens pour toutes les instances. Les résultats sont résumés dans le tableau 1.9. Les temps CPU sont aussi illustrés par la figure 1.6 avec en abscisse les différents nombres de cibles. Les valeurs des objectifs sont reportées dans le tableau 1.10. Les temps CPU rapportés sont exprimés en secondes.

#Cibles	CPU (Discrétisation)	CPU (UB)	CPU (P1)	CPU (P2)	CPU (P3)	Temps CPU global
1	0.10	0.15	0.55	9.09	3.02	12.90
2	0.10	0.28	0.49	9.33	3.22	13.41
3	0.10	0.39	0.50	9.30	3.30	13.57
4	0.10	0.49	0.75	9.75	3.38	14.48
5	0.10	0.61	1.19	10.38	3.60	15.88
6	0.10	0.74	1.84	11.25	3.66	17.59
7	0.10	0.89	5.30	12.13	4.18	22.61

Tableau 1.9 – Effort de calcul moyen avec différents nombres de cibles

Dans cette expérimentation, nous observons que P1, P2 et P3 ont des temps CPU croissants qui se traduisent par une croissance globale de l'effort de calcul. Cependant, les résultats peuvent être analysés plus en profondeur. Tout d'abord, pour la discrétisation, seule la partie du calcul des itinéraires des cibles est impactée, et non la discrétisation de la communication ou des zones prioritaires. Par conséquent, le temps consommé dans cette phase augmente trop lentement pour être mesurable. Deuxièmement, les temps les plus fortement croissants concernent P1 et le calcul des bornes supérieures car ils dépendent totalement du nombre de cibles : plus de cibles implique plus de fenêtres de temps et un rayon de stabilité plus complexe à calculer. Enfin, le temps nécessaire pour P2 et P3 augmente lentement car seule la partie du modèle héritée de P1 est impactée.

Pour conclure sur l'effort de calcul, comme prévu, chaque partie de la méthode de résolution a besoin de plus de temps lorsque des cibles sont ajoutées.

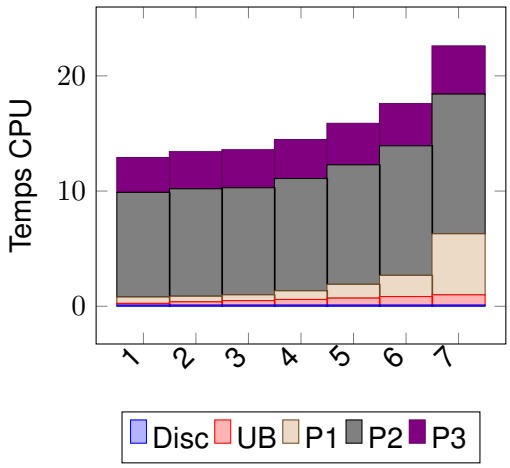

FIGURE 1.6 – Représentation de l'effort de calcul moyen avec différents nombres de cibles

#Cibles	Rayon de stabilité	T_2	T_1	f_3
1	207.60	55.84	5.93	5629.14
2	127.61	51.30	4.63	9568.11
3	105.98	48.30	3.25	13606.10
4	98.79	43.40	1.95	18139.19
5	93.77	35.90	1.05	22820.62
6	87.98	26.07	0.67	28297.36
7	72.55	20.31	0.36	33337.56

Tableau 1.10 – Évolution des objectifs avec le nombre de capteurs

Les valeurs des objectifs du tableau 1.10 montrent qu'avec plus de cibles, le rayon de stabilité diminue évidemment. En effet, avec plus de cibles, le réseau est plus limité, car il consomme plus pour surveiller les cibles. De plus, si nous augmentons le rayon de stabilité, la garantie doit s'appliquer à toutes les cibles. Par conséquent, nous consommons plus d'énergie lorsque nous augmentons le rayon de stabilité pour les problèmes avec un nombre croissant de cibles.

La même observation peut être faite avec les autres objectifs. Le temps garanti dans les zones prioritaires diminue et la consommation totale d'énergie augmente également. Cela montre aussi que le réseau a encore beaucoup d'énergie après avoir résolu P1 pour seulement quelques cibles. Dans de tels cas, le rayon de stabilité est limité par une région spécifique du réseau faiblement couverte, ou par la contrainte sur le nombre de capteurs activés en même temps. Par conséquent, la quantité totale d'énergie qui est consommée (minimisée en P3) est limitée, et davantage d'énergie est disponible pour garantir un temps plus long dans les zones prioritaires. Cependant, lorsqu'il y a de nombreuses cibles, le réseau consomme plus d'énergie pour les surveiller, donc le troisième objectif augmente et, de plus, il y a moins d'énergie pour les zones prioritaires. Par conséquent, avec plus de cibles, même avec un rayon de stabilité

décroissant, nous consommons plus d'énergie et pouvons garantir moins de temps pour les zones prioritaires.

Impact du nombre de zones prioritaires et du nombre de rangs

Dans cette expérimentation, nous étudions l'impact des zones prioritaires en générant de nombreuses instances avec des variations dans le nombre de zones générées ainsi que le nombre de rangs. De même que pour l'expérience sur les cibles, nous générons et résolvons d'abord un ensemble de cinquante instances qui seront ensuite faiblement modifiées pour simuler différents nombres de rangs et de zones prioritaires. Les instances initiales ont seulement deux zones de même rang. Ensuite, nous ajoutons des zones et des rangs sur ces mêmes instances. Par conséquent, nous obtenons différentes versions d'une même instance, avec plus ou moins de zones prioritaires. Pour chaque instance, nous avons des versions avec 1 à 7 rangs pour respectivement 2, 4 à 14 zones avec au moins une zone par rang. Les temps CPU moyens sont présentés dans le tableau 1.11 et les valeurs des objectifs pour P2 et P3 (le rayon de stabilité ne varie pas) dans le tableau 1.12. La figure 1.7 illustre les temps CPU avec en abscisse les différents nombres de rangs. Les temps CPU sont exprimés en secondes.

#Rangs	#Zones	CPU (Discrétisation)	CPU (UB)	CPU (P1)	CPU (P2)	CPU (P3)	Temps CPU global
1	2	0.06	0.33	0.55	4.59	3.38	8.90
2	4	0.08	0.36	0.57	10.77	4.17	15.96
3	6	0.12	0.33	0.56	14.82	4.09	19.92
4	8	0.16	0.33	0.56	19.12	4.12	24.29
5	10	0.20	0.32	0.55	23.94	3.95	28.96
6	12	0.25	0.33	0.57	29.39	4.04	34.57
7	14	0.29	0.32	0.57	34.51	4.11	39.80

Tableau 1.11 – Effort de calcul moyen avec différent nombres de rangs et zones de priorité

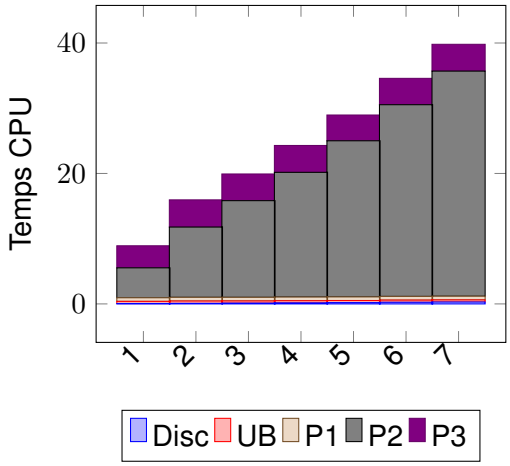

FIGURE 1.7 – Représentation de l'effort de calcul moyen avec différents nombres de rangs

Cette expérience montre que les nombres de rangs et de zones augmentent l'effort de calcul pour la discrétisation et pour le problème P2. De plus, comme prévu, le calcul de la borne supérieure sur le rayon de stabilité et le problème P1 ne sont pas impactés par le découpage des zones. On peut observer qu'avec l'augmentation du nombre de rangs, le temps CPU global augmente, d'un facteur cinq quand on passe d'un à sept rangs.

#Rangs	T_1	T_2	T_3	T_4	T_5	T_6	T_7	f_3
1	46.93							10058.05
2	7.58	50.56						10158.19
3	2.97	12.61	64.61					10383.87
4	1.44	3.94	24.39	68.00				10425.95
5	0.09	0.91	6.48	25.07	65.57			10197.06
6	0.00	0.00	1.31	6.04	22.87	66.62		10407.70
7	0.00	0.00	0.16	1.22	6.77	26.86	70.71	10480.30

Tableau 1.12 – Évolution des objectifs avec le nombre de rangs

Le tableau 1.12 montre que le temps garanti dans une zone dépend fortement de la priorité qui lui est assignée. En effet, à titre d'exemple, les zones de rang 1 ont beaucoup de temps garanti lorsqu'elles sont les seules zones prioritaires. Cependant, pour les mêmes zones, le temps diminue lorsque nous ajoutons des rangs plus élevés. En effet, avec beaucoup de rangs, le temps garanti est toujours très faible pour les rangs les moins élevés. Par conséquent, nous concluons que le temps garanti dans une zone dépend fortement de son rang. Pour les zones les plus prioritaires, le temps garanti est énorme mais diminuera fortement selon la priorité. Pour une faible priorité, il n'y a souvent pas de temps garanti. Nous conseillons d'utiliser au plus trois ou quatre rangs.

Deuxièmement, nous remarquons que l'énergie totale consommée varie très peu. En effet, il semble que le temps global garanti dans toutes zones prioritaires (la somme des T_ℓ, pour tous les ℓ) n'augmente pas considérablement car il est probablement limité par la région la moins bien couverte par les capteurs quel que soit le nombre de rangs. Par conséquent, l'état du réseau et l'énergie résiduelle après la résolution de P2 ne diffèrent pas beaucoup avec plus ou moins de rangs. Le processus de minimisation de la consommation d'énergie obtient donc des valeurs très proches.

Impact de la prise en compte des communications sur les objectifs

Dans cette expérimentation, nous étudions l'impact de la communication. Il s'agit de montrer comment le transfert de données obligatoire vers la station de base impacte la qualité de la solution et l'effort de calcul. La prise en compte de la communication rend le problème plus complexe à résoudre et limite la valeur des objectifs. Afin d'étudier de telles hypothèses, nous comparons les objectifs de la solution obtenue par la méthode avec et sans considérer la

communication sur un ensemble d'instances. Plusieurs coûts de communication différents sont testés pour représenter différents impacts sur le réseau. Cinquante instances sont générées avec les paramètres par défaut présentés précédemment (section 1.7.2). Notre méthode est exécutée plusieurs fois sur chacune de ces instances, chaque fois avec un profil de consommation d'énergie différent pour la communication. Les valeurs utilisées pour p^R et p^T sont $\{1, 2, 3\}$, avec $p^T = p^R$. La valeur de p^S est toujours égale à la valeur par défaut. Ensuite, pour chaque instance, nous exécutons notre méthode sans considérer les coûts de communication (et donc $p^T = p^R = 0$). Nous rapportons la valeur moyenne des objectifs dans le tableau 1.13, pour chaque valeur testée.

Ensuite, un deuxième ensemble d'instances réalisables est généré, avec les paramètres par défaut, à l'exception du nombre de cibles fixé à 1 et sans zone prioritaire. Par conséquent, sur ces cas, nous pouvons comparer notre méthode et la méthode développée dans les travaux précédents [21]. Il est rappelé que la communication est ignorée dans cet article. Plusieurs coûts de communication sont testés à nouveau pour notre méthode, avec $p^R = p^T$ et p^R en $\{0, 1, 2, 3\}$. Les temps CPU (toujours en seconde) nécessaires au calcul de UB et P1 sont reportés dans le tableau 1.14.

$p^R = p^T$	Rayon de stabilité	T_2	T_1	f_3
0	124.60	90.52	40.68	0
1	124.27	52.94	4.58	9559.63
2	122.38	27.79	0.84	19975.08
3	111.92	14.63	0.10	30292.13

Tableau 1.13 – Évolution des objectifs pour différentes consommations liées à la communication

Problème	$p^R = p^T$	CPU (UB)	CPU (P1)
Notre problème	0	0.17	0.13
	1	0.17	0.44
	2	0.17	1.05
	3	0.17	1.19
[21]	0	0.004	0.08

Tableau 1.14 – Effort de calcul moyen avec considération de différents coûts de communication

Comme prévu, toutes les valeurs des objectifs se dégradent lors de l'augmentation de la consommation due à la communication. Bien que, pour le rayon de stabilité, la différence soit très faible entre le cas où la communication n'induit aucune consommation ($p^R = p^T = 0$) et les cas où la communication se fait à faible consommation. Cependant, il y a une plus grande différence lorsque la consommation des communications est élevée ($p^R = p^T = 3$), avec une baisse d'environ 10 $\%$ du rayon de stabilité. Les temps de couverture laissés dans les zones prioritaires, pour les deux rangs, diminuent également rapidement avec des consommations

liées à la communication plus élevées. Lorsque la consommation de la communication augmente, les zones prioritaires de rang 1 perdent presque tout leur temps de couverture, et les zones de rang 2 une partie importante. Enfin, l'énergie globale consommée pour la communication (c'est-à-dire f_3) est évidemment en constante augmentation quand la consommation des communications s'accroît.

Ces résultats confirment que, comme on pouvait s'y attendre, la consommation due à la communication limite les objectifs. La valeur du rayon de stabilité est impactée, car plus d'énergie est nécessaire pour couvrir les cibles. Cependant, cela ne varie pas beaucoup avec de faibles consommations liées à la communication. En effet, le rayon de stabilité est également limité par la contrainte stipulant qu'un seul capteur au plus soit activé pour chaque cible à tout moment. Ainsi, dans ces cas, la sur-consommation due à la communication ne réduit pas nécessairement le rayon de stabilité. C'est pourquoi les valeurs atteintes par le rayon de stabilité sont similaires pour des consommations de communication nulles ou faibles. Cependant, le rayon de stabilité et les autres objectifs diminuent sensiblement la communication consomme davantage d'énergie. La consommation d'énergie augmente beaucoup, car il n'y a plus seulement les capteurs à portée d'une cible qui consomment de l'énergie. L'énergie réservée dans le réseau aux zones prioritaires est beaucoup moins abondante, limitant ainsi les valeurs atteignables des objectifs.

Pour conclure sur les valeurs des objectifs, si la prise en compte de la communication les impacte négativement, la baisse de robustesse offerte reste modérée, en particulier pour les faibles coûts de communication.

Il était également prévisible que les temps CPU dans le tableau 1.14 sont plus élevés pour le problème de ce chapitre avec prise en compte de la communication. Les temps de calcul pour la borne supérieure et pour P1 sont plus longs. Pour la borne supérieure, ils ne varient pas avec les différents profils de consommation liée à la communication. Cependant, la différence entre les temps nécessaires à la résolution de P1, avec et sans communication, devient de plus en plus importante lorsque la consommation imputable à la communication augmente. Cela montre également que résoudre le problème sans considérer la communication est plus rapide que résoudre le même problème tout en considérant la communication mais sans aucun coût, ce qui, encore une fois, n'est pas surprenant.

Ces résultats étaient attendus, car le problème résolu est beaucoup plus complexe, et même avec des coûts de communication nuls, la nécessité de prévoir l'acheminement des données vers la station de base demeure. Le programme linéaire résolu dans la méthode dichotomique de ce chapitre nécessite beaucoup plus d'efforts de calcul que dans [21], où un problème de transport doit être résolu. La borne supérieure peut également être moins serrée, augmentant ainsi le temps de résolution de P1. Enfin, la nouvelle borne UB_3, même si elle est efficace, augmente évidemment le temps de résolution de UB.

1.8 Conclusion du premier chapitre

Dans ce chapitre, nous avons étendu le problème d'origine, traité dans [21]. Leur objectif était de trouver un ordonnancement d'activation pour suivre une cible avec un réseau de capteurs sans fil. Nous l'avons développé pour rendre le problème plus général et réaliste en prenant en compte désormais des situations avec plusieurs cibles à suivre en même temps par le même ensemble de capteurs. De plus, nous avons ajouté la prise en compte des communications entre les capteurs, avec pour tâche de transférer toutes les données recueillies vers une station de base. La communication est une généralisation importante car son impact énergétique sur un réseau est significatif, et peut être prépondérante dans certaines applications. Ces deux extensions modifient le processus de résolution. En effet, la discrétisation est étendue pour prendre en compte la communication et les cibles multiples. De même, l'optimisation du rayon de stabilité (problème P1) est plus complexe, car la méthode dichotomique nécessite de résoudre un programme linéaire au lieu d'un problème de transport comme dans [21]. Nous avons adapté les bornes supérieures précédentes à de multiples cibles et communications et introduit une nouvelle borne, basée sur la consommation d'énergie due à la communication. La pertinence de cette nouvelle borne a également été vérifiée. En outre, nous avons ajouté deux objectifs à notre problème, optimisés après P1. Les nouveaux objectifs sont la maximisation du temps de surveillance garanti à l'intérieur des zones prioritaires après la mission courante (P2) et la maximisation de l'énergie globale restante (P3). Nous avons ajouté deux étapes à la méthode de résolution de ces objectifs, basée sur de nouveaux modèles formulés comme des programmes linéaires.

Nous avons conçu de nombreuses expérimentations pour tester et analyser la méthode de résolution. Les résultats montrent que l'augmentation du nombre de capteurs, cibles, rangs ou zones contribue à l'augmentation du temps de calcul global. Le nombre de capteurs, et plus particulièrement ici, la densité, est le paramètre le plus important en termes de temps CPU. Enfin, nous avons également montré que la nouvelle borne supérieure basée sur la communication, est efficace et s'avère utile dans le processus de résolution.

Les travaux effectuées dans ce chapitre peuvent être le point de départ de nombreuses extensions, dont certaines sont abordées dans cette thèse. Premièrement, que se passe-t-il si une cible a un retard ou une avance en dehors du rayon de stabilité ? Si un tel événement se produit, nous n'avons plus aucune garantie que la cible soit couverte, et on peut donc la considérer comme perdue, ce qui met en échec l'ordonnancement robuste. Pour éviter une telle situation, nous proposons de compléter l'ordonnancement robuste par une stratégie de réaction en ligne. Ce faisant, nous devons nous placer dans un contexte temps-réel puisque la mission est en cours au moment où l'on intervient pour modifier l'ordonnancement. Il est donc préférable d'éviter de recalculer entièrement un nouvel ordonnancement, puisque chaque se-

conde devient cruciale. Les problématiques soulevées par ces questions sont abordées dans le chapitre 2. Une autre extension naturelle des travaux de ce chapitre consiste à considérer la question de l'incertitude spatiale. Que se passe-t-il lorsque la déviation de la cible par rapport à la trajectoire estimée n'est plus uniquement temporelle mais aussi spatiale ? Nos réflexions préliminaires sont consignées dans le chapitre 3.7, à la sous-section des perspectives avancées. Nous y présentons la formulation du problème où l'incertitude est uniquement spatiale, et des pistes de résolution, de manière similaire au cas de l'incertitude temporelle. Quelques résultats préliminaires accompagnent ces réflexions. Nous présentons aussi dans la conclusion, une perspective moins développée sur le cas où les incertitudes sont à la fois spatiales et temporelles. Enfin, une dernière extension est envisagée, consistant à illustrer et à tester avec plus de profondeur notre méthode, notamment à l'aide d'un outil plus réaliste et interactif tel que le simulateur CupCarbon [61, 62].

CHAPITRE 2

RÉACTIONS DYNAMIQUES POUR MAINTENIR LE SUIVI DE CIBLES : DE LA ROBUSTESSE AUX GARANTIES DE PERFORMANCE EN LIGNE

2.1 Introduction

Ce chapitre prend la suite des travaux effectués dans le chapitre 1. L'idée est de compléter l'approche robuste du premier chapitre et notamment de répondre à la question : que faire si une cible a un retard ou une avance dont la valeur est supérieure à celle du rayon de stabilité ? Ne rien faire expose à perdre la cible, alors que le réseau de capteurs offre peut-être des solutions au problème, à condition qu'il soit possible de réagir en temps-réel. On propose ici une approche dynamique pour éviter de perdre la cible, ce qui compromettrait la mission du réseau. Ainsi, dans ce chapitre, nous reprenons les notations et principes de la discrétisation introduits au chapitre 1, puisque le problème reste fondamentalement le même (c'est-à-dire, surveiller continûment des cibles sujettes à une incertitude temporelle), mais la possibilité de réagir aux avances et retards est maintenant permise. L'ordonnancement fourni par la méthode de résolution du chapitre 1 est quant à lui aussi réutilisé, et sert de base pour l'approche dynamique proposée ici. Il ne sera modifié que partiellement. Cependant, malgré les similitudes, les approches algorithmiques proposées dans ce chapitre ne sont plus les mêmes. En effet, eu égard à sa dimension dynamique, la méthode proposée dans ce chapitre se veut compatible avec des contraintes dites "temps réel", elle donc doit s'exécuter très rapidement. Elle offre ainsi la possibilité de s'adapter aux véritables retards et avances des cibles. En remettant en question les activités des capteurs, elle permet même d'offrir, dans certains cas, de nouvelles garanties de performances propres au contexte dynamique.

L'utilisation dynamique de réseaux de capteurs a fait l'objet de nombreuses études dans la littérature [63] mais sur des problématiques différentes. Par exemple, [64] propose une solution basée sur un ensemble de capteurs et des prises de décision dynamiques pour permettre

l'exploitation de moteurs distants. Les auteurs relèvent les défis du monde réel et enquêtent sur la faisabilité de leur système par le biais d'expériences en laboratoire et d'essais sur le terrain. Un autre exemple d'utilisation dynamique est [65], où les auteurs comparent l'efficacité de différents protocoles de routage. Ils basent leur travail sur des ensembles de capteurs mobiles, donc avec une topologie du réseau sans fil qui évolue avec le temps. Les protocoles de routage sont donc dynamiques et offrent une robustesse contre les liaisons sans fil instables.

Dans ce chapitre, ainsi qu'il l'a été mentionné plus tôt, nous souhaitons trouver des solutions dynamiques aux retards et aux avances qui sortent du rayon de stabilité d'un ordonnancement pour le suivi de cibles par un réseaux de capteurs sans fil. Sans réaction, on sait que l'on s'expose à faire échouer la mission de surveillance continue. L'approche considérée dans ce chapitre ne consiste plus à calculer une solution robuste au préalable (hors ligne) qui sera ensuite implémentée dans les capteurs avant le début de la mission. Il s'agit maintenant d'une approche dynamique (en ligne) où la méthode est appelée plusieurs fois au cours de la mission, lorsque les cibles sont en mouvement. On considère que le réseau de capteurs dispose d'un ordonnancement robuste initial obtenu avec la méthode du chapitre 1. Les objectifs visés dans ce chapitre sont, dans un premier temps, de pouvoir détecter les avances et les retards qui sortent du rayon de stabilité. Dans un second temps, on souhaite pouvoir réagir à de tels événements pour assurer la continuité de la couverture des cibles. De plus, on souhaite profiter de ces interventions pour offrir des garanties sur les performances de la méthode en ligne pour compléter la robustesse de la solution initiale. Il s'agit plus précisément de garantir qu'une solution réalisable pourra être obtenue en ligne si certaines conditions spécifiques, qui seront détaillées dans la suite, sont satisfaites. Enfin, chaque fois que ces conditions ne sont plus remplies, les gestionnaires du réseau se voient proposer une alternative de dernier recours pour exploiter encore au mieux les capteurs pour continuer à étendre la couverture des cibles. C'est-à-dire que la méthode proposera éventuellement des solutions pour pallier les avances et retards, quitte à dégrader ses capacités de réaction futures.

La principale contribution de ce chapitre est la méthode en ligne qui adapte dynamiquement un ordonnancement initial pour couvrir en permanence les cibles. Cette méthode repose sur un algorithme hors ligne, qui permet de prendre des décisions rapides en ligne. Tout d'abord, la partie hors ligne (voir la section 2.3) construit un ensemble de solutions partielles pour fournir une couverture supplémentaire aux cibles. Ces solutions partielles sont appelées *chemins de secours*, elles sont constituées d'un ensemble de capteurs qui peuvent surveiller une cible et transférer les données vers la station de base lorsque l'ordonnancement initial ne le permet plus. En préambule à la méthode en ligne, nous introduisons le rayon de stabilité dynamique, afin d'étendre le rayon de stabilité [23]. Le rayon de stabilité dynamique est la valeur maximale du retard pour laquelle les chemins de secours garantissent une couverture continue de toutes les cibles. Par conséquent, tant que les cibles n'ont pas de retard ou d'avance supé-

rieur au rayon de stabilité dynamique, la méthode en ligne garantit de couvrir toutes les cibles. Deuxièmement, nous présentons la méthode en ligne qui met en œuvre cette stratégie. Cela commence par la détection dynamique (voir la section 2.4) des retards et avances qui peuvent compromettre la surveillance continue des cibles (c'est-à-dire, les retards et avances qui dépassent le rayon de stabilité de l'ordonnancement robuste initial). Cette détection déclenche la stratégie en ligne présentée dans la section 2.5, qui consiste à exploiter les chemins de secours et à modifier l'ordonnancement robuste. Son objectif est d'adapter la solution initiale calculée hors-ligne aux parcours réels des cibles, afin de s'assurer que toutes les cibles soient couvertes.

Ce chapitre est organisé comme suit : La section 2.2 énonce formellement le problème. La section 2.3 présente la partie hors-ligne de notre approche, qui correspond au calcul des chemins de secours qui seront disponibles pour la partie en ligne lorsque l'ordonnancement initial n'est plus réalisable. Ensuite, nous décrivons la partie en ligne à la section 2.4 avec la détection des retards et des avances, puis la section 2.5 présente la modification de l'ordonnancement, et l'utilisation dynamique des chemins de secours. Des expériences numériques sont rapportées dans la section 2.6 et les conclusions sont présentées à la section 2.7.

2.2 Définition du problème et idée générale de la solution

2.2.1 Définition du problème

Le problème traité dans ce chapitre est le même que celui qui est abordé au chapitre 1, il s'agit d'un ensemble de capteurs statiques qui est utilisé pour surveiller un ensemble de cibles mobiles. Le transfert de toutes les données collectées est nécessaire, et on est toujours limité à un seul capteur actif à tout instant pour surveiller une même cible. Cependant, on ne considère plus les zones prioritaires. Nous réutilisons les notations précédentes et le principe de discrétisation de la trajectoire d'une cible en faces et fenêtres de temps reste le même. On considère que l'ordonnancement initial a un rayon de stabilité ρ.

Comme mentionné dans l'introduction, le problème abordé ici consiste à réagir dynamiquement à des retards et avances qui sortent du rayon de stabilité de l'ordonnancement initial, afin de maintenir la couverture continue des cibles. Nous prenons ici, comme base, l'ordonnancement produit par la méthode du chapitre précédent qui maximise ρ. Notons que tout ordonnancement réalisable pourrait aussi être utilisé. Cependant, un ordonnancement non robuste qui surveillerait les cibles uniquement pour leurs trajectoires estimées ($\rho = 0$) nécessiterait beaucoup plus de réactions en ligne car le moindre retard ou avance pourrait compromettre son admissibilité. Les activités de surveillance et de communication des capteurs sont modifiées par notre méthode en ligne, certaines supprimées et d'autre ajoutées. De nouvelles notations

sont introduites pour décrire les activités de l'ordonnancement.

L'ordonnancement robuste initial définit pour chaque cible j, un ensemble d'activités de surveillance A_j. Chaque activité $a \in A_j$, correspond à un capteur i_a démarrant une activité de surveillance à la date s_a et la terminant à la date d_a. Le transfert des données n'est pas dépendant du temps car il n'est affecté que par les contraintes de la batterie et peut se faire immédiatement après l'activité de surveillance correspondante. Le routage des données par l'ordonnancement initial est représenté par une fonction $f(i_1, i_2), \forall (i_1, i_2) \in I^2$, qui définit le flot entre le capteur i_1 et i_2, et est toujours égal à 0 si les capteurs ne sont pas à portée.

Si une cible a une avance ou un retard supérieur à la valeur ρ du rayon de stabilité, alors l'ordonnancement n'est potentiellement plus admissible car la cible pourrait n'être plus couverte. Dans ce cas, l'estimation du reste de la trajectoire temporelle est modifiée et décalée, de sorte que toutes les positions futures p d'une cible j ont un nouveau temps estimé $\tau_j(p) = \tau_j(p) \pm \lambda$ avec λ une avance ou un retard. Bien qu'une cible soit temporairement perdue sans réaction, l'ordonnancement garantit toujours la surveillance des autres cibles et garde son efficacité par rapport aux autres métriques (consommation d'énergie, couverture garantie, ...). Il n'est donc pas nécessaire de remettre en question les activités associées aux autres cibles, qui elles n'ont pas de retard ou d'avance en dehors du rayon de stabilité. Le problème est traité dynamiquement, ce qui signifie que pendant que le réseau calcule une solution pour couvrir la cible retardée ou avancée j, la couverture de cette dernière n'est pas garantie. Par conséquent, les solutions recherchées par le problème doivent être calculées rapidement (dans le contexte des applications temps-réel) afin de ne pas perdre la cible trop longtemps ou définitivement. C'est pour cela que chaque fois qu'une cible a un retard ou une avance en dehors du rayon de stabilité, il n'est pas possible de recalculer un nouvel ordonnancement. Il convient seulement de l'ajuster localement.

De plus, la méthode présentée dans ce chapitre vise à garantir dynamiquement ses performances : la méthode construit une solution permettant de couvrir les cibles tant que les retards et les avances restent inférieurs à un rayon de stabilité dynamique Δ_d. En effet, la méthode doit adapter l'ordonnancement aux nouvelles trajectoires temporelles estimées des cibles, en fonction des retards et avances passés. Cependant, dans ce cas, des retards et des avances futurs restent possibles. C'est pourquoi la décision prise en ligne pour ajuster l'ordonnancement à un retard ou à une avance ne doit pas compromettre le potentiel de réaction de la méthode à de futurs écarts.

En effet, une décision prise par la méthode en ligne après un retard ou une avance impacte irrémédiablement le réseau par la suite. Avec une mauvaise décision, la méthode en ligne pourrait ne pas être en mesure de supporter d'autres retards ou avances et donc perdre les cibles, alors qu'une autre décision aurait pu pallier le problème sans conséquences néfastes. Par la suite, nous visons à intégrer une stratégie dans notre méthode en ligne qui tend à préserver,

voire à augmenter, les garanties de performances pour le futur. Par conséquent, la méthode maximise Δ_d, une garantie en ligne telle que les cibles sont assurées d'être couvertes tant que leurs retards et avances restent inférieurs à Δ_d. Δ_d est calculé hors ligne, c'est-à-dire avant le début de l'ordonnancement initial et de la mission. Cependant, cette garantie est aussi dynamique et donc la méthode peut recalculer dynamiquement sa valeur, et ainsi l'augmenter tant qu'aucun retard ou avance n'est supérieur à Δ_d. De plus, chaque fois qu'un retard ou une avance dépasse le rayon de stabilité dynamique, au lieu de perdre la faisabilité et sa robustesse, la méthode a une stratégie de meilleur effort pour garder les cibles couvertes aussi longtemps que possible. Dans ce cas, elle prend les décisions indépendamment de toute considération de retard ou d'avance futur. Au lieu de perdre toute la robustesse Δ_d, dans de tels cas, la méthode peut également être en mesure d'offrir à nouveau des garanties de performance en ligne malgré l'épuisement des garanties précédentes. On dispose donc d'un rayon de stabilité ρ, propre à l'ordonnancement initial, et un rayon de stabilité dynamique Δ_d lié à la méthode dynamique.

Pour résumer, dans ce travail, nous visons à proposer une méthode qui :

— détecte les avances et les retards en dehors du rayon de stabilité ;
— offre dynamiquement une solution pour surveiller les cibles qui seraient perdues en l'absence de réaction ;
— offre des garanties de performance sur les retards et avances futurs ;
— peut être appliquée dans un contexte temps-réel.

La méthode proposée maximise le rayon de stabilité dynamique Δ_d, et lorsqu'un retard ou une avance dépasse le rayon de stabilité dynamique, elle s'efforce de couvrir la cible concernée, mais sans aucune garantie pour faire face aux retards ou avances futurs.

Le tableau 2.1 reprend les nouvelles notations introduites.

A_j	Ensemble des activités de surveillance de l'ordonnancement pour la cible j
i_a	Capteur correspondant à l'activité a
s_a	Date de début de l'activité a
d_a	Date de fin de l'activité a

Tableau 2.1 – Résumé des nouvelles notations introduites

2.2.2 La méthode de résolution – vue générale

Dans cette sous-section, nous présentons un aperçu de la méthode proposée pour résoudre le problème abordé dans ce chapitre. Ensuite, nous détaillons avec plus de précision les différentes phases de notre méthode. Elles sont décrites en détail dans les sections 2.3, 2.4 et 2.5.

La méthode de résolution que nous proposons est divisée en deux phases. La première est une phase hors-ligne et englobe tous les calculs lourds qui peuvent être effectués avant le début de la mission. Cette phase comprend la production de l'ordonnancement initial 1, et le calcul d'un ensemble de solutions partielles appelées chemins de secours (introduites dans la section 2.3) qui pourraient être utilisées dynamiquement pour couvrir temporairement les cibles. La deuxième phase de la méthode se déroule en ligne. Il s'agit de la détection dynamique et de la réaction aux retards et avances, en temps-réel. Afin d'assurer une réaction rapide, notre méthode s'appuie sur les chemins de secours calculées hors-ligne lors de la phase précédente. La phase en ligne décale l'ordonnancement initial pour l'adapter aux retards et aux avances, et couvre temporairement les cibles concernées à l'aide des chemins de secours si l'ordonnancement ne le permet plus. Fondamentalement, la méthode décale l'ordonnancement pour adapter les dates des activités aux nouvelles dates estimées de la trajectoire. La méthode supprime également les activités de surveillance qui ne sont plus nécessaires (en cas d'avance) et en crée de nouvelles, avec les chemins de secours, en cas de besoin. La détection des avances et des retards est présentée dans la section 2.4 et la réaction dans la section 2.5.

2.3 Chemins de secours

Dans cette section, nous présentons les chemins de secours. Un chemin de secours est une solution temporaire pour couvrir une cible lorsque celle-ci n'est plus couverte par l'ordonnancement. Il s'agit donc d'un ensemble de capteurs qui surveillent les cibles et transfèrent les données à la station de base. Notre objectif est de disposer, pour chaque cible, d'un ensemble de chemins de secours pour la couvrir, partout où elle peut se trouver. Par conséquent, pour chaque face visitée par une cible, la méthode calcule un ensemble de chemins. Les chemins de secours ne sont utilisés qu'avec des retards, car la méthode de ré-ordonnancement (section 2.5) permet de faire face à toute avance.

2.3.1 Présentation des chemins de secours

Soit C_f l'ensemble des chemins de secours de la face f. Un chemin $R \in C_f$ est défini comme un ensemble de capteurs qui pourront être utilisés pour couvrir une cible dans la face f, lorsqu'elle reste plus longtemps que prévu dans cette face. Les chemins de secours sont calculés et mis en œuvre hors-ligne dans le réseau. Par conséquent, les décisions en ligne sont limitées au choix du chemin de secours à utiliser lorsque c'est nécessaire, voire à en sélectionner un autre quand le premier est inutilisable faute d'énergie, et ainsi de suite tant que la cible prolonge son séjour dans la face. Ainsi, les chemins de secours offrent une solution rapide et efficace pour couvrir temporairement une cible puisque tous les calculs ont été ef-

fectués hors ligne. Un chemin $R \in C_f$, avec $R = \{i_1, \ldots, i_{|R|}\}$, doit pouvoir surveiller la cible à l'intérieur de f et transférer les données collectées vers la station de base. Par conséquent, le premier capteur i_1 du chemin R est inclus dans $S(f)$. C'est le capteur qui sera utilisé pour surveiller la cible. Les autres capteurs du chemin servent à transférer les données, donc pour tous les $i_n \in R$ avec $n > 1$ le n-ème capteur de R, i_n est dans la portée de communication de i_{n-1} et la station de base est dans la portée de $i_{|R|}$.

Chaque chemin $R \in C_f, \forall f \in F$ a une capacité c_R qui représente le temps total pendant lequel le chemin R peut être utilisé pour surveiller une cible dans f. Ainsi, la quantité maximale d'énergie consommée par un chemin R dans un capteur $i_n \in R$ est $(p^R + p^T) \times c_R$ pour $|R| \geq n > 1$ et $(p^S + p^T) \times c_R$ si $n = 1$. Évidemment, chaque capteur d'un chemin R doit avoir suffisamment d'énergie disponible dans sa batterie pour supporter la consommation de R.

Nous définissons deux types de chemins de secours (appelés type 1 et type 2), qui seront tous deux utilisés pour surveiller les cibles et transférer les données, mais ces deux types de chemin ont des objectifs différents, comme indiqué dans la suite de cette section. Chaque face f a un ensemble C_f^1 de chemins de secours de type 1, et un ensemble C_f^2 de chemins de type 2, tels que $C_f^1 \cup C_f^2 = C_f$. Enfin, la capacité totale d'un ensemble C de chemins, c'est-à-dire la somme des capacités de ces derniers, est notée $c(C)$.

Les nouvelles notations introduites dans la présente section apparaissent dans le tableau 2.2.

C_f	Ensemble de chemins de secours dans la face f
C_f^1	Ensemble de chemins de secours de type 1 dans la face f
C_f^2	Ensemble de chemins de secours de type 2 dans la face f
c_R	Capacité d'un chemin R
$c(C)$	$\sum_{R \in C} c_R$, Somme des capacités des chemins de C

Tableau 2.2 – Résumé des notations introduites pour les chemins de secours

2.3.2 Chemins de secours de type 1

L'objectif des chemins de secours est de couvrir temporairement les cibles, en utilisant l'énergie disponible du réseau. Par conséquent, afin de fournir une garantie de performance, nous devons avoir des garanties sur la couverture disponible par les chemins de secours dans toutes les faces que les cibles visiteront. Le problème ici est que de nombreux retards peuvent avoir été subis par le réseau, en utilisant les chemins de secours des autres faces, avant qu'une cible ne se présente dans une face f. Par conséquent, la couverture effectivement disponible offerte par les chemins de secours d'une face est impossible à prévoir avant que la cible n'arrive dans cette face. Il faut donc pour définir des garanties de performance avoir une approche robuste (dit de pire cas). Ainsi, nous définissons un ensemble robuste de chemins de

secours (chemins de secours de type 1), qui ont leurs capacités minimales garanties tant que seulement d'autres chemins de type 1 auront été utilisés pour couvrir les retards et avances des cibles.

Pour chaque face $f \in F$, nous définissons un ensemble de chemins de secours de type 1, appelé C_f^1. Ces chemins sont calculés hors ligne et doivent garantir que l'utilisation d'un chemin $R \in C_f^1, \forall f \in F$ n'affecte pas la capacité des autres chemins $R' \in C_f^1, \forall f \in F$, ainsi que les activités programmées dans l'ordonnancement. Ainsi, le réseau peut supporter la consommation énergétique qui résulte de l'utilisation de tout chemin $R \in C_f^1, \forall f \in F$ pendant c_R unité de temps, en plus de l'ordonnancement initial, sans compromettre ni cet ordonnancement, ni la capacité des autres chemins de secours. Pour cela, nous devons construire tous ces chemins $R \in C_f^1, \forall f \in F$, de telle sorte que la somme de leurs capacités (donc de leur consommation maximale) et de la consommation future de l'ordonnancement initial ne dépasse pas l'énergie disponible dans les capteurs. Ainsi, pour chaque capteur i, la consommation des activités de l'ordonnancement ne sera pas compromise par la consommation maximale des chemins de type 1, qui se limitent à E_i. Avec une telle stratégie, nous obtenons la garantie que l'utilisation des seuls chemins de secours de type 1 et l'application de l'ordonnancement ne compromettent la capacité d'aucun autre chemin de type 1, par construction de ces chemins. Par conséquent, tant que nous n'utilisons que des chemins de type 1 et l'ordonnancement, nous pouvons calculer une limite inférieure sur $c(C_f)$ pour chaque $f \in F$, qui est la valeur de $c(C_f^1)$.

Notre méthode utilise des chemins de secours pour couvrir temporairement la cible en attendant que l'ordonnancement initial légèrement décalé, donc n'induisant pas de consommation supplémentaire, redevienne réalisable (la section 2.5 traite ce point plus en détail). Le rayon de stabilité dynamique Δ_d qui exprime la garantie de performance dynamique offerte par les chemins de secours de type 1 dépend donc, à tout instant, des capacités résiduelles de ces chemins de secours. La méthode en ligne proposée a donc recours prioritairement aux chemins de secours de type 1, qui ont des capacités connues au préalable sur toutes les faces si seulement des chemins de type 1 sont utilisés. Ainsi, Δ_d peut être garanti en utilisant les valeurs des capacités $c(C_f^1), \forall f \in F$. Notons qu'une face f peut être visitée par plusieurs cibles et plusieurs fois par la même, mais utilise le même C_f pour couvrir ces cibles pour chaque visite. Par conséquent, nous définissons Visit(f) le nombre de fois où une face f est visitée. Soit Λ_f qui définit l'ensemble des valeurs des retards et des avances, qui seront couverts par les chemins de type 1 dans la face f. Chaque $\lambda \in \Lambda_f$ est une valeur, négative si avance, positive si retard. Par définition, Δ_d est garanti tant que $\lambda \leq \Delta_d, \forall \lambda \in \Lambda_f, f \in F$. De plus, Visit$(f) \geq |\Lambda_f|$ car une face f ne peut pas avoir plus de retards ou d'avances que le nombre de fois où elle est visitée. Donc pour chaque face f :

$$\sum_{\lambda \in \Lambda_f} \lambda \leq \Delta_d \text{Visit}(f)$$

Δ_dVisit(f) est donc le temps de couverture minimum nécessaire dans la face f pour garantir le rayon de stabilité dynamique Δ_d. Il s'agit de la couverture totale nécessaire dans la face f pour couvrir tous les passages des cibles dans cette face si chaque passage a un retard égal au rayon de stabilité dynamique Δ_d. Par conséquent, la capacité totale des chemins de type 1 $c(C_f^1), \forall f \in F$, doit être supérieure à Δ_dVisit(f). Nous obtenons alors la formalisation suivante :

$$\begin{aligned} \Delta_d \text{Visit}(f) &\leq c(C_f^1) \quad \forall f \in F. \\ \Delta_d &\leq \frac{c(C_f^1)}{\text{Visit}(f)} \quad \forall f \in F. \end{aligned}$$

On en déduit :

$$\Delta_d \leq \min_{f \in F} \frac{c(C_f^1)}{\text{Visit}(f)}.$$

cela signifie que pour maximiser le rayon de stabilité dynamique de notre stratégie, nous devons maximiser la couverture de la face la moins bien couverte par les chemins de secours de type 1. Dans ce but, les ensembles initiaux de $C_f^1, \forall f \in F$ sont obtenus en résolvant un seul programme linéaire qui maximise Δ_d. Ce programme linéaire calcule les flots entre les faces (sources) et la station de base (puits). Ensuite, ces flots sont décomposés en chemins de secours de type 1. Le programme linéaire résolu est représenté par le modèle 2.1.

Maximiser Δ_d			(1)
Visit$(f)\Delta_d$	$\leq \sum_{i \in S_f} x_{if}$	$\forall f \in F$	(2)
$p^S \sum_{\forall f \mid i \in S_f} x_{if} + p^T \sum_{\forall j \in N(i)} f_{ij} + p^R \sum_{\forall j \in N(i)} f_{ji}$	$\leq ER_i$	$\forall i \in I$	(3)
$\sum_{\forall f \mid i \in S_f} x_{if} - \sum_{\forall j \in N(i)} f_{ij} + \sum_{\forall j \in N(i)} f_{ji}$	$= 0$	$\forall i \in I$	(4)
x_{if}	≥ 0	$\forall f \in F, i \in S_f$	(5)
f_{ij}	≥ 0	$\forall i \in I, j \in N(i)$	(6)
Δ_d	≥ 0		(7)

Modèle 2.1 – Calcul des chemins de secours de type 1

Ce modèle s'appuie sur deux types de variables de décision en plus de Δ_d. x_{if} représente un flot de données entre une face f et un capteur $i \in S(f)$. Plus précisément, il représente la durée des activités de surveillance par le capteur i dans la face f, et donc dans les chemins de secours de type 1 de cette face. f_{ij} est le flot d'informations transmises du capteur i au capteur j. Dans ce programme linéaire, la fonction objectif consiste à maximiser Δ_d, qui avec (2) et (7) est borné entre 0 et la somme des capacités des chemins de secours de type 1 de la face la moins couverte ($\forall f \in F, c(C_f^1) = \sum_{i \in S_f} x_{if}$). Les contraintes (3) traduisent les contraintes énergétiques induites par l'énergie résiduelle contenue dans les batteries des capteurs (ER_i pour

un capteur i, est donc le niveau de batterie initialement disponible moins l'énergie nécessaire pour l'ordonnancement initial). Enfin, les contraintes (4) sont les contraintes de conservation du flot, c'est-à-dire que toutes les données collectées et reçues par un capteur doivent être transmises.

Bien qu'il maximise le rayon de stabilité, ce modèle présente l'inconvénient de produire des solutions pour lesquelles la consommation des chemins de secours n'est pas nécessairement minimale. Par exemple, il est possible que les chemins obtenus par cette résolution soient inutilement long (un chemin qui fait un détour, ou des circuits sur plusieurs capteurs avant un envoi à la station de base). Par conséquent, notre méthode résout un deuxième programme linéaire, avec la valeur de Δ_d fixée à sa valeur optimale trouvée par le programme linéaire précédent. La fonction objectif de ce second programme linéaire consiste à minimiser la consommation dans le réseau de ces chemins, soit la somme $\forall i \in I$ de la partie gauche des contraintes (3). La résolution de ce deuxième programme linéaire, transforme en réalité le calcul des chemins de type 1 en un problème à deux objectifs ordonnés de manière lexicographique, où nous maximisons le rayon de stabilité dynamique en premier lieu, puis nous minimisons ensuite la consommation énergétique des chemins de secours produits.

Dans la section 2.5, nous présentons les ajustements possibles des chemins de secours pour optimiser dynamiquement l'objectif Δ_d. Plus l'ensemble initial des chemins de type 1 est grand, plus la méthode disposera de possibilités d'ajustement, permettant d'offrir de meilleures performances. Ainsi, nous ajoutons dans C_f^1 les routes utilisées par l'ordonnancement initial pour couvrir les cibles en f. Ces chemins sont ajoutés avec une capacité égale à 0, de sorte qu'ils ne compromettent pas les chemins précédents et la consommation totale. Ils donneront seulement plus de possibilités d'augmenter le rayon de stabilité dynamique. Sans ces routes, les instances pour lesquelles la valeur initiale de Δ_d est nulle n'ont aucun chemin de secours de type 1, et donc aucune possibilité d'augmenter Δ_d par la suite.

2.3.3 Les chemins de secours de type 2

Les chemins de secours de type 2 sont calculés par une stratégie de type meilleur effort. Ici, il s'agit de prolonger la couverture d'une cible dans une face donnée, sans se préoccuper des conséquences que cela peut avoir sur la capacité du réseau à faire face à de futurs retards ou avances. On utilise ces chemins de secours en dernier recours, lorsqu'il n'y a plus de chemin de type 1 disponible dans cette face. Le cas se produit lorsque la cible accuse un retard supérieur au rayon de stabilité Δ_d. Dans ce cas, les garanties de performance du réseau sont perdues, et le seul objectif des chemins de type 2 est de couvrir les cibles le plus longtemps possible. Comme la stratégie robuste a échoué, on ne se soucie maintenant plus des retards futurs et on doit juste s'assurer de ne pas compromettre l'ordonnancement. Évidemment, comme on ne connaît pas le retard à supporter dans ce cas, le temps de couverture demandé aux chemins de

secours est également inconnu. L'idée est donc, dans chaque face f, de disposer de chemins de secours permettant de couvrir une cible le plus longtemps possible. C'est pour cela que la méthode de calcul proposée cherche à maximiser la capacité totale des chemins de secours dans une face. Pour une face f, les chemins de C_f^2 sont utilisés une fois que ceux de C_f^1 ne peuvent plus l'être faute d'énergie. Ici, la stratégie est myope aux possibles retards et avances des autres faces, donc lors du calcul de C_f^2, il n'y a pas de considération pour les $C_{f'}, \forall f' \in F$. Ainsi, pour chaque face f, nous résolvons le programme linéaire du modèle 2.2 qui maximise $c(C_f^2)$.

$$\text{Maximiser} \sum_{i \in \mathbf{S(f)}} x_i \tag{8}$$

$$p^S x_i + p^T \sum_{\forall j \in N(i)} f_{ij} + p^R \sum_{\forall j \in N(i)} f_{ji} \leq ER'_i \qquad \forall i \in I \tag{9}$$

$$x_i - \sum_{\forall j \in N(i)} f_{ij} + \sum_{\forall j \in N(i)} f_{ji} = 0 \qquad \forall i \in I \tag{10}$$

$$x_i \geq 0 \qquad i \in I \tag{11}$$

$$f_{ij} \geq 0 \qquad \forall i \in I, j \in N(i) \tag{12}$$

Modèle 2.2 – Calcul des chemins de secours de type 2

Ce modèle s'appuie sur deux types de variables de décision. x_i représente la durée des activités de surveillance par le capteur i dans les chemins de secours de type 2 pour la face f. f_{ij} est le flot de données du capteur i vers le capteur j. La fonction objectif de ce programme linéaire vise à maximiser $\sum_{i \in \mathbf{S(f)}} x_i$, qui est égale à $c(C_f^2)$. Les contraintes (9) limitent les consommations d'énergie dans les capteurs, $\forall i \in I$. Pour un capteur $i \in I$, cette limite est l'énergie résiduelle après l'utilisation des chemins de secours de type 1 et la consommation prévue par l'ordonnancement initial. C'est-à-dire que ER'_i est l'énergie initiale E_i moins l'énergie nécessaire pour l'ordonnancement, moins l'énergie nécessaire pour tous les chemins de secours de type 1 de cette face f. Enfin, les contraintes (10) sont les contraintes de conservation du flot.

Chaque fois que nous résolvons ce programme linéaire pour une face f, comme nous le faisons pour les chemins de secours de type 1, nous résolvons ensuite un deuxième programme linéaire pour réduire la consommation globale des chemins de type 2 obtenus. Dans ce but, la valeur de $\sum_{i \in \mathbf{S(f)}} x_i$ est fixée dans une nouvelle contrainte à sa valeur optimale. Le nouvel objectif du deuxième programme linéaire est de minimiser la consommation énergétique, c'est-à-dire de minimiser l'expression suivante :

$$\sum_{\forall i \in I} \left(p^S x_i + p^T \sum_{\forall j \in N(i)} f_{ij} + p^R \sum_{\forall j \in N(i)} f_{ji} \right)$$

Les chemins de secours de type 2 d'une face f sont calculés en considérant uniquement la consommation des chemins de secours de type 1 de f et de l'ordonnancement. Par conséquent, ils doivent être ajustés dynamiquement chaque fois qu'un chemin correspondant à une autre face est utilisé (quel que soit son type), afin d'être réalisable. Ce sujet est abordé dans la section 2.5.3. Lors de la partie ajustement, la fonction objectif reste la même mais on se limite à juste changer les capacités des chemins de secours et non pas à en recalculer de nouveaux. Alors, afin d'avoir plus d'opportunités lors de l'ajustement en ligne, ce qui conduit à de meilleures performances, nous ajoutons à l'ensemble des chemins de type 2 d'une face f, les chemins de type 1 de cette face f avec des capacités égales à 0. Avec ces capacités nulles, le flot optimal calculé n'est pas compromis puisque ces chemins ne seront pas utilisés tant que leurs capacités (durée maximale d'utilisation) sont égales à 0.

2.3.4 Résumé sur les chemins de secours

Avant la première apparition réelle d'une cible, la méthode calcule hors ligne pour chaque face f qui sera visitée, deux ensembles de chemins de secours C_f^1 et C_f^2. Le premier ensemble, C_f^1, est utilisé pour couvrir une cible tout en garantissant un rayon de stabilité dynamique Δ_d. Ils sont utilisés prioritairement, jusqu'à ce qu'il n'y ait plus besoin de chemins ou jusqu'à ce qu'il soit impossible de les utiliser plus longtemps (capacités atteintes). Si le besoin de surveillance dans la face se prolonge, la garantie sur le rayon de stabilité dynamique Δ_d est perdue, et ne reste que les chemins de secours de type 2, à partir de C_f^2, pour couvrir la cible. Ce deuxième ensemble est calculé pour couvrir aussi longtemps que possible les cibles, dans cette face, mais il est aveugle aux retards et aux avances dans les autres faces, ce qui signifie que leur utilisation peut compromettre les capacités futures du réseau. Ainsi, chaque fois qu'un surcroît de couverture temporelle est nécessaire dans une face f, notre stratégie préconise l'utilisation en premier lieu de C_f^1, donc des chemins de type 1, jusqu'à épuisement puis recours aux chemins de secours de C_f^2. Il n'y a pas de préférence entre des chemins de même type, nous utilisons donc les chemins de type 1 dans un ordre aléatoire, et de même pour les chemins de type 2. Une méthode de sélection plus avancée, où un chemin de type 1 peut être privilégié par rapport à un autre, est discuté dans la section 2.7.

2.4 Détection des avances et des retards

La détection des retards et avances se fait en associant les activités de l'ordonnancement aux faces visitées. Cette association activités-faces est mise à jour à chaque fois qu'une cible quitte une face ou qu'une activité se termine. Lors de cette mise à jour, si certains critères sont remplis, la méthode détecte un retard ou une avance.

2.4.1 Association des activités aux faces

Comme vu dans la section 1.4, les trajectoires des cibles sont représentées par un ensemble de faces visitées, associées à des fenêtres de temps déduites des trajectoires estimées. La k-ième face visitée par une cible j, doit être couverte pendant la fenêtre de temps $[t_k^j - \rho, t_{k+1}^j + \rho]$, où $[t_k^j, t_{k+1}^j]$ est la fenêtre temporelle estimée de la visite et ρ le rayon de stabilité de l'ordonnancement initial. Par conséquent, à chaque instant $t \in [t_k^j - \rho, t_{k+1}^j + \rho]$, le capteur activé par l'ordonnancement pour surveiller la cible j peut être utilisé pour couvrir j dans sa k-ième face visitée. Ainsi, nous associons à toutes les faces $k \in K_j, \forall j \in J$, l'ensemble M_k^j constitué de toutes les activités de surveillance que l'ordonnancement a affectées à la fenêtre de temps correspondante, permettant de couvrir la cible j. Il s'agit de toutes les activités de surveillance ayant une intersection non vide avec la fenêtre de temps $[t_k^j - \rho, t_{k+1}^j + \rho]$. Par conséquent, quels que soient les retards ou les avances à l'intérieur du rayon de stabilité ρ, l'ordonnancement couvrira la cible j dans sa k-ième face visitée avec seulement des tâches de M_k^j.

L'ensemble M_k^j comprend toutes les activités $a \in A_j$ qui vérifient $s_a \geq t_k^j - \rho$ et $s_a \leq t_{k+1}^j + \rho$, donc tous les travaux qui commencent et se terminent dans la fenêtre temporelle correspondante. De plus, on y ajoute des morceaux de certaines activités de l'ordonnancement, qui chevauchent la fenêtre de temps. Ainsi, on ajoute à M_k^j, une nouvelle activité correspondant à une partie de l'activité $a_{start} \in A_j | s_{a_{start}} < t_k^j - \rho, d_{a_{start}} \geq t_k^j - \rho$, à savoir l'activité qui commence avant mais se termine après le début de cette fenêtre de temps. Cette nouvelle activité a juste une date de début différente égale à $t_k^j - \rho$. De même pour l'activité qui commence avant la fin de la fenêtre de temps mais finit après. À partir de cette activité $a_{end} \in A_j | s_{a_{end}} \leq t_{k+1}^j + \rho, d_{a_{end}} > t_{k+1}^j + \rho$, on crée une nouvelle activité dans M_k^j avec la même date de début mais une date de fin différente égale à $t_{k+1}^j + \rho$.

Dans l'exemple de la figure 2.1, L'ordonnancement initial a un rayon de stabilité $\rho = 1$ seconde, et couvre une cible avec deux faces visitées. La face $\{1, 2\}$ est couverte par le capteur 1 et le capteur 2, et $\{2\}$ est couverte uniquement par le capteur 2. Il y a deux activités dans cet exemple, a_1 avec le capteur 1 activé de 0 à 8 secondes et a_2 de 8 à 16. L'ensemble M_1^1 est associé à la visite de la face $\{1, 2\}$ et M_2^1 à la face $\{2\}$. Le tableau 2.3 décrit M_1^1 et M_2^1, chaque ligne étant une activité ou une partie d'une activité de l'un des deux ensembles. La colonne a est l'activité initiale, i_a est le capteur activé, s_a l'instant de début et d_a l'instant de fin. Dans l'ordonnancement, on voit que le job a_2 correspond à la couverture de deux fenêtres temporelles et est donc associé aux deux. Cependant, seule une partie de a_2, de 8 à 11, est associée à M_1^1, de même pour M_2^1 avec 9 à 16 (temps d'entrée estimé dans la face $\{2\}$ moins le rayon de stabilité).

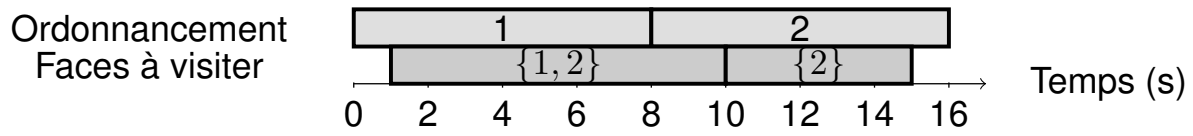

FIGURE 2.1 – Exemple d'un ordonnancement avec rayon de stabilité $\rho = 1$ s

M_k^j	Activité a	Capteur i_a	Date de début s_a	Date de fin d_a
M_1^1	a_1	1	0	8
	a_2	2	8	11
M_2^1	a_2	2	9	16

Tableau 2.3 – Association entre activités et faces dans l'exemple de la figure 2.1

2.4.2 Retards

Un retard est détecté lorsqu'une cible j est toujours à l'intérieur de la k-ième face (ou n'y est pas encore entrée dans le cas de la première face visitée) alors que toutes les tâches de M_k^j sont terminées. Dans ce cas, la cible doit encore être couverte dans cette face, mais l'ordonnancement n'a plus d'activité calculée associée à la couverture de la cible dans cette face. La cible est alors dite « perdue » dans cette face, ce qui met en échec la mission de surveillance continue des cibles. Cet échec est causé par un retard puisque la cible devait quitter une face mais ne l'a toujours pas fait, sortant ainsi des conditions du rayon de stabilité. Afin de détecter de tels retards, pour une cible j dans sa k-ième face, un événement est déclenché dynamiquement chaque fois qu'une activité, ou une partie d'une activité de M_k^j se termine. Ainsi, on peut considérer que la date courante t est égale à la date de fin d'une activité de M_k^j, qu'on peut lister au préalable. Par exemple, dans le tableau 2.3, cet événement sera déclenché deux fois pour M_1^1, à $t = 8$ et à $t = 11$ secondes. Lorsque ce type d'événement est déclenché par une cible j, la méthode supprime toutes les activités qui sont maintenant terminés dans $M_k^j, \forall k \in K_j$. Si l'ensemble M_k^j correspondant à la face visitée actuellement par la cible est vide mais que j ne quitte pas la face, alors un retard en dehors du rayon de stabilité ρ est détecté. En effet, l'ordonnancement n'a pas été calculé pour surveiller plus longtemps la cible dans cette face alors que celle-ci s'y trouve encore.

Dans la figure 2.1, si la cible prolonge son séjour dans la face $\{1, 2\}$ d'une durée supérieure au rayon de stabilité $\rho = 1$ seconde, alors la méthode détecte le retard après deux événements. D'abord à 8 secondes, lorsque la première activité est terminée, un évènement est déclenché qui supprime a_1 de M_1^1, mais aucune alerte pour signaler un retard n'est envoyée, car la cible est toujours à l'heure (il y a toujours une activité programmée pour la face $\{1, 2\}$). À 11 secondes, un deuxième événement est déclenché et supprime l'activité restante de M_1^1. Cette fois, il n'y a plus d'activité dans M_1^1, donc une alerte signalant un retard est émise. En effet, la

méthode peut déduire que la cible est toujours dans la face $\{1, 2\}$ et n'est pas sur le point de la quitter alors que la date actuelle correspond au dernier instant où la cible aurait pu se trouver dans cette face.

2.4.3 Avances

Une avance est détectée lorsque l'activité de surveillance d'une cible qui aurait dû se terminer dans une face n'est pas encore terminé lorsque la cible quitte réellement la face. En d'autres termes, une avance se manifeste lorsqu'une cible change de face (quitte ou arrive dans la portée d'un capteur) et que l'activité en cours n'était pas prévue pour couvrir la cible dans la nouvelle face. C'est-à-dire que cette activité devait se finir avant la date au plus tôt (date estimée moins avance acceptée par le rayon de stabilité) où la cible est attendue dans cette face. Afin de détecter une avance, la méthode dynamique utilise un événement déclenché chaque fois qu'une cible j quitte une face (à l'exception des dernières faces visitées). Lors d'un tel évènement, s'il existe encore une activité $a \in M_k^j$ avec une date de fin $d_a < t_{k+1}^j - \rho$, alors une avance est détectée. En effet, cette activité devait se terminer avant que la cible j sorte de sa k-ième face si celle-ci a une avance égale à ρ. On se trouve donc dans une situation où la sortie d'une face se produit avant l'estimation la plus précoce couverte par l'ordonnancement. Par conséquent, une alerte signalant une avance en dehors du rayon de stabilité est émise.

Si dans l'exemple de la figure 2.1, la cible quitte la face $\{1, 2\}$ à l'instant $t = 6$ secondes, et ainsi entre en avance dans la face $\{2\}$. Un événement est déclenché à l'instant t, car la cible quitte une face. La méthode dynamique vérifie les tâches restantes dans M_1^1 et détecte une activité, a_1, qui n'est toujours pas terminée. Cependant, cette activité est censée s'être terminée avant l'entrée dans la face suivante, car $d_{a_1} < t_2^1 - \rho$. La méthode en déduit que la cible est en avance de plus de ρ unités de temps.

2.4.4 Résumé des détections des avances et des retards

Deux types d'évènements dynamiques déclenchent des appels à la méthode en ligne, pour assurer le suivi de l'ordonnancement et permettre la détection des retards et des avances au-delà du rayon de stabilité. Le premier type d'événement est la fin d'une activité de $M_k^j, \forall j \in J, k \in K_j$. Si un tel événement est déclenché par une cible j séjournant dans sa k-ième face, alors dans un premier temps, les ensembles $M_{k'}^j, \forall k \leq k' \leq |K_j|$ sont mis à jour. Ensuite, la méthode vérifie s'il reste au moins une activité pour couvrir j dans cette face, en consultant M_k^j. Si ce n'est pas le cas, la méthode déduit que la cible est affectée d'un retard supérieur à ρ, ce qui nécessite l'ajout d'activités et le décalage de l'ordonnancement pour y faire face. Le deuxième type d'événement est déclenché chaque fois qu'une cible quitte une face. Dans ce cas, si une cible j quitte sa k-ième face, et qu'il y a encore une activité $a \in M_k^j$ non terminée,

avec une date de fin $d_a < t_k^j - \rho$ alors la cible j est en avance.

2.5 Méthode de résolution

Dans cette section, nous présentons notre stratégie en ligne pour couvrir dynamiquement les cibles chaque fois qu'un retard ou qu'une avance dépasse le rayon de stabilité ρ de l'ordonnancement. La stratégie en ligne est appelée chaque fois qu'une face est quittée par une cible ou qu'un retard est détecté. Elle a un comportement différent si une alerte de retard ou d'avance est émise. Elle vise à proposer une solution pour couvrir les cibles en utilisant la date courante t dans l'horizon temporel, l'ordonnancement initial et les chemins de secours calculés au préalable. Comme mentionné précédemment, nous visons également à garantir les performances de notre stratégie en ligne, en utilisant le rayon de stabilité dynamique Δ_d, qui peut évoluer avec les retards et avances effectifs des cibles. Cette stratégie est également limitée par le contexte temps-réel de sa mise en œuvre, et les décisions et calculs doivent être faits très rapidement. Par conséquent, nous proposons une stratégie en ligne en trois phases. Tout d'abord, la méthode utilise des chemins des secours pour surveiller temporairement la cible lorsque celle-ci a un retard hors du rayon de stabilité. Deuxièmement, on décale l'ordonnancement initial pour l'adapter aux nouvelles dates estimées. Troisièmement, la méthode ajuste les capacités des chemins de secours, afin de refléter l'état réel du réseau et de continuer à conserver, voir d'étendre, les garanties de performances. Il s'agit plus spécifiquement de maximiser l'amplitude des retards et avances auxquels on garantit de pouvoir répondre à l'aide des chemins de secours de type 1, puis de maximiser la capacité des chemins de secours de type 2 dans chaque face.

2.5.1 Avances

La méthode détecte et réagit à une avance lorsqu'une cible quitte une face plus tôt que prévu et met en défaut les activités de l'ordonnancement initial. Bien que, dans ce cas, la cible soit perdue et l'ordonnancement rendu irréalisable, les autres cibles peuvent toujours être couvertes par le reste des activités de surveillance prévue par l'ordonnancement, tant qu'il reste suffisamment d'énergie dans les batteries des capteurs concernés. Par conséquent, seule la partie de l'ordonnancement A_j, avec j la cible en avance, doit être modifiée sans aucun ajustement des autres activités de $A_{j'}, \forall j' \neq j$. La solution que nous proposons dans ce cas consiste à re-planifier uniquement les jobs en A_j, afin de les adapter à la nouvelle trajectoire temporelle estimée de j, qui peut être déduite après son avance. Il n'est pas nécessaire d'ajouter de nouvelles activités dans l'ordonnancement et certaines peuvent même voir leur durée réduite, ce qui permet d'économiser de l'énergie. Par conséquent, l'ordonnancement ne consommera

pas plus d'énergie et il restera réalisable pour les autres cibles. De plus, l'énergie économisée pourra permettre une augmentation de la capacité des chemins de secours afin d'accroître les garanties de performances.

La méthode de résolution que nous proposons, pour une avance de ξ secondes, consiste à avancer les activités de A_j de ξ secondes. En effet, si la cible j quitte ξ secondes plus tôt que prévu sa k-ième face, alors la valeur réelle du tick t^j_{k+1} doit être mise à jour et prend la valeur $t^j_{k+1} - \xi$. Puisque la vitesse estimée de la cible n'est pas modifiée, on peut en déduire que $\forall k' | k+1 \leq k' \leq |K_j| + 1$ la date $t^j_{k'}$ prend la valeur $t^j_{k'} - \xi$, c'est-à-dire que toutes les dates des ticks restants pour la cible sont avancées de ξ secondes. Par conséquent, l'avance actuelle est prise en compte et se répercute sur le reste de la trajectoire. Pour conclure, avec une avance de ξ secondes pour une cible j, si les dates de démarrage des activités de couverture de j sont avancées de ξ secondes, l'ordonnancement est adapté à la nouvelle trajectoire temporelle de j pour le reste de la mission de surveillance. Les autres cibles conservent leurs activités programmées sans aucune modification.

Nous illustrons cette stratégie de prise en compte des avances à la figure 2.2, sur la base de l'exemple de la figure 2.1. Une alerte pour une avance est émise à l'instant $t = 5$ secondes, car la cible quitte 5 secondes plus tôt que prévu la face $\{1, 2\}$, ce qui est au delà du rayon de stabilité ρ. Par conséquent, la face $\{2\}$ voit désormais son intervalle prévisionnel de visite passer de $[10, 15]$ initialement, à $[5, 10]$, puisque la cible est arrivée dans cette face à $t = 5$ secondes. L'ordonnancement est donc simplement décalé à gauche de 5 secondes, et le résultat est représenté à la figure 2.2. La partie en pointillés de cet ordonnancement représente les activités qui à l'instant t sont déjà exécutées, avant le re-planification. Les parties hachurées sont des tâches ou des parties de tâches qui n'ont pas été exécutées à la date t, avant la re-planification, mais sont désormais planifiées pour une date antérieure à la date actuelle t. Par conséquent, ces activités sont désormais réputées avoir été exécutées et sont donc supprimées. Notons par exemple, que l'activité du capteur 2 est désormais réduite, car une partie de celui-ci a été supprimée. Par conséquent, les chemins de secours relatifs aux faces restant à visiter peuvent désormais voir leur capacité augmenter car ils sont initialement calculés pour économiser de l'énergie pour les activités hachurées, qui sont maintenant supprimées et constituent donc un « gisement » d'énergie potentiellement disponible à l'avenir. Enfin, la dernière activité restante, opérée par le capteur 2, a été avancée de 5 secondes pour s'adapter au reste de la trajectoire de la cible.

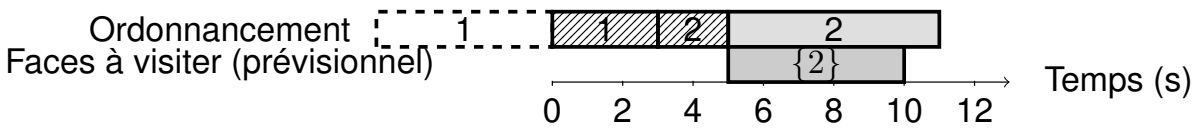

FIGURE 2.2 – Exemple de ré-ordonnancement dynamique lorsqu'une cible est en avance

Comme nous pouvons le voir dans l'exemple précédent, avec une telle stratégie de ré-ordonnancement, le nouvel ordonnancement des activités conserve sa robustesse, et la valeur de ρ est inchangée. De plus, une telle stratégie permet au réseau de ne jamais perdre de cibles en cas d'avance.

2.5.2 Retards

La méthode en ligne reçoit une alerte lorsqu'un délai supérieur à ρ est détecté. Dans ce cas, un ensemble M_k^j est vide pour une cible $j \in J$ et sa fenêtre $k \in K_{,}j$. Il n'y a donc plus d'activité dans l'ordonnancement prévu pour couvrir j dans sa k-ième face. Par conséquent, nous devons d'abord ajouter des activités à l'ordonnancement, pour couvrir la cible à l'intérieur de cette face f, et ce jusqu'à ce qu'elle la quitte. Pour cela, la méthode en ligne utilise des chemins de secours, qui permettent la couverture de la cible pendant une certaine durée. Cette solution a l'avantage de ne pas compromettre les activités restantes puisque les chemins sont calculés en prenant en compte l'énergie nécessaire requis par l'ordonnancement initial. De plus, chaque face dispose d'un ensemble de chemins, dont les capacités font l'objet d'une garantie de performance dynamique mesurée par Δ_d.

Ensuite, si la mission n'est pas terminée, la cible arrive dans sa face $k+1$. Avec un retard de ξ, les dates prévisionnelles des ticks $t_{k'}^j, \forall k' \{k+1, \ldots, |K_j|+1\}$ sont décalées et prennent la valeur $t_{k'}^j = t_{k'}^j + \xi$. Par conséquent, la partie restante de l'ordonnancement est à nouveau décalée (à droite) pour s'adapter à la nouvelle valeur des ticks. Cependant, contrairement aux cas des avances, nous ne pouvons pas nous contenter de différer les tâches restantes de ξ secondes. Par exemple, c'est le cas si dans la figure 2.1 la cible est retardée de 5 secondes et commence sa mission en retard. Ici, on considère qu'elle quitte la face vide et arrive dans la face suivante, $\{1, 2\}$, à 5 secondes au lieu de 1. Alors, une alerte est émise à l'instant $t = 2$ secondes, lorsque le retard dépasse le rayon de stabilité $\rho = 1$ seconde. Par conséquent, l'ordonnancement a déjà commencé l'activité du capteur 1 dans l'intervalle $[0, 2]$. Si les activités sont retardées de $\xi = 5$ secondes alors nous obtenons la solution présentée à la figure 2.3. La partie en pointillés correspond à une tâche déjà exécutée. Comme nous pouvons le voir, avec une telle stratégie, la méthode en ligne provoque un problème dans la face $\{1, 2\}$, qui est la face suivante visitée par la cible, ainsi que dans la face courante (ici la face vide). La cible est perdue et une activité déjà exécutée est répétée, ce qui induit une perte d'énergie non prise en compte lors du calcul de l'ordonnancement. Cette situation en compromet l'admissibilité. Cela est dû au fait que l'ensemble des activités calculé pour couvrir la face $\{1, 2\}$, l'ensemble M_{k+1}^j (M_1^j dans notre exemple), a déjà commencé lorsque le retard est détecté. En effet, les jobs qui couvrent ces faces ont déjà été exécutés pendant 2ρ secondes afin de couvrir la cible dans le cas d'une avance ou d'un retard inférieur à la valeur du rayon de stabilité. Pour pallier ces problèmes, nous proposons de décaler les activités de seulement $\rho - \xi$ secondes. Dans

le même exemple de la figure 2.1, toujours avec un retard de 5 secondes pour entrer dans la face $\{1, 2\}$, nous obtenons l'ordonnancement présenté à la figure 2.4. Cependant, bien que l'ordonnancement redevienne apte à couvrir la cible avec sa nouvelle trajectoire temporelle estimée, il perd sa robustesse. Il ne sera donc capable de couvrir les cibles qu'en l'absence de retard et d'avance. La méthode en ligne considérera donc désormais que le rayon de stabilité ρ est égal à 0, et elle sera appliquée par la suite chaque fois qu'une cible aura un retard ou avance (supérieur à 0, la nouvelle valeur de ρ) par rapport aux nouveaux ticks estimés.

Grâce à une telle stratégie, nous sommes en mesure de garantir la couverture des cibles aussi longtemps que le permet la capacité des chemins de secours dans les faces où les cibles sont en retard. Par conséquent, la stratégie en ligne a des performances garanties (couverture des cibles) tant que le retard maximum ne dépasse pas Δ_d, et donc la capacité totale minimale des chemins de type 1 dans une face f divisée par le nombre de fois où f est visitée. Nous ne pouvons cependant pas garantir de couvrir les retards $\rho + \Delta_d$ à un instant t avec ρ supérieur à 0. En effet, tout futur retard supérieur à ρ et inférieur à $\Delta_d + \rho$ décalera l'ordonnancement, qui perdra ainsi son rayon de stabilité initial (ρ deviendra nul).

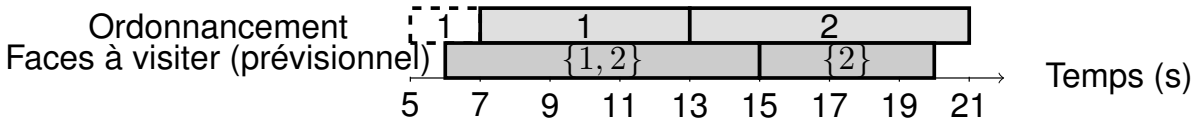

FIGURE 2.3 – Exemple d'un ordonnancement retardé de ξ secondes

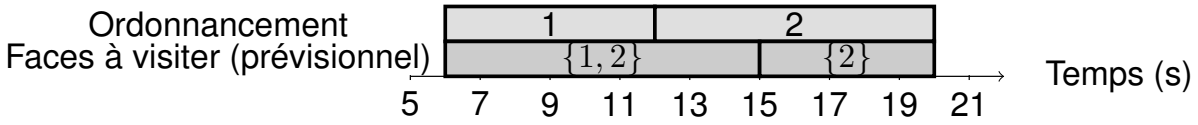

FIGURE 2.4 – Exemple d'un ordonnancement retardé de $\xi - \rho$ secondes

2.5.3 Ajustement des chemins de secours

Dans cette sous-section, nous présentons les ajustements dynamiques des chemins de secours. L'ajustement est la modification des capacités des chemins, afin soit de garder une solution réalisable (chaque fois qu'une cible a un retard supérieur à Δ_d, car les capacités des chemins peuvent avoir été impactées) soit de maximiser dynamiquement Δ_d. Ces ajustements sont effectués dans un contexte temps-réel, et devraient donc consommer peu de puissance de calcul, d'autant que les chemins sont potentiellement ajustables chaque fois qu'une cible quitte une face. Ces ajustements doivent être effectués car :

— Si une avance est détectée, des activités sont supprimées ce qui libère l'énergie qui leur avait été réservée (réf. Section 2.5.1).

— Si un retard supérieur à Δ_d est détecté, des chemins de secours de type 2 sont utilisés, ce qui impacte potentiellement la capacité de tous les autres chemins (réf. Section 2.3). Par conséquent, nous devons ajuster les capacités des chemins des types 1 et 2 des autres faces, car certains peuvent ne plus être utilisable si leur capacité devient nulle.
— Lorsqu'une face f n'est plus visitée par aucune cible, l'énergie nécessaire à C_f^1 peut être réaffectée pour maximiser le rayon de stabilité dynamique dans les faces qui peuvent encore être visitées (réf. Section 2.3).

Un recalcul complet des chemins de secours vers la solution optimale, comme vu dans la section 2.3 n'est pas considéré, car le calcul de nouveaux chemins et leur implémentation dans le réseau prendrait beaucoup trop de temps et n'est donc pas applicable dans un contexte temps-réel. En effet pendant le recalcul des chemins de secours, les cibles sont toujours en déplacement, et la mission continue son déroulement. Ainsi, une cible nécessitant des chemins de secours suite à un retard pourrait alors ne pas être couverte si le recalcul était toujours en cours. A contrario, une mise à jour des capacités des chemins semble plus adaptée, avec un calcul rapide et une mise en œuvre aisée. En effet, la solution que nous proposons dans cette sous-section, consiste simplement à recalculer la capacité des chemins existants. Pour cela, on utilise des programmes linéaires simples, qui contrairement à la section 2.3, ont peu de variables de décision et de contraintes et qui sont ainsi résolus plus rapidement.

Comme mentionné dans cette section, si les chemins sont utilisés pour couvrir une cible avec un retard supérieur à Δ_d, l'ajustement est obligatoire. Cependant, dans les autres cas, l'ajustement n'est pas obligatoire mais cette adaptation en ligne peut permettre d'offrir de meilleures performances aux chemins ainsi qu'une meilleure garantie de performance dynamique mesurée par Δ_d. Plusieurs stratégies d'ajustement sont envisagées. La stratégie par défaut, utilisée dans notre méthode en ligne, consiste à ajuster les chemins chaque fois qu'une cible quitte une face (franchissement de la limite de la portée d'un capteur). Avec une telle stratégie, la méthode en ligne maximise Δ_d aussi souvent que possible. Plus précisément, un ajustement a lieu dès qu'il y a un changement dans les contraintes des programmes linéaires des modèles 2.1 et 2.2 (modification ou suppression de contraintes). Cependant, dans certains cas extrêmes avec beaucoup de faces visitées en peu de temps, une telle stratégie peut ne pas être applicable dans un contexte temps-réel. Nous définissons une deuxième stratégie, qui opère un ajustement après qu'une cible quitte une face si et seulement si un chemin de secours a été utilisé, ou si le dernier ajustement date d'au moins x secondes, où x est un paramètre. Ce paramètre x permet de trouver un compromis entre la valeur de Δ_d et le temps de calcul consommé en ligne, car des valeurs plus élevées induisent moins d'ajustements donc de plus faibles valeurs de Δ_d. La version par défaut de notre méthode correspond à $x = 0$, et dans notre section expérimentation nous testons et comparons les résultats obtenus avec $x = 1$. En d'autres termes, nous ajustons seulement une fois par seconde si aucun retard n'utilise de

chemins de secours. Par la suite, cette deuxième option est nommée la méthode à période contrôlée.

Cas des chemins de secours de type 1

Comme introduit dans la section 2.3, les chemins de secours de type 1 visent à maximiser le rayon de stabilité. Ce rayon, Δ_d, dépend directement de leurs capacités. Nous considérons maintenant uniquement les chemins associés aux faces qui restent à visiter à l'instant courant, et non plus le problème considérant toutes les faces à visiter pendant la mission comme dans la section 2.3. Par conséquent, le programme linéaire résolu considère de moins en moins de faces à chaque appel, et résout donc le modèle 2.3, qui devient de plus en plus facile.

$$\text{Maximiser } \Delta_d \tag{13}$$

$$\text{Visit}(f) \times \Delta_d \leq \sum_{r \in C_f^1} x_r \quad \forall f \in F_{left} \tag{14}$$

$$\sum_{r \in C_f^1, \forall f \in F_{left} | i \in r} x_r \times u_{r,i} \leq ER_i \quad \forall i \in I \tag{15}$$

$$x_r \geq 0 \quad \forall f \in F_{left}, r \in C_f^1 \tag{16}$$

Modèle 2.3 – Ajustement des capacités des chemins de secours de type 1

Nous définissons F_{left} comme l'ensemble des faces qui doivent encore être visitées au moins une fois. C'est-à-dire telles que Visit$(f) > 0$, avec Visit(f) le nombre de fois que la face f devra encore être visitée, et tel que quand une cible quitte une face f, Visit$(f) =$ Visit$(f) - 1$. Le modèle n'a qu'un seul type de variables de décision, x_r, associée à la capacité d'un chemin de secours r de type 1 qui permet la couverture d'une face $f \in F_{left}$. Le nombre de variables de décision dans ce programme linéaire est égal à $\sum_{f \in F_{left}} |C_f^1|$. L'objectif est de maximiser Δ_d qui est lié avec les contraintes (14) à la face la moins couverte par les chemins de secours de type 1. Dans les contraintes (15), $u_{r,i}$ est la consommation du chemin r dans le capteur i. C'est-à-dire $p^S + p^T$ si le capteur est utilisé pour surveiller une cible dans ce chemin r, $p^R + p^T$ si le capteur relaie uniquement les données dans r, et 0 sinon. Les contraintes (15) représentent la consommation d'énergie limitée, dans chaque capteur i, par ER_i l'énergie restante dans la batterie du capteur i moins l'énergie nécessaire pour le reste de l'ordonnancement. Après un pré-traitement rapide, nous ne gardons que les contraintes (15) pour les capteurs qui sont utilisés au moins une fois dans les chemins de type 1. On peut noter que le nombre de variables de décision (une par chemin de type 1) ainsi que le nombre des contraintes (le nombre de faces plus le nombre de capteurs apparaissant au moins une fois dans les chemins) sont relativement faibles. On obtient donc un modèle facile à résoudre, lorsque le nombre de faces visitées n'est pas extrêmement élevé. Un tel problème nous semble pouvoir être résolu dans un contexte de temps-réel.

À moins qu'il n'ait fallu faire face à un retard important, c'est-à-dire supérieur à Δ_d, la méthode est susceptible d'augmenter la valeur du rayon de stabilité dynamique chaque fois qu'une cible quitte une face. En effet, à chaque itération, soit on considère moins de faces, soit le nombre de visites d'une face, Visit(f), diminue. Dans les deux cas, le modèle est moins contraint par (14) et moins d'énergie est nécessaire pour maintenir la valeur de Δ_d, créant ainsi une marge d'amélioration potentielle. De plus, on considère moins de chemins et donc moins de capteurs, ce qui contribue à rendre le modèle de plus en plus facile à résoudre. Enfin, l'énergie résiduelle considérée ER_i dans les contraintes (15), pour un capteur i, augmente également en cas d'avance puisque certaines activités sont supprimées de l'ordonnancement. Pour conclure, avec Δ_{dt} la valeur de Δ_d à l'instant t, s'il n'y a pas de retard $\xi > \Delta_{dt}$ à l'instant t, nous avons la garantie d'obtenir $\Delta_{dt} \leq \Delta_{dt+1}$, et donc une augmentation de Δ_d, car les capacités des chemins restants sont de moins en moins limitées par les contraintes (15).

Dans le cas où un retard $\xi > \Delta_d$ se produit, le rayon de stabilité dynamique n'est pas entièrement perdu, car en mettant à jour les capacités des chemins de secours de type 1 avec le programme linéaire du modèle 2.3, on peut obtenir une nouvelle valeur du rayon de stabilité dynamique strictement supérieure à 0. En effet, le modèle est toujours moins contraint et, de plus, le retard ne s'est pas forcément produit dans une face contraignante. Si dans de tels cas Δ_d diminue généralement, il peut aussi arriver qu'il augmente ou qu'il soit inchangé.

Cas des chemins de secours de type 2

Comme pour les chemins de type 1, l'ajustement des chemins de secours de type 2 correspond au calcul de nouvelles capacités pour chaque chemin, en utilisant un modèle similaire à celui de la section 2.3. Puisque les capacités des chemins de type 2 dans C_f^2 sont calculées en considérant les capacités des chemins de type 1 dans C_f^1, pour toutes les faces $f \in F$, alors chaque fois que ces dernières sont ajustées, les capacités des chemins de type 2 doivent également être ajustées. Nous ajustons les capacités de C_f^1 pour toutes les faces restant à visiter $f \in F_{left}$, chaque fois qu'une cible quitte une face. Cependant, il n'est pas nécessaire d'ajuster aussi les capacités de C_f^2 pour toutes ces faces. En effet, les capacités des chemins de secours de type 2, d'une face f, n'ont besoin d'être à jour qu'aux instants où une cible est à l'intérieur de f. La cible peut alors nécessiter des chemins de secours pour un supplément de couverture s'il y a un retard. Or, pour toute face $f \in F$, les capacités de C_f^2 ne sont plus garanties dès que les capacités de C_f^1 sont modifiées, donc quand une cible quitte une face. Il suffit alors d'ajuster les capacités des chemins de type 2 uniquement pour les faces qui sont couramment visitées et qui peuvent nécessiter des chemins avant un prochain ajustement. Il s'agit de toute face où une cible vient tout juste d'arriver et de toute face dans laquelle se trouve une cible qui n'est pas sur le point de la quitter.

Un programme linéaire est résolu pour chacune de ces faces. Par conséquent, le nombre

maximum de programmes linéaires résolus, pour ajuster les itinéraires des chemins de type 2, est égal au nombre de cibles. Si deux cibles ou plus sont dans la même face, cette face n'a besoin que d'un seul ajustement. Le programme linéaire résolu pour une face f est le modèle 2.4.

$$\text{Maximiser} \sum_{r \in C_f^2} x_r \tag{17}$$

$$\sum_{r \in C_f^2 | i \in r} x_r u_{r,i} \leq ER'_i \quad \forall i \in I \tag{18}$$

$$x_r \geq 0 \quad \forall r \in C_f^2 \tag{19}$$

Modèle 2.4 – Ajustement des chemins de secours de type 2

Là encore, il n'y a qu'un seul type de variables de décision, x_r la capacité d'un chemin $r \in C_f^2$. Cela signifie que le nombre de variables de décision, lors de la résolution du programme linéaire d'une face f, est égal à son nombre de chemins de type 2, c'est-à-dire $|C_f^2|$, qui est généralement faible. Les contraintes (18) sont similaires aux contraintes (15), dans le modèle 2.3, tout en ne considérant que les chemins de la face f et avec ER'_i l'énergie des batteries à laquelle on a soustrait l'énergie nécessaire pour l'ordonnancement restant et pour les chemins de C_f^1.

L'augmentation de $c(C_f^1)$ induit une diminution de $c(C_f^2)$, car ER'_i diminue. Ainsi les valeurs des capacités totales des chemins de type 2 dans toutes les faces $f \in F$ évoluent à l'opposé de Δ_d : quand Δ_d augmente, les valeurs des $c(C_f^2)$ ont tendance à diminuer.

Tel que mentionné dans la section 2.3, l'ensemble C_f^2 contient initialement aussi tous les chemins de C_f^1 avec des capacités nulles. Cela permet dans le modèle 2.4 d'offrir plus de possibilités pour maximiser la fonction objectif.

2.6 Expérimentations numériques

2.6.1 Questions abordées

Les expérimentations présentées dans cette section sont conçues pour répondre à des questions sur l'efficacité de la méthode proposée. Les questions sont les suivantes :

- **Q 1** : Comment la méthode réagit-elle aux divers cas de retards et d'avances ?
- **Q 2** : Quelle est la nature et l'amplitude du rayon de stabilité dynamique offert par la méthode ?
- **Q 3** : L'ajustement dynamique des chemins de secours reste-t-il viable dans un contexte temps-réel, lorsque le nombre de cibles va croissant (plus de faces visitées) ?
- **Q 4** : La méthode est-elle toujours utilisable dans un contexte temps-réel avec un nombre croissant de capteurs (chemins de secours plus nombreux et plus longs) ?

— **Q 5** : Comment se comporte la méthode contrôlée par période comparé à la méthode par défaut ?

Les deux premières questions **Q 1**, **Q 2**, étudient l'efficacité de la méthode en ligne et la robustesse qu'elle procure pour répondre aux retards et avances des cibles. Elles étudient l'impact de différents scénarios sur le temps CPU et sur la valeur du rayon de stabilité dynamique, par rapport au rayon de stabilité initial. Ces questions visent à montrer l'importance des ajustements, et notamment leur capacité à offrir de meilleures garanties de performances et ainsi l'importance du re-calcul dynamique de Δ_d. Les questions restantes étudient l'application en temps-réel de la méthode. Premièrement, **Q 3** étudie l'efficacité de la version par défaut face à une augmentation du nombre de cibles, et ainsi du nombre de faces. **Q 4** se concentre sur l'augmentation du nombre de capteurs dans le réseau, lié à la taille et au nombre des chemins de secours. Enfin, la dernière question évalue le compromis potentiel entre efficacité et temps consommé que peut atteindre la version à période contrôlée.

2.6.2 Description des instances

Dans cette section, nous présentons les instances utilisées dans les expérimentations numériques. L'objectif de nos expérimentations est d'appliquer la méthode en ligne sur des instances aux caractéristiques différentes et d'analyser les résultats lorsque l'on fait varier certains paramètres. Les instances sont produites par le même générateur que celui qui est présenté au chapitre 1 (section 1.7.2), et qui génère les capteurs, leurs positions, et la trajectoire des cibles. Les paramètres utilisés sont principalement les mêmes, mais leurs valeurs par défaut ont été modifiées afin de produire des instances qui ont plus d'intérêt pour ce chapitre. Les nouvelles valeurs sont présentées dans le tableau 2.4. On précise ici que toutes les cibles se déplacent dans le même intervalle de temps $H = 60$ secondes, et qu'on ne considère plus de zones prioritaires.

Paramètre	Valeur
Nombre de capteurs m	100
Énergie dans les capteurs $[E_{min}, E_{max}]$	$[400, 600]$
Nombre de cibles n	2
Dimension de la zone $L_1 \times L_2$	300×300
p^R	1
p^T	1
p^S	2.8
Portée de surveillance R_s	35
Portée de communication R_c	70
H	60

Tableau 2.4 – Valeurs par défaut pour le générateur d'instances

2.6.3 Description des scénarios

Un scénario est constitué des trajectoires temporelles réelles des cibles au cours d'une mission, avec la véritable date de passage dans les faces. En pratique, ce sont les dates réelles des ticks et donc le temps passé dans chaque face par les cibles. Avec les scénarios, nous sommes capables de simuler la mission des cibles et de tester l'efficacité de notre méthode. Chaque scénario doit être adapté à l'instance sur laquelle il va être appliqué. Par conséquent, nous développons un générateur de scénarios qui produit une trajectoire temporelle pour chaque cible en fonction de sa trajectoire estimée. Ainsi, le générateur calcule tous les ticks réels d'une cible. Il applique également des retards et des avances aléatoires, dont l'amplitude est contrôlée par un paramètre donné.

Appelons ta_k^j la date réelle, dans le scénario, du tick k de la cible j. La date réelle ne peut pas être générée en prenant simplement le tick estimé et en ajoutant un écart aléatoire, car dans un scénario réaliste, un retard ou une avance est susceptible d'avoir un impact sur la trajectoire temporelle et entraîne encore plus de retards ou d'avances. Le générateur calcule récursivement les ticks réels avec un écart aléatoire $d \in [d_{min}, d_{max}]$, en utilisant les longueurs des fenêtres de temps mais aussi du dernier tick réel. C'est-à-dire que la date réelle de fin d'une fenêtre de temps est égale à la date de fin de la fenêtre de temps précédente, plus la durée de la fenêtre à laquelle on ajoute un retard ou soustrait une avance. Un écart $d < 0$ signifie qu'une cible est en avance et $d > 0$ signifie qu'elle est en retard.

Les dates réelles des ticks sont calculées comme suit. ta_0^j, la date d'apparition de la cible j, est égale à $t_0^j + d$, avec d dans $[d_{min}, d_{max}]$ une variable aléatoire. Ensuite, la longueur réelle d'une fenêtre de temps, notée la_k^j, est égale à max$(l_k^j + d,$ min$(0.5, l_k^j))$ avec l_k^j la longueur estimée. Notons que la longueur réelle ne peut pas être inférieure à min$(0.5, l_k^j)$ donc le scénario n'aura pas de face avec une longueur nulle ou négative. Par conséquent, les ticks réels sont calculés de telle sorte que $ta_k^j = la_k^j + ta_{k-1}^j, \forall j \in J, k \in K_j, k > 0$. Ainsi, avec un tel comportement, le scénario simulera les trajectoires temporelles réelles des cibles, avec des dates de passage croissantes (les ticks).

Les scénarios sont générés après le calcul hors ligne de l'ordonnancement initial, afin de connaître son rayon de stabilité. Lors de la génération d'un scénario, les écarts d (les avances ou retards) sont générés dans l'intervalle $[-r\Delta_{d0}, r\Delta_{d0}]$, avec Δ_{d0} la valeur initiale du rayon de stabilité dynamique de la méthode en ligne (calculé hors ligne) et r un paramètre. La valeur par défaut de r est de 1.2. Ainsi, les scénarios seront équilibrés en termes d'occurrence et d'amplitude, entre les retards et les avances, et quelques valeurs importantes au-delà du rayon de stabilité pourront être présentes.

2.6.4 Impact des retards et avances – Q 1

Dans cette première expérience, nous souhaitons analyser le comportement de la méthode en ligne sur différents scénarios, et mesurer l'efficacité des ajustements. Pour cela, nous générons 100 instances réalisables, c'est-à-dire pour lesquelles il existe un ordonnancement robuste initial, en utilisant le générateur d'instances et les paramètres par défaut. Ensuite, pour chaque instance, nous générons plusieurs scénarios et exécutons notre méthode en ligne pour résoudre cette instance avec chaque scénario. Les scénarios des instances sont tous générés de manière similaire. Par exemple, le premier type de scénario n'a ni retard ni avance, les points de passage sont tous à l'heure estimée. Le deuxième type de scénario généré a des écarts dans $[-2\Delta_{d0}, -0.5\Delta_{d0}]$, ainsi il y a seulement des avances importantes, avec Δ_{d0} la valeur initiale du rayon de stabilité dynamique de l'instance, calculé hors ligne. Les scénarios de ce type devraient tous permettre à la méthode en ligne d'assurer la couverture des cibles sur tout l'horizon de temps considéré. Le troisième type de scénario a des écarts dans $[-\Delta_{d0}, \Delta_{d0}]$, donc seulement des avances et des retards inférieurs au rayon de stabilité dynamique. Encore une fois, on s'attend à ce que la méthode en ligne trouve une solution réalisable pour tous les scénarios de ce type. Le quatrième type de scénario présente des écarts dans $[-1.5\Delta_{d0}, 1.5\Delta_{d0}]$. Il s'agit de scénarios équilibrés, avec la même probabilité d'avances et de retards, qui peut présenter des écarts au-delà du rayon de stabilité dynamique. De tels scénarios permettent de solliciter tous les cas de figures prévus par la méthode en ligne. Enfin, les deux derniers types de scénarios ne testent que des retards importants avec des écarts dans $d \in [0.90\Delta_{d0}, 1.1\Delta_{d0}]$ et $d \in [\Delta_{d0}, 2\Delta_{d0}]$. Les scénarios de ces deux types n'auront probablement pas de solution réalisable car les retards cumulés sont importants, mais cela nous permet de voir combien de temps la méthode est en mesure de fournir une couverture aux cibles dans un contexte d'accroissement fort des besoins de couverture des cibles. Si un scénario n'est plus réalisable à un moment donné (pas assez de chemins de secours pour répondre à un retard), la cible est perdue et la méthode en ligne est donc arrêtée en raison de l'échec de la mission. Les ajustements restants qui auraient dû être exécutés ne le sont donc pas. À la place de ces ajustements, nous reportons toutefois dans les tableaux de résultats un temps d'exécution de 0 seconde, avec une valeur $\Delta_d = 0$, pour comparer avec les autres scénarios.

Les temps CPU globaux pour tous les ajustements d'une instance sont notés $\sum T_{adj}$. De plus, nous rapportons également le nombre moyen d'ajustements effectués par instance. Ceci sert d'indicateur pour mesurer le temps pendant lequel la méthode en ligne a réussi à couvrir toutes les cibles sous ces scénarios, avant que la mission n'échoue. Le nombre d'ajustements maximal est égal au nombre d'ajustements des trois premiers scénarios. Enfin, nous rapportons la valeur moyenne de Δ_d en fonction du nombre d'ajustements exécutés. Par exemple, avec $[0.1, 0.2[$ dans le tableau 2.5, nous rapportons la valeur moyenne de Δ_d pour tous les instants où les cibles ont visité entre 10% et 20% du nombre total de faces à visiter. Chaque face

correspond à un ajustement, il y a alors 10% à 20% des ajustements qui sont déjà effectués. Ces valeurs sont rapportées dans les tableaux 2.5 et 2.6.

Déviation	Moyenne $\sum T_{adj}$ (s)	#Ajustements	< 0.1	$[0.1, 0.2[$	$[0.2, 0.3[$	$[0.3, 0.4[$	$[0.4, 0.5[$
$[0, 0]$	0.2523	67.97	55.75	58.27	60.81	63.74	67.70
$[-2\Delta_{d0}, -0.5\Delta_{d0}]$	0.2470	67.97	55.99	58.66	61.03	64.36	68.39
$[-1\Delta_{d0}, 1\Delta_{d0}]$	0.2468	67.97	55.60	57.53	59.70	62.05	65.34
$[-1.5\Delta_{d0}, 1.5\Delta_{d0}]$	0.2470	67.97	55.44	56.90	58.75	60.83	63.66
$[0.9\Delta_{d0}, 1.1\Delta_{d0}]$	0.2067	54.33	50.46	48.77	47.23	45.17	40.55
$[1\Delta_{d0}, 2\Delta_{d0}]$	0.0984	25.35	41.32	23.03	10.36	3.98	0.94

Tableau 2.5 – Résultats en moyenne pour différents types de scénarios avec 100 instances

Déviation	Moyenne $\sum T_{adj}$ (s)	#Ajustements	$[0.5, 0.6[$	$[0.6, 0.7[$	$[0.7, 0.8[$	$[0.8, 0.9[$	≥ 0.9
$[0, 0]$	0.2523	67.97	73.95	82.90	99.49	139.18	268.58
$[-2\Delta_{d0}, -0.5\Delta_{d0}]$	0.2470	67.97	73.66	81.97	97.04	134.84	286.14
$[-1\Delta_{d0}, 1\Delta_{d0}]$	0.2468	67.97	69.80	76.35	88.47	119.38	241.85
$[-1.5\Delta_{d0}, 1.5\Delta_{d0}]$	0.2470	67.97	67.55	73.14	83.37	109.93	218.34
$[0.9\Delta_{d0}, 1.1\Delta_{d0}]$	0.2067	54.33	33.03	26.26	15.95	6.56	1.15
$[1\Delta_{d0}, 2\Delta_{d0}]$	0.0984	25.35	0.18	0.00	0.00	0.00	0.00

Tableau 2.6 – Résultats en moyenne pour différents types de scénarios avec 100 instances

Notons que la valeur initiale moyenne de Δ_d (c'est-à-dire Δ_{d0}), calculée hors ligne, est de 54.07 secondes. Ces calculs se déroulent avant le début de la mission et ne sont donc pas impactés par nos contraintes de calcul en temps-réel. Tout d'abord, ces résultats montrent peu ou pas de différences entre le temps CPU global de l'ajustement pour les quatre premiers types de scénarios. En effet, tous ces scénarios étaient réalisables, donc les nombres d'ajustements effectués sont les mêmes (mêmes nombres de faces visitées), avec en outre des programmes linéaires à résoudre relativement proches. Cependant, nous ne pouvons pas avoir la même observation avec le temps global enregistré pour les deux autres types de scénarios, où les retards sont extrêmes. En effet, comme prévu, ces scénarios sont souvent irréalisables, avec une cible perdue avant la fin de l'horizon temporel. Par conséquent, dans de tels cas une fois constaté l'échec de la mission, la méthode en ligne est arrêtée et ne calcule plus aucun ajustement. C'est pourquoi des temps d'ajustement globaux inférieurs sont rapportés.

Deuxièmement, les résultats montrent également que dans les quatre premiers ensembles de scénarios, Δ_d est en constante augmentation. De toute évidence, l'ajustement de Δ_d permet une grande expansion du rayon de stabilité dynamique dans ces cas, avec une expansion plus rapide lorsque les cibles atteignent la fin de leurs trajectoires. En effet, pour les trois premiers ensembles, c'est peu surprenant puisqu'il n'y a pas de retard hors Δ_{d0}, donc pas de retard hors ρ. Par conséquent, avec des activités dans l'ordonnancement supprimées après quelques avances, ou avec la diminution des faces restantes, les contraintes (15) sont beaucoup moins contraignantes. Ainsi Δ_d a plus de marge pour augmenter. Étonnamment, les mêmes résultats avec une augmentation légèrement plus lente sont obtenus pour le quatrième type de scénario, où l'écart est dans $[-1.5\Delta_{d0}, 1.5\Delta_{d0}]$. Même avec plusieurs retards en dehors de Δ_{d0}, pour

lesquels la méthode en ligne avait suffisamment de chemin de secours de type 2 pour les couvrir (car aucune mission n'a échoué), la méthode est capable d'augmenter Δ_d tout en couvrant les cibles. De plus, dans certains cas, à un instant t donné, Δ_{dt} devient supérieur à $1.5\Delta_{d0}$, et plus aucun retard supérieur à Δ_d n'est désormais possible après cette date. Cependant, dans les deux dernières séries d'expériences, l'accumulation de retards importants dans toutes les faces visitées impacte la capacité de la méthode à trouver des solutions réalisables (il n'en existe peut-être pas, faute d'énergie). Dans ces quatre premiers ensembles de scénarios, plus l'ajustement est tardif (déjà un grand nombre d'ajustements exécutés), plus la valeur de Δ_d augmente. On observe en effet dans les tableaux 2.5 et 2.6 que l'augmentation de la valeur moyenne Δ_d est plus importante vers la fin. Par exemple, sa valeur moyenne est approximativement doublée lorsque l'on passe de $[80\%, 90\%[$ des ajustements déjà effectués à plus de 90%. Ce gain est nettement inférieur lorsque l'on passe de moins de 10% des ajustements déjà effectués à $[10\%, 20\%[$.

Dans l'ensemble $[0.90\Delta_{d0}, 1.1\Delta_{d0}]$, pour 99 scénarios sur 100, une cible a été perdue, mettant en échec la méthode en ligne. Cependant, cette méthode est toujours en mesure de faire en moyenne 54.33 ajustements sur les 67.97 qui devaient se faire avant de perdre une cible. Par conséquent, on en déduit que même dans ces scénarios caractérisés par des retards extrêmes, la méthode en ligne est capable de couvrir les cibles relativement longtemps. Pendant un long moment, la méthode est même capable d'obtenir des valeurs de Δ_d non nulles et ainsi des garanties sur les retards couverts. Bien entendu, ces valeurs diminuent le long de l'horizon temporel puisque les garanties offertes sont souvent dépassées. Les scénarios de l'ensemble $[1\Delta_{d0}, 2\Delta_{d0}]$ ont des retards extrêmes, et donc la valeur de Δ_d diminue très rapidement. Dans chacun de ces scénarios, la mission échoue après plusieurs retards puisqu'il n'y a plus suffisamment de chemins de secours. Cependant, on peut noter que la méthode est toujours capable de trouver des solutions, et des garanties non nulles pour tous les ajustements avant le 25ème ajustement en moyenne.

Ces résultats confirment que la méthode en ligne a le comportement désiré. En effet, pour des instances et des scénarios réalistes, elle est capable d'augmenter rapidement les garanties dynamiques de performances. Même avec quelques retards importants en dehors du rayon de stabilité dynamique, les garanties en ligne augmentent sans que la méthode perde une cible. Plus nous ajustons tard dans l'horizon temporel, plus la valeur de Δ_d augmente. Cependant, avec une répétition de retards extrêmes, la méthode finit par perdre une cible malgré la couverture et les garanties offertes à l'issue des premiers retards.

2.6.5 L'intérêt du rayon de stabilité dynamique – Q 2

Dans cette deuxième expérience, nous testons l'efficacité des garanties dynamiques de la méthode en ligne, c'est-à-dire à quel point le rayon de stabilité dynamique Δ_d est une contri-

bution intéressante par rapport au rayon de stabilité d'origine ρ. Nous savons déjà que, sans la méthode en ligne, les cibles sont perdues chaque fois qu'un retard ou une avance est au delà de ρ. Cependant, avec la méthode en ligne, nous ajoutons une autre couche de robustesse, puisque cette dernière est capable de couvrir toutes les avances et certains retards supérieurs à ρ. Par conséquent, la méthode en ligne offre déjà plus de garanties de performance que l'ordonnancement initial si la valeur de Δ_d calculé hors-ligne, Δ_{d0}, est supérieure à 0. De plus, $\Delta_{d0} > \rho$ signifie que la méthode en ligne, même en utilisant un autre ordonnancement de base non robuste ($\rho = 0$), est plus robuste que l'ordonnancement maximisant ρ. Par conséquent, pour montrer l'efficacité de la robustesse offerte par la méthode en ligne, nous rapportons dans cette expérimentation les valeurs de Δ_{d0}, par rapport à ρ. Étant donné que l'énergie disponible dans les capteurs est censée avoir un impact sur la différence entre ρ et Δ_{d0}, nous faisons varier les niveaux d'énergie minimum et maximum disponibles dans la batterie des capteurs. Ainsi, nous générons une centaine d'instances avec pour chacune, l'intervalle d'énergie disponible dans chaque capteur $b \in \{[10, 50], [25, 75], [50, 150], [100, 200], [200, 400], [400, 600], [600, 1400], [1000, 2000]\}$. Nous rapportons dans le tableau 2.7 la valeur médiane de Δ_{d0} et de ρ, avec le nombre de fois où $\Delta_{d0} > \rho$.

Énergie disponible dans les batteries	Médiane de ρ	Médiane de Δ_{d0}	#($\Delta_{d0} > \rho$)
$[10, 50]$	5.7048	0.0	0
$[25, 75]$	7.0135	2.1168	7
$[50, 150]$	7.6348	8.4548	58
$[100, 200]$	7.3338	13.3366	77
$[200, 400]$	7.7054	33.0490	91
$[400, 600]$	7.3485	63.6148	99
$[600, 1400]$	6.8821	100.3776	96
$[1000, 2000]$	7.9694	172.5926	98

Tableau 2.7 – Valeurs médiane de Δ_{d0} and ρ avec différents niveaux d'énergie dans les capteurs

Tout d'abord, comme prévu, la valeur de Δ_{d0} s'accroît avec l'augmentation de l'énergie disponible dans le réseau. Cette augmentation est vraiment importante, et avec beaucoup d'énergie disponible comme dans le dernier ensemble de cas, la robustesse dynamique offerte par la méthode en ligne est très importante. En effet, Δ_{d0} est presque égal à trois fois la durée de la mission ($\Delta_{d0} = 172$ secondes pour une mission de $H = 60$ secondes). Cependant, avec un faible niveau de batteries, Δ_{d0} est plus faible, avec déjà une part importante de l'énergie disponible absorbée par l'ordonnancement robuste initial. Dans le cas de $[10, 50]$, il n'y a pas assez d'énergie disponible dans le réseau, donc tous les ordonnancements robustes calculés sont contraints par l'énergie disponible, laissant ainsi $\Delta_{d0} = 0$. Alors que, dans les autres cas, la valeur de ρ atteint rapidement sa borne supérieure dûe au fait qu'à chaque instant il n'y a pas plus d'un capteur avec une activité de surveillance par cible. Par conséquent, dans les différents ensembles d'instances, les valeurs moyennes de ρ ne peuvent pas dépasser 8

secondes. Bien que ρ soit largement meilleur que la valeur initiale de Δ_{d0} dans les cas où le niveau initial des batteries est très contraignant, Δ_{d0} devient rapidement supérieur à ρ. En effet, dans $[10, 50]$ et $[25, 75]$, ρ est supérieur à Δ_{d0} dans 193 cas sur 200. Cependant, cette tendance s'inverse et on retrouve une nette domination de Δ_{d0} pour les intervalles $[200, 400]$ et plus. Dans ces cas, Δ_{d0} est supérieur à ρ dans 384 des 400 instances, et avec des valeurs largement plus élevées. Dans l'ensemble des instances $[50, 150]$ et $[100, 200]$, les résultats de Δ_{d0} et ρ sont plus proches, mais le premier est encore meilleur dans 135 cas sur 200, avec une valeur médiane largement meilleure en $[100, 200]$. Nous arrivons à la conclusion que dans les réseaux avec une très faible quantité d'énergie disponible, l'ordonnancement robuste initial monopolise toute l'énergie ce qui laisse peu de marge à la méthode en ligne, voire parfois aucune marge dans les cas extrêmes. Cependant, dès que l'énergie est plus abondante dans le réseau, ρ est rapidement limité par la contrainte d'un capteur activé par cible maximum à tout instant, permettant ainsi à Δ_{d0} d'augmenter considérablement avec l'énergie disponible. Par conséquent, chaque fois que les réseaux ont un niveau moyen d'énergie disponible ou plus, la robustesse offerte par la méthode en ligne est importante, et va bien au delà de ce que peut offrir l'ordonnancement robuste initial.

2.6.6 Impact du nombre de cibles sur l'effort de calcul – Q 3

Cette section traite de la question **Q 3**, en exécutant notre méthode en ligne sur des instances avec un nombre différent de cibles. Une augmentation du nombre de cibles tend à faire croître le nombre de faces visitées, donc cela a un impact sur le temps d'exécution de la méthode en ligne. En effet, la méthode effectue plus d'ajustements, qui nécessitent de résoudre plus de programmes linéaires pour les chemins de type 2, et des modèles beaucoup plus complexes pour les chemins de type 1 (plus de contraintes et de variables). La question qui guide cette expérimentation se concentre sur l'évaluation de l'impact du nombre de cibles sur l'effort de calcul et tire des conclusions sur les applications réalisables dans un contexte temps-réel. Dans ce but, nous exécutons la méthode en ligne sur des instances avec différents nombres de cibles $n \in \{1, 2, 5, 7, 10, 15\}$ et rapportons le temps nécessaire pour exécuter les ajustements en ligne. Pour chaque valeur testée, un ensemble de 100 instances avec chacune un scénario, est généré et ensuite résolu avec la méthode en ligne. Les paramètres utilisés pour la génération d'instances et de scénarios sont les paramètres par défaut présentés dans les sous-sections 2.6.2 et 2.6.3 (à l'exception de n évidemment). Nous considérons cet ensemble d'instances généré pour couvrir un large éventail de cas réalistes (1, 2, 5, 7 cibles) jusqu'au cas les plus extrêmes (10 et 15 cibles). Les résultats rapportés dans le tableau 2.8 sont : la moyenne des $\sum T_{adj}$, le temps CPU nécessaire pour tous les ajustements dans une instance, le nombre moyen d'ajustements par instance et le temps d'ajustement maximum pour toutes les instances.

#Cibles	Moyenne $\sum T_{adj}$ (s)	Moyenne #Ajustements	Maximum T_{adj} (s)
1	0.0765	32.00	0.007
2	0.2198	62.49	0.011
5	1.0861	157.03	0.026
7	2.3638	235.51	0.047
10	4.6195	325.74	0.106
15	12.4022	490.75	0.287

Tableau 2.8 – Résultats en moyenne pour différents scénarios avec différents nombres de cibles

Comme prévu, les résultats montrent qu'en augmentant le nombre de cibles dans les missions, le temps global d'ajustement par instance augmente également. En effet, dans l'ensemble de nos instances, environ 32 ajustements sont effectués par cible. L'ajout de cibles augmente donc irrémédiablement le nombre d'ajustements et donc le temps de calcul qu'elles nécessitent. De plus, comme prévu, les ajustements deviennent plus complexes, comme nous pouvons le voir avec le temps le plus long pour un ajustement (maximum T_{adj}) qui augmente toujours avec le nombre de cibles. Ainsi, nous concluons que le nombre de cibles dans une instance a un impact important sur l'effort de calcul requis par la méthode en ligne.

De plus, l'expérimentation confirme que la méthode en ligne, sur des instances avec un nombre de cibles faible ou moyen ($\{1, 2, 5, 7, 10\}$), est compatible avec le contexte temps-réel. En effet, le temps total d'ajustement et le plus long temps d'ajustement restent modestes dans ces cas, les instances s'exécutant d'un dixième de seconde à quelques secondes pour plus de cibles. Cependant, il devient plus complexe pour les cas les plus extrêmes (15 cibles). Dans de tels cas, la méthode a besoin d'environ 12.4 secondes par instance pour les ajustements. Avec des horizons de temps estimés à 60 secondes, ce n'est clairement pas compatible avec un contexte temps-réel. Bien que cela ne correspond qu'à environ 0.80 secondes de calcul par cible, la méthode pourrait être acceptable en temps réel à condition d'être appliquée sur un horizon de temps plus long. C'est-à-dire que si le nombre de cible et leur trajectoires spatiales restent les mêmes, il suffirait d'allonger l'horizon temporel de la mission (toutes les cibles ne commençant pas et ne finissant pas en même temps) pour que les besoins en puissance de calcul par unité de temps diminuent. Le besoin en temps de calcul devient important lorsque de nombreuses cibles voyagent en même temps : le temps d'ajustement maximum enregistré confirme ces observations, car dans près de 125 000 ajustements au total, l'ajustement le plus long n'a pris que 0.287 secondes.

Pour conclure, comme attendu, le nombre de cibles a un impact important sur le temps nécessaire aux ajustements. Cependant, la méthode reste acceptable pour une application dans un contexte temps-réel lorsque l'on considère des instances simples ou moyennes. Lorsqu'on atteint un nombre élevé de cibles à couvrir, dans un horizon de temps court, l'application en temps-réel de la méthode en ligne peut être remise en question, bien qu'un élargissement de

l'horizon de temporel de la mission fasse diminuer le besoin en puissance de calcul instantané.

2.6.7 Impact du nombre de capteurs sur l'effort de calcul – Q 4

Dans cette quatrième expérience, nous testons l'impact du nombre de capteurs (**Q 4**). L'objectif est de montrer les effets d'un réseau plus dense sur les performances de la méthode en ligne. En effet, avec plus de nœuds dans le réseau, la méthode a plus de possibilités pour les chemins de secours, donc ils sont plus nombreux et plus long. Par conséquent, les temps d'ajustement risquent d'augmenter avec le nombre de capteurs. Afin de vérifier cette intuition, nous exécutons et évaluons la méthode sur des instances avec différents nombres de capteurs $m \in \{50, 100, 150, 200, 300\}$ dans le réseau. Les instances sont générées de telle manière que le nombre d'ajustements reste à peu près le même. Pour cela, la densité (c'est-à-dire le nombre de capteurs par unité de surface) doit rester constante, puisqu'elle a un impact sur le nombre d'ajustements effectués par la suite. Ainsi la taille de la zone, où est déployé le réseau, varie avec le nombre de capteurs. Toutes les instances sont des surfaces de forme carrée, avec une largeur égale à $\{212, 300, 367, 424, 519\}m^2$ pour respectivement $\{50, 100, 150, 200, 300\}$ capteurs. Ensuite, les trajectoires des cibles sont dessinées dans une partie fixe de la zone, la sous-zone de taille 212×212. Un tel processus permet à toutes les instances d'avoir un nombre similaire de faces visitées, car la taille des trajectoires des cibles ainsi que la densité des capteurs sont les mêmes. On obtient ainsi un nombre similaire d'ajustements effectués dans toutes les instances. Comme pour les expérimentations précédentes, 100 instances sont générées pour chaque valeur de m testée. Les résultats sont rapportés dans le tableau 2.9, avec la moyenne et le maximum du temps total d'ajustement dans une instance, $\sum T_{adj}$. On rapporte aussi la durée de l'ajustement le plus long, T_{adj}, parmi toutes les instances d'un même m, le temps total d'ajustement maximal dans une instance (notée max$\sum T_{adj}$ dans le tableau) et le nombre d'ajustements.

#Capteurs	Moyenne $\sum T_{adj}$ (s)	#Ajustements	Maximum T_{adj} (s)	Max $\sum T_{adj}$
50	0.1043	43.91	0.008	0.401
100	0.1341	45.44	0.008	0.431
150	0.1864	48.88	0.018	0.706
200	0.1871	48.13	0.015	0.559
300	0.1943	44.79	0.020	0.507

Tableau 2.9 – Résultats en moyenne avec différents nombres de capteurs

Comme attendu, les nombres d'ajustements restent proches pour les différents ensembles d'instances et, par conséquent, la durée totale des ajustements dépend davantage du temps nécessaire pour chaque ajustement. D'une part, dans ces résultats, le temps CPU total pour les ajustements d'une instance passe d'une moyenne de 0.104 seconde à 0.194, en passant de

50 à 300 capteurs. Cela confirme qu'une augmentation du nombre de nœuds dans le réseau, même sans augmenter la densité, impacte toujours l'effort de calcul. En effet, cette augmentation a un effet sur la taille et le nombre de chemins de secours, augmentant ainsi la complexité des programmes linéaires résolus lors des ajustements (plus de variables de décision et plus de contraintes). Cependant, cela semble avoir moins d'impact sur la longueur maximale des ajustements et pour le temps d'ajustement total dans une instance, car ces deux valeurs n'augmentent pas toujours lorsque l'on ajoute des capteurs. Ainsi, nous concluons que l'augmentation du nombre de capteurs à densité égale implique une augmentation de l'effort de calcul global effectué en ligne, bien que cette augmentation soit relativement modeste.

Ces résultats montrent aussi qu'une application en temps-réel de notre méthode semble réaliste dans tous les cas testés. En effet, les instances des réseaux les plus complexes ont un temps d'ajustement total (somme des temps des ajustements) moyen de seulement 0.1943 seconde, avec 0.0194 pour le plus long ajustement, ce qui est un effort de calcul modeste compatible avec un contexte temps-réel. De plus, l'instance avec les temps totaux les plus élevés a 0.706 secondes d'ajustements. Encore une fois, même dans le pire des cas, la méthode en ligne reste réalisable dans un contexte temps-réel. Cela nous permet de conclure que, bien qu'une augmentation du nombre de capteurs induise également une augmentation de l'effort de calcul, la méthode en ligne reste compatible avec une application temps-réel.

2.6.8 Test de la méthode contrôlée par période – Q 5

Dans cette dernière expérimentation, nous étudions le comportement de la variante dite « à période contrôlée » de la méthode en ligne. Elle diffère de la méthode d'origine utilisée jusqu'à maintenant, car un ajustement est effectué lorsqu'une cible quitte une face si et seulement si le dernier ajustement a été effectué il y a au moins une seconde ou si un chemin de secours a été utilisé (car un retard dépassait le rayon de stabilité). Cette seconde version devrait avoir un temps de calcul en ligne beaucoup plus modeste, et constitue une tentative pour rendre la méthode en ligne compatible avec les exigences d'une application temps-réel, en particulier dans les cas problématiques identifiés dans les sections précédentes. Cette diminution du temps de calcul se fera au prix d'un rayon de stabilité dynamique, Δ_d, moins élevé, puisqu'il fera l'objet d'ajustements moins fréquents. Afin de comparer les deux variantes de la méthode en ligne, nous exécutons dans cette expérimentation la méthode en ligne à période contrôlée sur les mêmes instances et scénarios utilisés par la méthode par défaut dans certaines des expérimentations précédentes. Tout d'abord, pour comparer à la fois l'efficacité temporelle et l'évolution de la valeur de Δ_d, nous exécutons cette méthode sur les instances et les scénarios utilisés dans la section d'expérimentation 2.6.4, et signalons les différences dans le tableau 2.10 et le tableau 2.11. Dans ces tableaux, nous rapportons la différence du temps d'ajustement moyen entre la version par défaut et la méthode contrôlée par période. Nous rap-

portons aussi le nombre d'ajustements effectués par cette deuxième version en pourcentage du nombre d'ajustements effectués par la version par défaut et enfin nous rapportons la différence dans l'évolution de la valeur moyenne de Δ_d. Plus précisément, la colonne dont l'entête est $[0.1, 0.2[$ indique la différence moyenne entre Δ_d de la version contrôlée par période et Δ_d de la version par défaut, obtenus lorsque les cibles ont déjà traversé entre 10% et 20% des faces à traverser. Deuxièmement, nous exécutons la méthode contrôlée par période sur les mêmes instances et scénarios qu'à la sous-section 2.6.6. L'objectif est d'analyser le comportement de cette méthode lorsque le nombre de cibles varie, et plus précisément dans les cas extrêmes (15 cibles) où le temps de calcul de la méthode par défaut apparaît trop élevé. L'objectif est de déterminer si la nouvelle version de la méthode est compatible avec un contexte temps-réel. Les résultats sont rapportés dans le tableau 2.12 de la même manière que ceux rapportés dans le tableau 2.8, avec maintenant dans la deuxième colonne la somme des temps d'ajustement présentée en pourcentage du temps dans le tableau 2.8 et la troisième colonne en pourcentage du nombre d'ajustements effectués par la version par défaut.

Déviation	$\sum T_{adj}$ gagné (s)	#Ajustements	< 0.1	$[0.1, 0.2[$	$[0.2, 0.3[$	$[0.3, 0.4[$	$[0.4, 0.5[$
$[0, 0]$	-0.1492	44%	-0.1	-0.33	-0.36	-0.32	-0.44
$[-2\Delta_{d0}, -0.5\Delta_{d0}]$	-0.1975	23%	$+0.14$	-0.81	-0.79	-1.11	-1.25
$[-1\Delta_{d0}, 1\Delta_{d0}]$	-0.1006	60%	$+0.06$	-0.16	-0.26	-0.19	-0.35
$[-1.5\Delta_{d0}, 1.5\Delta_{d0}]$	-0.1016	60%	$+0.05$	-0.2	-0.27	-0.23	-0.39
$[0.9\Delta_{d0}, 1.1\Delta_{d0}]$	-0.0024	99%	-0.02	$+0.10$	$+0.31$	$+0.26$	$+0.20$
$[1\Delta_{d0}, \Delta_{d0}]$	-0.0058	96%	$+0.54$	$+0.59$	$+0.25$	$+0.14$	$+0.07$

Tableau 2.10 – Différences entre la version par défaut et la version contrôlée par période

Déviation	$\sum T_{adj}$ gagné (s)	#Ajustements	$[0.5, 0.6[$	$[0.6, 0.7[$	$[0.7, 0.8[$	$[0.8, 0.9[$	≥ 0.9
$[0, 0]$	-0.1492	44%	-0.68	-1.27	-2.17	-6.65	-25.30
$[-2\Delta_{d0}, -0.5\Delta_{d0}]$	-0.1975	23%	-1.54	-2.18	-4.73	-10.68	-58.02
$[-1\Delta_{d0}, 1\Delta_{d0}]$	-0.1006	60%	-0.46	-0.46	-2.13	-5.27	-26.0
$[-1.5\Delta_{d0}, 1.5\Delta_{d0}]$	-0.1016	60%	-0.65	-0.68	-2.07	-4.26	-24.15
$[0.9\Delta_{d0}, 1.1\Delta_{d0}]$	-0.0024	99%	-0.14	-0.08	$+0.70$	$+1.37$	$+1.92$
$[1\Delta_{d0}, \Delta_{d0}]$	-0.0058	96%	$+0.21$	$+0.23$	$+0.23$	$+0.23$	$+0.23$

Tableau 2.11 – Différences entre la version par défaut et la version contrôlée par période

#Cibles	$\sum T_{adj}$ (s)	Moyenne #Ajustements	Maximum T_{adj} (s)
1	64%	59%	0.0083
2	66%	60%	0.0138
5	60%	59%	0.0238
7	56%	57%	0.0513
10	49%	57%	0.0770
15	39%	44%	0.2034

Tableau 2.12 – Différences entre la version par défaut et la version contrôlée par période

D'une part, les tableaux 2.10 et 2.11 montrent, comme prévu, une différence dans l'évolution de la valeur de Δ_d, et dans les valeurs atteintes. En effet, dans les quatre premiers

ensembles de scénarios, les deux versions ne mettent pas à jour la valeur de Δ_d aussi fréquemment, ce qui conduit à des utilisations différentes des chemins de secours. Bien que dans les deux versions, Δ_d ait le même comportement et augmente tout au long de l'horizon temporel, il augmente plus rapidement avec la version par défaut. Ainsi, l'écart entre les valeurs moyennes de Δ_d s'élargit et la version par défaut offre toujours les meilleures garanties, avec de grandes différences à la fin de l'horizon temporel. Cependant, les différences sont faibles au début (quelques dixièmes de seconde) et la méthode contrôlée par période est même un peu meilleure avec certains scénarios pendant quelque temps. En général, pour les deux autres ensembles de scénarios, la version par défaut a des valeurs un peu meilleures au début de la mission, jusqu'à ce que la méthode contrôlée par période obtienne, étonnamment, toujours des valeurs moyennes plus élevées jusqu'à la fin de l'horizon temporel. Plus précisément, pour l'ensemble $[1\Delta_{d0}, 2\Delta_{d0}]$, nous avons remarqué que la méthode contrôlée par période obtenait toujours une solution réalisable pour une unique instance quasiment jusqu'à la fin de la mission (après 90 % des ajustements). Pour expliquer un tel phénomène, il est important de remarquer que lorsque l'on fait un ajustement, on modifie les capacités et donc on peut favoriser l'utilisation d'un chemin. Ainsi, rapidement la version par défaut utilisera des chemins différents de ceux utilisés par la version contrôlée par période. Cependant, l'utilisation d'un chemin r peut avoir un impact néfaste sur le réseau pour l'augmentation de Δ_d. Par exemple, r peut utiliser les mêmes capteurs que des chemins d'une face qui va être traversée par une cible. Donc, l'utilisation de r consomme de l'énergie dans ces capteurs communs, ce qui laisse moins de place pour une amélioration des capacités des chemins. Une telle observation ouvre des perspectives pour un raffinement de la méthode, permettant de faire des choix plus judicieux des chemins de secours, qui seront discutées à la section 2.7. Cependant, dans nos instances et nos scénarios, le meilleur gain de Δ_d par la version à période contrôlée n'est jamais supérieur à 2 secondes et le nombre de scénarios réalisables est le même. Ainsi de manière générale, la version par défaut offre des garanties supérieures par rapport à la méthode contrôlée par période. En effet, la première augmente la valeur de Δ_d plus rapidement et atteint des garanties supérieures à celles qu'offre la seconde. Cependant, dans les cas les plus extrêmes avec seulement des retards importants, les deux versions gardent des valeurs proches et font presque autant d'ajustements.

D'autre part, cette expérimentation montre également que la méthode avec période contrôlée est beaucoup plus rapide que la version par défaut. Uniquement pour les types de scénarios $[1\Delta_{d0}, 2\Delta_{d0}]$ et $[0.9\Delta_{d0}, 1.1\Delta_{d0}]$, la méthode contrôlée par période est dans des temps similaires à la version par défaut, car le nombre d'ajustements est presque égal. Cependant, la méthode contrôlée par période n'effectue pas tous les ajustements, car il y a toujours quelques retards qui restent inférieurs au rayon de stabilité initial ρ. Dans les quatre autres ensembles de scénarios, il existe de grandes différences. En effet dans ces ensembles de scénarios, entre

seulement 23% et 60% des ajustements sont effectués dans la méthode contrôlée par période contrairement à la version par défaut. De toute évidence, cela induit un impact similaire dans les temps d'ajustement globaux.

La deuxième partie de l'expérimentation, dont les résultats sont rapportés dans le tableau 2.12, confirme ces observations. En effet, la méthode contrôlée par période n'a besoin que d'un temps d'exécution compris entre 39% et 66% du temps d'exécution de la version par défaut. On observe donc qu'elle est entre 34% et 61% plus rapide. Comme pour la version par défaut, la méthode contrôlée par période voit le nombre d'ajustements effectués par instance augmenter entre 1 et 15 cibles. Cependant, le nombre d'ajustements est toujours beaucoup plus petit et cette différence augmente. De ces résultats et analyses, nous déduisons que la méthode à période contrôlée peut être compatible avec une application en temps-réel, et devient une bonne alternative à la version par défaut lorsque la complexité du problème (ici, le nombre de cibles) augmente. À noter que nous avons choisi dans notre méthode à période contrôlée des périodes d'une seconde, et que le temps de calcul peut encore plus diminuer avec des périodes de deux secondes par exemple.

Pour conclure, en contrôlant les ajustements avec une période qui est un paramètre de cette variante, nous transformons la méthode en ligne en une version à période contrôlée. Une telle version offre certainement de moins bonnes valeurs de Δ_d car moins d'ajustements sont effectués. Cependant, elle est beaucoup plus efficace en termes de temps de calcul en ligne, et peut être rendue compatible avec une application en temps-réel, même dans les cas les plus complexes (quitte à augmenter la valeur de la période pour cela). Ainsi, la méthode en ligne peut être adaptée au problème à résoudre et aux contraintes temps-réel grâce à la valeur ajustable de la période.

2.7 Conclusion du second chapitre

Dans ce chapitre, nous avons abordé la partie dynamique du problème de suivi de cibles robustes avec un réseau de capteurs sans fil. Nous proposons ici une nouvelle méthode dynamique qui utilise l'ordonnancement initial robuste proposé au chapitre 1, et l'étend pour répondre aux avances et aux retards dont la valeur est supérieure à celle du rayon de stabilité de l'ordonnancement initial robuste, dans un contexte temps-réel. La solution proposée détecte les retards et les avances, adapte l'ordonnancement et couvre temporairement les cibles à l'aide de chemins de secours préalablement calculés. Les performances offertes par la méthode en ligne sont mesurées par un rayon de stabilité dynamique qui est optimisé dynamiquement, lorsque les avances et retards réels sont connus. Une deuxième version de notre méthode a été proposée, avec période contrôlée, offrant des garanties de performances moins élevées, mais un temps de calcul plus modeste. Dans un ensemble d'expérimentations, nous avons

étudié l'efficacité de cette méthode en ligne, et avons déduit que (i) une application en temps-réel de cette méthode dans sa version par défaut est réaliste, sauf pour un nombre élevé de cibles (exemple, 15 cibles), (ii) la version de la méthode à période contrôlée offre des garanties de performance généralement plus modestes, mais se prête mieux à une application en temps-réel, (iii) les performances garanties obtenues sont potentiellement élevées et souvent meilleures que celles que confère l'ordonnancement initial robuste, (iv) dans la plupart des scénarios (sauf les scénarios à retard élevé uniquement), les performances garanties augmentent dynamiquement et atteignent des valeurs élevées de robustesse. Nous concluons de nos observations que la méthode en ligne permet de supporter de nombreux et importants retards et avances, tout en offrant des garanties de performance intéressantes. La version à période contrôlée offre un bon compromis entre les performances et le temps de calcul nécessaire, permettant ainsi d'adapter notre méthode en ligne à des problèmes plus difficiles ou à des contraintes temps-réel plus dures.

Ce travail présente une première implémention de la méthode en ligne et peut donner lieu à des extensions. Une première extension envisagée consiste à utiliser les chemins de secours plus judicieusement, comme évoqué à la section 2.6.8. Actuellement, lorsqu'une cible a un retard, la méthode en ligne utilise d'abord touts les chemins de type 1, dans un ordre aléatoire, avant d'utiliser les chemins de type 2. Cependant, tous les chemins de type 1 n'ont pas le même impact sur la capacité du réseau à répondre à de futurs retards. En effet, une stratégie simple qui pourrait être envisagée pour étendre la méthode en ligne, consisterait à utiliser d'abord les chemins de type 1 les plus isolés, c'est-à-dire, des chemins de secours qui partagent le moins de capteurs avec d'autres chemins de type 1, dans les faces restant à visiter. Une stratégie comme celle-ci éviterait l'utilisation de capteurs communs à plusieurs chemins qui sont contraignants lors de la maximisation de Δ_d. Éviter leur utilisation permettrait de donner plus d'opportunités pour maximiser Δ_d. C'est pourquoi nous souhaitons optimiser l'ordre des chemins de secours, en utilisant une méthode de programmation dynamique approximative [66] qui prendrait la meilleure décision (quel chemin utiliser ?) lorsqu'on est confronté à un retard. Ces décisions doivent prendre en compte les futurs retards et avances potentiels et leur impact sur le réseau. Une autre extension envisagée est, encore une fois, l'utilisation de CupCarbon [61, 62]. Cela peut offrir une illustration intéressante à la méthode en ligne, et permettre de tester des scénarios réalistes en donnant la possibilité de les modifier dynamiquement. On peut aussi ouvrir notre problème à une approche basée sur la théorie des jeux, où un adversaire disposant d'un budget de retard fixé, tenterait de mettre la méthode en défaut. Enfin, une dernière extension pourrait être considérée, il s'agirait d'adapter la méthode à des cibles dont la trajectoire spatiale est incertaine. Ces extensions seront abordées plus en détails dans la conclusion générale de cette thèse.

CHAPITRE 3

PLANIFICATION DE LA RECHERCHE D'UNE CIBLE AVEC DES CAPTEURS MOBILES EN CONSIDÉRANT DES COÛTS DE DÉPLACEMENT

3.1 Introduction

Le problème abordé dans ce chapitre est l'optimisation de la recherche d'une cible en utilisant un ensemble de capteurs. Ici, l'objectif est de produire une distribution optimale du budget de recherche (temps, énergie, ...) des capteurs qui minimise la probabilité de non-détection d'une cible. Ce problème est notamment abordé dans la littérature avec [40] et [41] qui ont servi de base à ce chapitre. Dans ces travaux, la zone de recherche est partitionnée en cellules. Les ressources disponibles (ou efforts de recherche) pour un ensemble de capteurs, généralement du temps, sont réparties de manière optimisée entre les cellules. On considère que l'on connaît, pour chaque cellule, la probabilité de présence de la cible. La fonction objectif résultante est la somme pondérée, avec les probabilités de présence, des probabilités de non-détection de la cible dans chaque cellule. On les appelle aussi probabilités conditionnelles de non-détection, et elles sont calculées en utilisant les ressources investies dans les cellules et leurs visibilités. La visibilité est un coefficient représentant l'efficacité de l'effort de recherche d'un capteur dans une cellule. Dans les travaux précédents [40, 41], les auteurs divisent la zone de recherche en sous-zones, qui sont des ensembles disjoints de cellules. Chaque capteur est affecté à une sous-zone et ne peut donc rechercher la cible qu'à l'intérieur des cellules de sa sous-zone. Les auteurs de ces travaux résolvent alors un problème hiérarchique à deux niveaux : trouver la meilleure allocation des capteurs aux sous-zones et, pour chaque capteur, trouver la meilleure distribution de ressources aux cellules de la sous-zone allouée. Un tel modèle permet à un capteur de naviguer librement à l'intérieur d'une même sous-zone, sans restreindre ni pénaliser ses mouvements. Dans ce chapitre, nous abordons un aspect plus réaliste et plus général de ce problème de distribution de ressources pour la recherche d'une cible, où les mouvements

des capteurs ont des coûts qui impactent la ressource disponible. En effet, dans les formulations des travaux précédents, il n'y a pas d'impact sur l'effort de recherche d'une cible lorsque celle-ci se déplace entre deux cellules d'une même sous-zone. Et cela même si les cellules sont éloignées les unes des autres (par exemple, dans des coins opposés de la sous-zone), comme si les capteurs pouvaient être téléportés entre ces cellules. L'autre paradoxe est que deux cellules adjacentes mais dans des sous-zones différentes ne peuvent pas être explorées par le même capteur. Qui plus est, ces sous-zones sont obtenues après une division arbitraire de la zone de recherche. Par exemple, avec les instances utilisées et résolues dans [40, 41], il est impossible pour un capteur de se déplacer entre deux cellules qui sont voisines sans aucune frontière physique et séparées de quelques mètres, car elles ne sont pas dans la même sous-zone. Cependant, le capteur peut librement, sans induire aucun coût, explorer successivement deux cellules beaucoup plus éloignées et séparées par une forêt, un lac ou une ville. Cette formulation ne représente pas de manière réaliste un problème de recherche où, par exemple, le déplacement du capteur entre deux cellules à explorer induit un coût temporel dépendant de la distance. La formulation du problème que nous avons choisie d'aborder se distingue des travaux de la littérature sur ce point. En effet, aucune sous-zone n'est considérée. Cependant, les capteurs consomment désormais des ressources pour passer d'une cellule à une autre. Par exemple, un capteur peut avoir besoin de consacrer 10 minutes pour passer d'un endroit à explorer à un autre. Une solution correspond maintenant, pour chaque capteur, à un chemin allant de cellule en cellule avec, dans les cellules visitées, un effort de recherche.

Dans ce chapitre, notre propre formulation du problème est étendue du cas de base avec cible statique, au cas où la cible est mobile. Dans un tel problème, les activités des capteurs sont toujours les mêmes (efforts de recherche dans les cellules avec déplacements entre elles). Cependant, la cible se déplace pendant un horizon de temps et donc sa probabilité d'être dans une cellule particulière évolue avec le temps. Ce problème est abordé en discrétisant les mouvements de la cible pendant l'horizon de temps, qui est découpé en périodes. Pendant chaque période, la probabilité de présence de la cible dans une cellule ne change pas et elle n'évolue qu'entre deux périodes. Ainsi, nous considérons que la cible ne se déplace pas pendant une période mais seulement entre deux périodes. Pour chaque période, chaque capteur dispose d'un budget de ressources pour rechercher la cible et se déplacer. Bien gérer les capteurs, compte tenu de tout l'horizon de temps, est important et rend le problème beaucoup plus complexe. Notre formulation finale est une généralisation des travaux précédents avec des sous-zones comme dans [40, 41, 48], avec des cibles statiques ou mobiles.

Les contributions de ce chapitre sont multiples. Tout d'abord, une formulation d'un nouveau problème est introduite, généralisant le problème multisous-zone multicapteur en un problème multicapteur avec des coûts de déplacement. Deuxièmement, nous proposons un algorithme pour résoudre ce problème. Contrairement aux travaux antérieurs de la littérature, il n'est pas

basé sur une méthode progressif-rétrogressif [45]. Notre méthode utilise une approximation de la fonction objectif pour calculer une utilité pour chaque variable de décision et construire, étape par étape, une solution. L'efficacité de ce nouvel algorithme est testée et comparée à une borne inférieure, également présentée ci-après.

Ce chapitre est organisé comme suit. À la section 3.2, on présente la définition formelle de notre problème. La section 3.3 introduit l'approximation linéaire utilisée dans notre algorithme. L'algorithme est détaillé à la section 3.5. Des tests numériques sont présentés à la section 3.6 pour étudier l'efficacité d'une telle méthode. Enfin, nous concluons ce chapitre dans la dernière section.

3.2 Définition du problème

Dans cette section, nous donnons la définition formelle du problème à l'aide de multiples sous-sections. Nous présentons la version cible mobile de notre problème. Le tableau 3.1 résume toutes les notations qui seront introduites dans la section et qui sont propres à ce chapitre. Dans la dernière sous-section, nous montrons comment les travaux précédents (multisous-zone [40, 41]) peuvent être généralisés à l'aide de notre formulation.

3.2.1 La zone de recherche et sa discrétisation

L'espace de recherche est nommé E. C'est une grande zone où les caractéristiques de recherche sont hétérogènes. Par exemple, certaines parties de la zone correspondent à des terrains escarpés plus difficiles à explorer que d'autres, ou un lac où la probabilité de présence de la cible est nulle. La discrétisation de la zone est sa division en un ensemble de cellules disjointes C. Les points de l'espace dans une même cellule sont homogènes. Ainsi donc, on considère que tous les paramètres de recherche sont constants dans une même cellule. La partition est faite de telle sorte que :

$$E = \bigcup_{c \in C} c \quad et \quad c \cap c' = \emptyset \ \ \forall c, c' \in C$$

Chaque paire de cellules $(c, c') \in C^2$ a un coût de déplacement, noté $t_{cc'}$. Ces coûts sont constants.

3.2.2 La cible

Une cible se déplace pendant un horizon temporel $[0, H]$. Cet horizon est divisé en un ensemble $T = \{1, 2, \ldots, n\}$ de n périodes. Dans chaque période t, la cible est cachée dans une cellule et nous considérons qu'elle reste dans cette cellule pendant toute la période. Lors

Constantes	
E	Zone de recherche
C	Ensemble de cellules
$t_{cc'}$	Distance entre c et c'
H	Horizon de temps
$T = \{1, \ldots, n\}$	Ensemble de périodes
Ω	Ensemble de trajectoires
$\alpha(\vec{\omega})$	Probabilité de $\vec{\omega}$
$I = \{1, \ldots, m\}$	Ensemble de capteurs
Φ_i^t	Ressources disponibles à la période t pour le capteur i
w_c^i	Visibilité du capteur i dans la cellule c
$P_c^i(x)$	Probabilité conditionnelle de non-détection pour le capteur i dans la cellule c en fonction de la ressource x allouée
Indices	
i	Indice de capteur
c	Indice de cellule
t	Indice de période
d	Dépôt
$\vec{\omega}$	Indice de trajectoire
Variables de décision	
R_i^t	Route du capteur i pendant la période t
c_p^{ti}	p-ième cellule visitée pendant la période t par le capteur i
φ_{ic}^t	Ressources allouées à la cellule c pendant la période t par le capteur i
$y_{cc'}^{ti}$	Variable égale à 1 si le capteur i, pendant la période t, se déplace de la cellule c à la cellule c'
h_{ic}^t	Variable égale à 1 si le capteur i explore la cellule c pendant la période t,
g_{ti}	Variable égale à 1 si le capteur i explore au moins une cellule pendant la période t
u_{ic}^t	Variable technique utilisée pour éviter les sous-tours dans la trajectoire du capteur i pendant la période t
q_{ti}	Ressources utilisées par le capteur i pour un déplacement qui commence pendant la période t mais finit dans une autre période
r_{ti}	Ressources utilisées par le capteur i pour un déplacement qui a commencé dans une autre période mais finit pendant la période t

Tableau 3.1 – Résumé des notations

d'un changement de période, la cible peut se déplacer vers une autre cellule ou rester dans la même. Pour représenter cela, un ensemble de trajectoires Ω, avec leurs probabilités, est donné. Une trajectoire, $\vec{\omega} \in \Omega$, est un vecteur de n cellules (une cellule par période) avec une probabilité $\alpha(\vec{\omega})$ que la cible la suive. La condition suivante est respectée :

$$\sum_{\vec{\omega} \in \Omega} \alpha(\vec{\omega}) = 1$$

$\vec{\omega}(t)$ est défini comme la position (une cellule) de la cible pendant la période t si celle-ci suit la trajectoire $\vec{\omega}$.

3.2.3 La recherche des capteurs

Pour la mission de recherche, on dispose d'un ensemble I de m capteurs mobiles. Cet ensemble est hétérogène et les capteurs peuvent ne pas avoir les mêmes caractéristiques. Un capteur $i \in I$ a pour chaque période t un montant Φ_i^t de ressources disponibles, généralement du temps. De plus, le capteur a également un coefficient de visibilité dans chaque cellule c, noté w_c^i. Il s'agit d'un coefficient représentant l'efficacité (ou récompense) de l'effort de recherche dans la cellule considérée. En règle générale, une haute visibilité signifie que le capteur explore efficacement la cellule. Et donc, la probabilité de manquer la cible, si elle se trouve dans cette cellule, décroît rapidement lorsque l'on augmente l'effort de recherche du capteur à l'intérieur (par exemple, le temps passé à explorer la cellule). C'est le cas, par exemple, d'une cellule où le capteur peut facilement se mouvoir et où il y a peu de cachettes pour la cible. À l'inverse, dans une cellule à faible visibilité, un effort de recherche important consenti par le capteur peut conduire à une probabilité non négligeable que la cible y soit présente, mais qu'elle ait échappé au capteur. C'est le cas pour les lieux difficiles à explorer.

Les activités de recherche des capteurs sont représentées par φ_{ic}^t, la ressource allouée par le capteur i à l'exploration de la cellule c pendant la période t, et R_i^t la route traversant toutes les cellules explorées par i pendant t. Pour t et i donnés, $R_i^t = \{c_1^{ti}, \ldots, c_{|R_i^t|}^{ti}\}$ représente une route où i se déplacera de la cellule c_p^{ti} vers la cellule c_{p+1}^{ti} pendant la période $t, \forall p \in \{1, \ldots, |R_i^t| - 1\}$. Il est également important de souligner que la route d'un capteur est continue sur tout l'horizon temporel considéré. Il n'y a pas d'interruption de la recherche entre deux périodes. Par conséquent, un capteur doit se déplacer de la dernière cellule explorée pendant une période, à la prochaine cellule visitée dans une autre période. Ce déplacement a un coût qui sera réparti sur les périodes concernées par le déplacement. Ainsi, un capteur i voyagera continûment de $c_{|R_i^t|}^{ti}$ à $c_1^{t_2 i}$, $\forall t \in T, t < |T|$ et $t < t_2 \leq |T|$ et tel qu'il n'y a pas de cellule explorée pour tout $t_3 \in]t, t_2[$. Ces déplacements, entre la dernière cellule visitée d'une période, jusqu'à la première cellule visitée dans une autre période, sont appelés déplacements inter-périodes. Les autres déplacements, qui ont lieu intégralement dans une même période, sont

dits intra-périodes.

3.2.4 La fonction objectif

Une cible est dite non détectée si elle n'a été trouvée à aucun moment par aucun capteur sur l'horizon de temps lors de mission de recherche. L'objectif est de minimiser la probabilité d'un tel événement. Nous utilisons la probabilité de non-détection conditionnelle habituellement utilisée dans la théorie de la recherche [4, 41], qui est une fonction non croissante et convexe. Cette fonction représente l'efficacité de la recherche au niveau de la cellule et est notée $P_c^i(\varphi_{ic}^t), \forall i \in I, c \in C, t \in T$ tels que :

$$P_c^i(\varphi_{ic}^t) = e^{-w_c^i \varphi_{ic}^t}$$

Ce modèle implique que les recherches des capteurs sont indépendantes, tant au niveau spatial (c'est-à-dire au niveau des cellules explorées) que temporel (quelles que soient les périodes d'exploration). Autrement dit, $\forall i \in I, \forall t \in T, \forall c \in c$, le calcul de $P_c^i(\varphi_{ic}^t)$ ne dépend d'aucun $P_c^{i'}(\varphi_{i'c'}^{t'}), \forall i' \in I, \forall t' \in T, \forall c' \in c | (i,t,c) \neq (i',t',c')$. Cela signifie que la probabilité qu'un capteur i ne trouve pas la cible, en recherchant la cellule c où elle se trouve pendant la période t, dépend uniquement de son effort de recherche, et pas du tout des éventuelles recherches des autres capteurs dans c. La fonction objectif, $f(X), X = \{\varphi_{ic}^t\}_{\forall t \in T, i \in I, c \in C}$, est alors la suivante :

$$\text{Minimiser } f(X) = \sum_{\vec{\omega} \in \Omega} \alpha(\vec{\omega}) \prod_{i \in I} \prod_{t \in T} P_c^i(\varphi_{i\vec{\omega}(t)}^t) \quad (1)$$

C'est une fonction objectif non linéaire.

3.2.5 Le modèle mathématique

Le modèle mathématique que nous utilisons pour représenter le problème est présenté dans cette section. Nous avons décidé de produire un modèle aussi linéaire que possible. Ainsi, toutes les contraintes sont linéaires, seule la fonction objectif ne l'est pas. De nombreuses variables de décision sont nécessaires pour représenter les routes, les allocations de ressources et tous les cas particuliers envisageables (par exemple, un capteur qui ne se déplace pas pendant une période ou qui au contraire consacre toute la durée de la période à se déplacer). Pour chaque capteur et chaque période, la route R_i^t est représentée en utilisant des éléments de modélisation inspirés du problème classique du voyageur de commerce [67] en ajoutant un nœud virtuel, le dépôt d, avec des coûts nuls pour s'y déplacer ou le quitter. On peut remarquer que le modèle proposé peut facilement être adapté au problème où les capteurs doivent retourner (à la fin de la mission ou à chaque fin de période) dans un dépôt. La complexité de notre problème et de sa formulation est élevée puisqu'il est nécessaire d'une part de décider de la

distribution des ressources disponibles pour chaque capteur, et d'autre part de déterminer une route passant par toutes les cellules que le capteur s'est vu attribuer.

Les différentes variables de décision sont les suivantes :

— φ_{ic}^{t} les ressources allouées à la cellule c par le capteur i pendant la période t comme présenté plus tôt. Il s'agit d'une variable continue.

— $y_{cc'}^{ti}$, une variable binaire, égale à 1 si le capteur i se déplace de c à c' pendant la période t, 0 sinon.

— y_{cd}^{ti} (respectivement y_{dc}^{ti}), une variable binaire, égale à 1 si la cellule c est la dernière cellule (respectivement, la première) explorée par le capteur i pendant la période t, 0 sinon.

— h_{ic}^{t}, une variable binaire, égale à 1 si la cellule c est explorée par le capteur i pendant la période t, 0 sinon.

— g_{ti}, une variable binaire, égale à 1 si le capteur i explore au moins une cellule pendant la période t, 0 sinon.

— q_{ti}, une variable continue, la quantité de ressources utilisées pendant la période t par le capteur i pour le déplacement qui commence en t mais finit dans une autre période postérieure à t.

— r_{ti}, une variable continue, la quantité de ressources utilisées pendant la période t par le capteur i pour un déplacement qui a commencé dans une période antérieure à t et qui finit dans la période t ou une période postérieure.

— u_{ic}^{t}, une variable entière, une variable technique utilisée pour les contraintes d'évitement des sous-tours comme présentées dans [68].

Le modèle du problème de recherche d'une cible mobile est présenté dans le modèle 3.1.

La signification des contraintes est développée ci-dessous :

— (1) est la fonction objectif comme présentée précédemment

— (2) limitent les ressources disponibles à chaque période pour chaque capteur

— (3) sont des contraintes partageant le coût d'un voyage inter-périodes entre les périodes concernées. Lors du déplacement d'une cellule c vers une cellule c' commençant à la période t et se terminant à la période t', le coût ($t_{cc'}$) est partagé entre t, t' et toutes les périodes dites *vides* qui les séparent (aucune cellule n'est explorée par le capteur au cours de ces périodes de déplacement). La contrainte n'est pas toujours active (c n'est pas la dernière cellule visitée, il y a des périodes non vides entre les deux, ...). Ainsi, une partie importante de ces contraintes est simplement utilisée pour les désactiver si les conditions ne sont pas remplies. Par exemple, la partie droite des contraintes est juste égale à $t_{cc'}$ lorsqu'elle est active. Si elle est inactive alors la partie de droite est une valeur négative ou nulle et ne restreint donc pas les valeurs des variables de décision

— (4) fixe g_{ti} à 1 s'il y a au moins une cellule explorée pendant la période t par le capteur i

$$\text{Minimiser } f(X) = \sum_{\vec{\omega}\in\Omega} \alpha(\vec{\omega}) \prod_{t\in T} \prod_{i\in I} P_c^i(\varphi^t_{i\vec{\omega}(t)}) \quad (1)$$

$$\sum_{c\in C} \varphi^t_{ic} + \sum_{c\in C} \sum_{c'\in C} y^{ti}_{cc'} t_{cc'} + q_{ti} + r_{ti} \leq \Phi^t_i \qquad \forall i \in I, \forall t \in T \quad (2)$$

$$q_{ti} + \sum_{t''=t+1}^{t''\leq t'} r_{t''i} \geq t_{cc'}(1 - 2(1 - (\tfrac{y^{ti}_{cd} + y^{t'i}_{dc'}}{2})) - \sum_{t''=t+1}^{t''<t'} g_{t''i}) \qquad \forall (c,c') \in C^2, c \neq c', \forall i \in I, \forall (t,t') \in T^2, t' \geq t+1 \quad (3)$$

$$\sum_{c\in C} h^t_{ic} \leq |C| g_{ti} \qquad \forall i \in I, \forall t \in T \quad (4)$$

$$\varphi^t_{ic} \leq \Phi^t_i h^t_{ic} \qquad \forall c \in C, \forall t \in T, \forall i \in I \quad (5)$$

$$\sum_{c\in C} y^{ti}_{cd} = g_{ti} \qquad \forall i \in I, \forall t \in T \quad (6)$$

$$\sum_{c\in C} y^{ti}_{dc} = g_{ti} \qquad \forall i \in I, \forall t \in T \quad (7)$$

$$\sum_{c'\in C\cup\{d\}} y^{ti}_{cc'} = h^t_{ic} \qquad \forall c \in C, \forall i \in I, \forall t \in T \quad (8)$$

$$\sum_{c'\in C\cup\{d\}} y^{ti}_{c'c} = h^t_{ic} \qquad \forall c \in C, \forall i \in I, \forall t \in T \quad (9)$$

$$u^t_{ic} - u^t_{ic'} + y^{ti}_{cc'}|C| \leq |C| - 1 \qquad \forall (c,c') \in C^2, c \neq c', \forall i \in I, \forall t \in T \quad (10)$$

$$\varphi^t_{ic} \in \mathbb{R}^+ \qquad \forall c \in C, \forall i \in I, \forall t \in T \quad (11)$$

$$h^t_{ic} \in \{0,1\} \qquad \forall c \in C, \forall i \in I, \forall t \in T \quad (12)$$

$$y^{ti}_{cc'} \in \{0,1\} \qquad \forall (c,c') \in C^2, \forall i \in I, \forall t \in T \quad (12)$$

$$0 \leq u^t_{ic} \leq |C| \qquad \forall c \in C, \forall i \in I, \forall t \in T \quad (13)$$

$$q_{ti} \in \mathbb{R}^+ \qquad \forall i \in I, \forall t \in T | t < |T| \quad (14)$$

$$r_{ti} \in \mathbb{R}^+ \qquad \forall i \in I, \forall t \in T | t > 1 \quad (15)$$

$$g_{ti} \in \{0,1\} \qquad \forall i \in I, \forall t \in T \quad (16)$$

Modèle 3.1 – Formulation mathématique du problème du chapitre 3

— (5) fixe h^t_{ic} à 1 si le capteur i recherche la cible dans la cellule c pendant la période de temps t

— (6) & (7) contraignent les variables de décision de telle sorte qu'une seule cellule soit la dernière cellule explorée par un capteur i pendant la période t, et qu'une seule cellule soit la première cellule explorée

— (8) & (9) sont des contraintes de flots relatives aux déplacements des capteurs. Plus précisément, si la cellule est explorée par un capteur pendant une période de temps, le capteur n'arrive et ne repart qu'une seule fois de cette cellule. Sinon le capteur ne passe pas par cette cellule

— (10) sont les contraintes classiques d'élimination des sous-tours telles qu'elles sont introduites dans [68]

— Toutes les autres contraintes sont des contraintes sur les domaines des variables

3.2.6 Le problème avec sous-zones – un cas particulier

Dans cette section, nous montrons comment le problème abordé dans les travaux précédents avec la zone de recherche divisée en multiples sous-zones [40, 41, 48] peut être vu comme un cas particulier de notre problème. Pour cela, nous montrons que toutes les instances avec sous-zones peuvent être transformées en instances de notre problème.

Comme indiqué précédemment, le problème multisous-zone est tel que chaque capteur est alloué à une sous-zone pré-définie. Une fois alloué, un capteur ne peut rechercher la cible que dans les cellules appartenant à sa sous-zone, avec cependant une liberté totale de mouvement entre ces cellules. Dans le cas de la cible mobile, un capteur est alloué à une sous-zone pour chaque période. Les autres aspects du problème (fonction objectif, visibilités, ...) sont identiques au problème présenté dans ce chapitre (voir [40, 41, 48]). Les coûts de déplacement sont importants pour représenter les allocations aux sous-zones dans notre formulation. Les coûts entre les cellules de différentes sous-zones doivent être infinis et être donc prohibitifs pour chacun de ces mouvements. De plus, les déplacements sont libres entre deux cellules de la même sous-zone, ce qui se traduit dans notre modèle par des coûts nuls. Avec $z(c), \forall c \in C$ renvoyant la sous-zone associée à une cellule c, on obtient mathématiquement :

$$\forall (c, c') \in C^2, t_{cc'} = \begin{cases} +\infty & \text{si } z(c) \neq z(c') \\ 0 & \text{sinon.} \end{cases}$$

En pratique, dans le premier cas ($z(c) \neq z(c')$), il suffit d'avoir des coûts de déplacement supérieurs aux ressources disponibles des capteurs. Avec de tels coûts, il est alors impossible pour un capteur d'effectuer un déplacement entre deux cellules de sous-zones différentes. Un tel déplacement nécessitant un coût supérieur à ses ressources disponibles, il ne pourra être retenu que dans des solutions non réalisables.

Cependant, dans les travaux avec sous-zones multiples, les capteurs peuvent être réaffectés à une sous-zone différente lors du changement de période. Il y a donc un déplacement possible entre deux cellules de sous-zones différentes si et seulement si ce déplacement a lieu pendant un changement de période. Les coûts de déplacement étant invariants dans notre formulation, nous représentons cette réaffectation différemment. Nous divisons chaque capteur d'une instances en n sous-capteurs, où n est le nombre de périodes. Chacun de ces sous-capteurs est affecté à une période et n'a une quantité de ressources non nulle que pendant cette période. C'est-à-dire, qu'un sous-capteur n'est actif que pendant une période. Par conséquent, il n'y a pas de déplacement inter-périodes. Chaque sous-capteur représente la recherche du capteur original pendant une seule période déterminée. Les coûts entre deux cellules de sous-zones différentes étant toujours infinis, un sous-capteur est également alloué à une unique sous-zone au cours de sa période. Ainsi, la recherche d'un capteur est représentée dans chaque période par un sous-capteur, qui est alloué à une sous-zone.

Ces coûts et cette division en sous-capteurs transforment notre problème en une généralisation des travaux antérieurs réalisés sur le problème avec multisous-zone, qu'on retrouve dans [40, 41, 48].

3.3 Approximation linéaire

Dans cette section, nous présentons l'approximation linéaire de la fonction objectif qui sera utilisée dans l'algorithme de résolution ainsi que pour les bornes inférieures (voir section 3.4 et section 3.5). L'objectif est de se débarrasser de l'exponentielle dans la fonction objectif et d'obtenir une fonction linéaire qui rendra le modèle plus facile à résoudre. Plus l'approximation linéaire sera proche de la fonction objectif originelle, meilleures seront les solutions produites par l'algorithme exploitant l'approximation. L'idée initiale est d'utiliser l'approximation $e^x \approx 1+x$ de la fonction exponentielle. La qualité d'une telle approximation est bonne lorsque x est proche de 0, mais se dégrade rapidement lorsque x s'en éloigne. Une solution à ce problème consiste à affiner l'approximation en la recalculant pour plusieurs points donnés. Par exemple, $e^x \approx exp(y)(1+(x-y))$ est une meilleure approximation lorsque la valeur de x est proche de la constante y.

Nous exprimons toutes les variables de décision $\varphi^t_{ic}, \forall i \in I, \forall c \in C, \forall t \in T$ comme la somme de deux termes : φ^{at}_{ic} et φ^{bt}_{ic}. Le premier correspond aux ressources déjà allouées à la cellule c par le capteur i à la période t. C'est une décision déjà prise, donc une constante. Le deuxième terme est une décision qui reste à prendre, une variable de décision. Ce second terme indique combien de ressources supplémentaires nous ajoutons dans la cellule c avec le capteur i pendant la période t. La nouvelle fonction objectif, pour une solution courante avec un ensemble donné d'allocations déjà faites $\vec{X} = \{\varphi^{at}_{ic}\}_{i\in I, t\in T, c\in C}$, est :

$$\text{Minimiser } \sum_{\vec{\omega}\in\Omega}\alpha(\vec{\omega})\prod_{i\in I}\prod_{t\in T}\left(e^{-w^i_{\vec{\omega}(t)}\varphi^{at}_{i\vec{\omega}(t)}} \times e^{-w^i_{\vec{\omega}(t)}\varphi^{bt}_{i\vec{\omega}(t)}}\right)$$

ou :

$$\text{Minimiser } \sum_{\vec{\omega}\in\Omega}\alpha(\vec{\omega})\left(e^{\sum_{i\in I}\sum_{t\in T}-w^i_{\vec{\omega}(t)}\varphi^{at}_{i\vec{\omega}(t)}} \times e^{\sum_{i\in I}\sum_{t\in T}-w^i_{\vec{\omega}(t)}\varphi^{bt}_{i\vec{\omega}(t)}}\right)$$

Si l'on applique l'approximation, on obtient :

$$\text{Minimiser } \sum_{\vec{\omega}\in\Omega}\alpha(\vec{\omega})e^{\sum_{i\in I}\sum_{t\in T}-w^i_{\vec{\omega}(t)}\varphi^{at}_{i\vec{\omega}(t)}}(1-\sum_{i\in I}\sum_{t\in T}w^i_{\vec{\omega}(t)}\varphi^{bt}_{i\vec{\omega}(t)})$$

et donc :

$$\text{Minimiser } \mathrm{K} - \sum_{\vec{\omega}\in\Omega}\sum_{i\in I}\sum_{t\in T}\alpha(\vec{\omega})w^i_{\vec{\omega}(t)}e^{\sum_{i'\in I}\sum_{t'\in T}-w^{i'}_{\vec{\omega}(t')}\varphi^{at'}_{i'\vec{\omega}(t')}}\varphi^{bt}_{i\vec{\omega}(t)}) \quad \text{(AP)}$$

Où K est une constante égale à $\sum_{\vec{\omega}\in\Omega} \alpha(\vec{\omega}) e^{\sum_{i\in I}\sum_{t\in T} -w^i_{\vec{\omega}(t)}\varphi^{at}_{i\vec{\omega}(t)}}$.

Bien sûr, plus les variables de décision φ^{bt}_{ic} sont grandes, plus l'approximation est éloignée de la fonction objectif.

3.4 Borne inférieure

Nous introduisons maintenant une méthode pour calculer des bornes inférieures pour le problème abordé dans ce chapitre. L'idée principale est de résoudre une version relaxée du problème, et d'utiliser l'approximation (AP). La fonction objectif du problème relaxé est une approximation linéaire par morceaux avec une erreur bornée (la distance entre l'approximation et la valeur réelle est contrôlée). On utilise plusieurs fois l'approximation (AP), appliquée en différents points $\vec{X}$. La complexité du problème étudié est due principalement au calcul des routes des capteurs. Ainsi, nous proposons de résoudre une version relaxée du problème, sans calcul de route entre les cellules mais seulement le calcul d'une borne inférieure sur le coût de déplacement. L'idée est de considérer qu'un capteur explorant x cellules pendant une période de temps, devra accomplir $x-1$ déplacements intra-périodes pour les visiter toutes. Chaque déplacement a un coût supérieur ou égal au coût minimum de déplacement entre deux cellules, que l'on note $T_{min} = \min_{c,c'\in C|c\neq c'} t_{cc'}$. Le problème relaxé ainsi obtenu est résolu à l'aide du solveur commercial Gurobi [69].

3.4.1 Modèle du problème relaxé

Le problème relaxé est obtenu en supprimant les contraintes (4), (6)-(10), et en modifiant les contraintes (2) pour obtenir les contraintes (2'). La fonction objectif du problème relaxé est présentée plus en détail par la suite et appelée $g(X)$. Cette fonction objectif, pour une allocation de ressources donnée, est toujours inférieure à $f(X)$.

Dans le modèle du problème relaxé, nous ajoutons un nouveau type de variable $k_{ti}, \forall t \in T, \forall i \in I$. Il s'agit du nombre de déplacements intra-périodes nécessaires au capteur i pour visiter toutes les cellules explorées pendant la période t. La variable k_{ti} est égale au nombre de cellules visitées moins 1 s'il y a au moins une cellule visitée, 0 sinon. On obtient alors, $k_{ti} = max\left(0, (\sum_{c\in C} h^t_{ic}) - 1\right), \forall t \in T, \forall i \in I$. Dans le modèle, l'expression de k_{ti} est linéarisée à l'aide des contraintes (17) et (18). Le modèle du problème relaxé est alors celui présenté avec le modèle 3.2.

Lemme : Le modèle 3.2 est une relaxation du problème abordé dans ce chapitre dont la formulation mathématique est le modèle 3.1.

Minimiser $g(X)$		(1')
$\sum_{c\in C} \varphi_{ic}^t + k_{ti} t_{min} \leq \Phi_i^t$	$\forall i \in I, \forall t \in T$	(2')
$\varphi_{ic}^t \leq \Phi_i^t h_{ic}^t$	$\forall c \in C, \forall t \in T, \forall i \in I$	(5)
$k_{ti} \geq (\sum_{c\in C} h_{ic}^t) - 1$	$\forall t \in T, \forall i \in I$	(17)
$\varphi_{ic}^t \in \mathbb{R}^+$	$\forall c \in C, \forall i \in I, \forall t \in T$	(11)
$h_{ic}^t \in \{0,1\}$	$\forall c \in C, \forall i \in I, \forall t \in T$	(12)
$g_{ti} \in \{0,1\}$	$\forall i \in I, \forall t \in T$	(16)
$k_{ti} \in \mathbb{R}^+$	$\forall i \in I, \forall t \in T$	(18)

Modèle 3.2 – Formulation mathématique du problème relaxé

Preuve : Les contraintes du problème relaxé proviennent du problème initial, à l'exception de (2'), (17) et (18). (17) et (18) sont deux types de contraintes utilisées pour fixer les variables nouvellement introduites k_{ti} et ne contraignent pas plus la solution. (2') est une modification de (2). Ainsi, pour prouver que ce modèle est bien une relaxation du problème d'origine, nous devons prouver que les contraintes (2) sont plus restrictives que (2'). C'est vrai si les inégalités suivantes sont satisfaites :

$$\sum_{c\in C} \varphi_{ic}^t + k_{ti} t_{min} \leq \sum_{c\in C} \varphi_{ic}^t + \sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti} t_{cc'} + q_{ti} + r_{ti}, \forall i \in I, t \in T$$

Il est clair que $t_{min} \leq t_{cc'}, \forall (c,c') \in C^2$. Il reste à montrer que :

$$k_{ti} \leq \sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti}, \forall i \in I, \forall t \in T$$

Considérant maintenant que :

$$k_{ti} = max\left(0, (\sum_{c\in C} h_{ic}^t) - 1\right), \forall i \in I, \forall t \in T$$

Cela conduit à distinguer deux cas pour $i \in I$ et $t \in T$ donnés. Soit $k_{ti} = 0$, ce qui implique que l'inégalité est satisfaite. Soit, $k_{ti} > 0$, auquel cas il faut démontrer que :

$$(\sum_{c\in C} h_{ic}^t) - 1 \leq \sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti}$$

En utilisant les contraintes (4) et (6), nous pouvons déduire que, dans ce cas, $\sum_{c\in C} y_{cd}^{ti} = 1$. Ainsi, nous avons :

$$(\sum_{c\in C} h_{ic}^t) - 1 \leq (\sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti}) + (-1 + \sum_{c\in C} y_{cd}^{ti})$$

$$(\sum_{c\in C} h^t_{ic}) - 1 \leq (\sum_{c\in C}\sum_{c'\in C\cup\{d\}} y^{ti}_{cc'}) - 1$$

Cependant, avec les contraintes (8), nous savons que $\sum_{c'\in C\cup\{d\}} y^{ti}_{cc'} = h^t_{ic}$. Par conséquent, l'inégalité devient :

$$(\sum_{c\in C}\sum_{c'\in C\cup\{d\}} y^{ti}_{cc'}) - 1 \leq (\sum_{c\in C}\sum_{c'\in C\cup\{d\}} y^{ti}_{cc'}) - 1$$

On a ainsi montré que le modèle proposé est bien une relaxation du problème d'origine. □

3.4.2 La fonction objectif linéaire par morceaux avec erreur bornée

La fonction objectif utilisée dans le modèle du problème relaxé, $g(X)$, est un minorant de $f(X)$ linéaire par morceaux. Son erreur est bornée par ε, où ε est un paramètre strictement positif donné et représente un pourcentage d'écart avec les valeurs réelles. Ainsi, la distance maximale entre la valeur de $g(X)$ et $f(X)$ est $\varepsilon f(X)$, pour toute solution réalisable X. C'est-à-dire que $g(X) \geq (1-\varepsilon)f(X)$. Nous considérons $\varepsilon \in [0, 1[$. Plus ε est faible, et plus la valeur optimale du modèle relaxé sera proche de la valeur optimale du problème d'origine. Cependant, une valeur faible de ε augmente également la complexité (plus de morceaux dans la fonction linéaire par morceaux). Par conséquent, un compromis entre la qualité de la solution et la complexité de la résolution doit être trouvé. La fonction objectif linéaire par morceaux utilisée dans le problème relaxé utilise l'approximation (AP) (présentée dans la section 3.3) appliquée en différents points. Il est à noter que cette fonction d'approximation est d'abord un minorant, elle est donc toujours inférieure à la fonction objectif du problème original.

Pour chaque trajectoire $\vec{\omega} \in \Omega$, considérons la fonction $A_{\vec{\omega}}(x) = \alpha(\vec{\omega})e^{-x}$ sur l'intervalle $[0, L_{\vec{\omega}}]$, où $L_{\vec{\omega}} = \sum_{i\in I}\sum_{t\in T} \Phi^t_i w^i_{\vec{\omega}(t)}$. La fonction objectif initiale peut s'écrire $f(X) = \sum_{\vec{\omega}\in\Omega} A_{\vec{\omega}}(\sum_{x_{\vec{\omega}}\in X} x_{\vec{\omega}})$. Si, pour tout $\vec{\omega}$ dans Ω, nous trouvons un minorant linéaire de $A_{\vec{\omega}}$ supérieur à $(1-\varepsilon)A_{\vec{\omega}}(x), \forall x \in [0, L_{\vec{\omega}}]$ alors nous obtenons un minorant de $f(X)$ qui l'approxime avec une erreur bornée par ε en sommant ces approximations. En effet, pour tout $\vec{\omega} \in \Omega$, la différence entre la valeur du minorant et de $A_{\vec{\omega}}$ est inférieure à $\varepsilon A_{\vec{\omega}}$ et donc la différence entre $f(X)$ et la somme de ces minorants est bornée par $\varepsilon f(X)$.

Dans le reste de cette section, nous construisons un minorant de $A_{\vec{\omega}}$ pour tout $\vec{\omega} \in \Omega$. Nous cherchons plus spécifiquement à obtenir un minorant linéaire par morceaux de $A_{\vec{\omega}}$, noté $A'_{\vec{\omega}}$, qui en fournit aussi une bonne approximation. Pour s'assurer que l'erreur d'approximation est bornée par ε, nous divisons l'intervalle $[0, L_{\vec{\omega}}]$ en sous-intervalles, où $A'_{\vec{\omega}}$ est linéaire dans chaque sous-intervalle. La droite de chaque sous-intervalle joint deux points de $(1-\varepsilon)A_{\vec{\omega}}$ et est tangente à $A_{\vec{\omega}}$. Seul le dernier sous-intervalle peut ne pas consister en une droite tangente à $A_{\vec{\omega}}$. Tous les sous-intervalles (sauf peut-être le dernier) ont la même largeur.

Connaissant x_0, la borne inférieure du sous-intervalle courant, la première étape consiste à

trouver l'abscisse $x_1 \geq x_0$ où la droite est tangente à $A_{\vec{\omega}}$. Si x_1 s'avère supérieur à $L_{\vec{\omega}}$, alors le sous-intervalle est $[x_0, L_{\vec{\omega}}]$ et l'algorithme se termine. La deuxième étape consiste à déterminer l'abscisse $x_2 \geq x_1$ de telle sorte que la droite coupe $(1-\varepsilon)A_{\vec{\omega}}$ en x_2. Encore une fois, si x_2 s'avère supérieur à $L_{\vec{\omega}}$, il prend la valeur $L_{\vec{\omega}}$ et l'algorithme se termine. Ces deux étapes sont effectuées en résolvant une équation de la forme $\alpha e^{\zeta} + \beta\zeta + \gamma = 0$ en utilisant la fonction W de Lambert [70].

La figure 3.1 montre l'approximation d'une fonction $A_{\vec{\omega}}$ avec $\vec{\omega} \in \Omega$ à l'aide de plusieurs morceaux linéaires. Le sous-intervalle courant est le deuxième morceau défini entre x_0 et x_2. Il existe $x_1 \in [x_0, x_2]$ où le segment est tangent à e^{-x}.

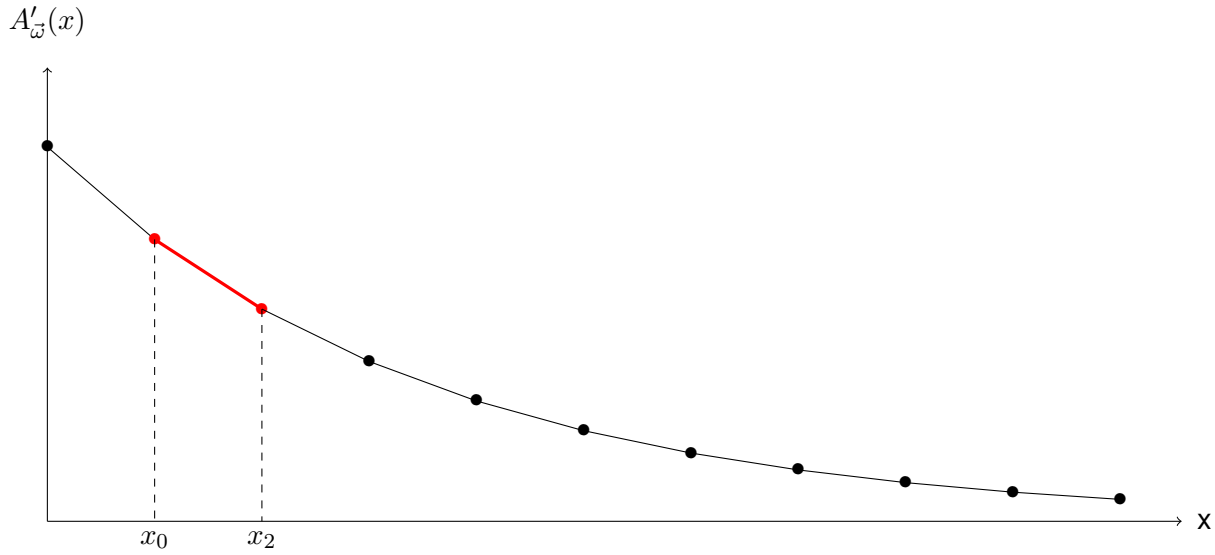

FIGURE 3.1 – Illustration d'une approximation par morceaux de $A_{\vec{\omega}}(x), \vec{\omega} \in \Omega$

Nous recherchons l'équation de la droite l, sachant qu'elle passe par le point de coordonnées $(x_0, (1-\varepsilon)A_{\vec{\omega}}(x_0))$. l est tangente à $A_{\vec{\omega}}$ en $x = x_1$. En utilisant (AP) définie en section 3.3 et $l(x) = ax + b$, la pente de cette droite est :

$$a = -\alpha(\vec{\omega})e^{-x_1}$$

et b est tel que :

$$(1-\varepsilon)A_{\vec{\omega}}(x_0) = ax_0 + b$$

$$b = (1-\varepsilon)\alpha(\vec{\omega})e^{-x_0} - ax_0$$

En remplaçant a par $-\alpha(\vec{\omega})e^{-x_1}$, nous obtenons

$$b = (1-\varepsilon)\alpha(\vec{\omega})e^{-x_0} + x_0\alpha(\vec{\omega})e^{-x_1}$$

Puisque la droite l est tangente à $A_{\vec{\omega}}$ en x_1, elle satisfait $A_{\vec{\omega}}(x_1) = ax_1 + b$, donc nous avons

$$\alpha(\vec{\omega})e^{-x_1} = -x_1\alpha(\vec{\omega})e^{-x_1} + (1-\varepsilon)\alpha(\vec{\omega})e^{-x_0} + x_0\alpha(\vec{\omega})e^{-x_1}$$

$$\Leftrightarrow 1 = -x_1 + (1-\varepsilon)e^{-x_0}e^{x_1} + x_0$$

$$\Leftrightarrow (1-\varepsilon)e^{-x_0}e^{x_1} - x_1 + x_0 - 1 = 0$$

En posant $\zeta = x_1$, nous avons :

$$(1-\varepsilon)e^{-x_0}e^{\zeta} - \zeta + x_0 - 1 = 0$$

Donc l'équation peut s'écrire $\alpha e^{\zeta} + \beta\zeta + \gamma = 0$ avec

$$\begin{aligned}\alpha &= (1-\varepsilon)e^{-x_0}\\ \beta &= -1\\ \gamma &= x_0 - 1\end{aligned}$$

Le nombre de solutions de cette équation est déterminé par le discriminant $\Delta = \frac{\alpha}{\beta}e^{-\frac{\gamma}{\beta}} = -(1-\varepsilon)e^{-x_0}e^{x_0-1} = -\frac{1}{e}(1-\varepsilon)$. Comme $\Delta \in]-\frac{1}{e}, 0[$, cette équation a deux solutions. Nous nous intéressons à la plus grande, car la plus petite est inférieure à x_0, donc :

$$x_1 = \left(-W_{-1}(\Delta) - \frac{\gamma}{\beta}\right) = \left(-W_{-1}(-\frac{1}{e}(1-\varepsilon)) + x_0 - 1\right)$$

$$\Leftrightarrow x_1 = x_0 - \left(W_{-1}(-\frac{1}{e}(1-\varepsilon)) + 1\right)$$

Si $x_1 \geq L_{\vec{\omega}}$, on fixe $x_1 = L_{\vec{\omega}}$ et l'algorithme se termine.

Connaissant x_1, nous recherchons $x_2 \geq x_1$ de telle sorte que la tangente à $A_{\vec{\omega}}$ en x_1 passe par le point de coordonnées $(x_2, (1-\varepsilon)A_{\vec{\omega}}(x_2))$. Cette tangente a pour équation $l(x) = ax + b$ avec $a = -\alpha(\vec{\omega})e^{-x_1}$, et $b = (1+x_1)\,\alpha(\vec{\omega})e^{-x_1}$. Comme elle passe par le point de coordonnées $(x_2, (1-\varepsilon)A_{\vec{\omega}}(x_2))$, on a :

$$(1-\varepsilon)A_{\vec{\omega}}(x_2) = ax_2 + b = (1-\varepsilon)\alpha(\vec{\omega})e^{-x_2}$$

$$\Leftrightarrow -x_2e^{-x_1} + (1+x_1)e^{-x_1} = (1-\varepsilon)e^{-x_2}$$

En posant $\zeta = -x_2$, cette égalité devient :

$$\zeta e^{-x_1} + (1+x_1)e^{-x_1} = (1-\varepsilon)e^{\zeta}$$

$$\Leftrightarrow (1-\varepsilon)e^{\zeta} - \zeta e^{-x_1} - (1+x_1)e^{-x_1} = 0$$

$$\Leftrightarrow \alpha e^{\zeta} + \beta\zeta + \gamma = 0$$

$$\begin{aligned}\alpha &= (1-\varepsilon)\\ \beta &= -e^{-x_1}\\ \gamma &= -(1+x_1)e^{-x_1}\end{aligned}$$

On sait que le nombre de solutions de la fonction W de Lambert dépend du discriminant $\Delta = \frac{\alpha}{\beta}e^{-\frac{\gamma}{\beta}} = -(1-\varepsilon)e^{x_1}e^{\frac{-(1+x_1)e^{-x_1}}{e^{-x_1}}} = -(1-\varepsilon)e^{x_1}e^{-(1+x_1)} = -\frac{1}{e}(1-\varepsilon)$.

Comme $\Delta \in]-\frac{1}{e}, 0[$, l'équation $\alpha e^{\zeta} + \beta\zeta + \gamma = 0$ a deux solutions. La plus petite est x_0, l'autre est $x_2 = \Big(W_0(\Delta) + \frac{\gamma}{\beta}\Big) = \Big(W_0(-\frac{1}{e}(1-\varepsilon)) + (1+x_1)\Big) = x_1 + \Big(W_0(-\frac{1}{e}(1-\varepsilon)) + 1\Big)$.

Nous obtenons donc une droite sur l'intervalle $[x_0, x_2]$ tangente à $A_{\vec{\omega}}$ dans x_1, avec :

$$\begin{aligned}x_1 &= x_0 - \Big(W_{-1}(-\tfrac{1}{e}(1-\varepsilon)) + 1\Big)\\ x_2 &= x_1 + \Big(W_0(-\tfrac{1}{e}(1-\varepsilon)) + 1\Big)\end{aligned}$$

Puisque $x_2 - x_0 = \Big(W_0(-\frac{1}{e}(1-\varepsilon)) - W_{-1}(-\frac{1}{e}(1-\varepsilon))\Big)$ ne dépend que de ε, une fonction linéaire par morceaux, comme $A'_{\vec{\omega}}$, dont la distance à $A_{\vec{\omega}}$ est bornée par ε sur $[0, L_{\vec{\omega}}]$, peut être construite en divisant cet intervalle en $p_g = \left\lceil \frac{L_{\vec{\omega}}}{x_2 - x_0} \right\rceil$ sous-intervalles qui ont tous pour largeur $x_2 - x_0$ (sauf le dernier, dont la largeur peut-être inférieure à $x_2 - x_0$). Par conséquent, pour tout $i \in \{1, \ldots, p_g\}$, nous définissons $\chi_i = (i-1)(x_2 - x_0)$ comme la borne inférieure du i-ème sous-intervalle de $[0, L_{\vec{\omega}}]$ et $\chi_{p_g+1} = L_{\vec{\omega}}$, donc pour tout $i \in \{1, \ldots, p_g - 1\}$, le i-ème morceau de g est le segment joignant les points $(\chi_i, (1-\varepsilon)A_{\vec{\omega}}(\chi_i))$ et $(\chi_{i+1}, (1-\varepsilon)A_{\vec{\omega}}(\chi_{i+1}))$. Le dernier morceau de g est le segment qui relie le point $(\chi_{p_g}, (1-\varepsilon)A_{\vec{\omega}}(\chi_{p_g}))$ au point $(L_{\vec{\omega}}, A_{\vec{\omega}}(L_{\vec{\omega}}))$ si $L_{\vec{\omega}} - \chi_{p_g} < \frac{x_2 - x_0}{2}$, et au point $(L_{\vec{\omega}}, aL_{\vec{\omega}} + b)$ sinon, où $y = ax + b$ est l'équation de la tangente à $A_{\vec{\omega}}$ en $x_1 = \chi_{p_g} + \frac{x_2 - x_0}{2}$.

On sait que pour tout réel $z \in]-\frac{1}{e}, 0[$, on a $e^1 z \leq W_0(z) \leq z$, et $-1 \geq W_{-1}(z) \geq \frac{1}{z}$. On sait aussi que $-\frac{1}{e}(1-\varepsilon) \in]-\frac{1}{e}, 0[$ si $\varepsilon \in [0, 1[$ donc nous obtenons $\varepsilon \leq x_2 - x_0$ avec $\varepsilon \in [0, 1[$ en utilisant la formule $x_2 - x_0 = \Big(W_0(-\frac{1}{e}(1-\varepsilon)) - W_{-1}(-\frac{1}{e}(1-\varepsilon))\Big)$.

Ainsi :

$$\frac{1}{x_2 - x_0} \leq \frac{1}{\varepsilon} \Rightarrow p_g \leq \left\lceil \frac{L_{\vec{\omega}}}{\varepsilon} \right\rceil$$

Lemme 1 Pour tous réels $A < B$, s'il existe d, un minorant linéaire approximant $A_{\vec{\omega}}$ sur $[A, B]$ avec une erreur inférieure ou égale à ε, alors $d_{\min}^{A,B}$, la droite qui passe par les points $(A, (1-\varepsilon)A_{\vec{\omega}}(A))$ et $(B, (1-\varepsilon)A_{\vec{\omega}}(B))$ est également un minorant linéaire approximant $A_{\vec{\omega}}$ sur $[A, B]$ avec une erreur inférieure ou égale à ε, et elle satisfait $d_{\min}^{A,B}(x) \leq d(x)$ pour tout x dans $[A, B]$.

Preuve du Lemme 1 S'il existe d, un minorant linéaire de $A_{\vec{\omega}}$ sur $[A, B]$ avec une erreur d'approximation inférieure ou égale à ε, alors $d(A) \geq (1-\varepsilon)A_{\vec{\omega}}(A)$ et $d(B) \geq (1-\varepsilon)A_{\vec{\omega}}(B)$, donc $d_{\min}^{A,B}(x) \leq d(x) \leq A_{\vec{\omega}}(x)$ pour tout $x \in [A, B]$. De plus, par construction, la droite $d_{\min}^{A,B}$ satisfait $(1-\varepsilon)A_{\vec{\omega}}(x) \leq d_{\min}^{A,B}(x)$ pour tout $x \in [A, B]$, donc $d_{\min}^{A,B}$ est un minorant linéaire de $A_{\vec{\omega}}$ sur $[A, B]$ avec une erreur d'approximation inférieure ou égale à ε. □

Lemme 2 Pour tous réels A et B tels que $0 \leq A < B \leq L_{\vec{\omega}}$ avec $B - A = x_2 - x_0$, $d_{\min}^{A,B}$ est l'unique minorant linéaire de $A_{\vec{\omega}}$ sur $[A, B]$ qui l'approxime avec une erreur inférieure ou égale à ε.

Preuve du Lemme 2 (par contradiction) Soit $d \neq d_{\min}^{A,B}$ un minorant linéaire approximant $A_{\vec{\omega}}$ sur $[A, B]$ avec une erreur inférieure ou égale à ε. Comme $(1-\varepsilon)A_{\vec{\omega}}(A) < d(A)$ ou $(1-\varepsilon)A_{\vec{\omega}}(B) < d(B)$, on a $d(x) > d_{\min}^{A,B}(x)$ pour tout $x \in (A, B)$. En particulier, $d(x_1) > d_{\min}^{A,B}(x_1) = A_{\vec{\omega}}(x_1)$, et comme la fonction d'approximation recherchée est toujours inférieure à la fonction objectif du problème d'origine, d n'est pas un minorant linéaire approximant $A_{\vec{\omega}}$ sur $[A, B]$ avec une erreur inférieure ou égale à ε. □

Théorème 1 $p_{A'_{\vec{\omega}}}$, le nombre de morceaux de $A'_{\vec{\omega}}$ qui est le minorant linéaire par morceaux de $A_{\vec{\omega}}$ sur $[0, L_{\vec{\omega}}]$ qui l'approxime avec une erreur inférieure ou égale à ε, est minimum.

Preuve du Théorème 1 (par contradiction) Soit h un minorant linéaire de $A_{\vec{\omega}}$ qui l'approxime avec une erreur inférieure ou égale à ε sur $[0, L_{\vec{\omega}}]$, pour lequel $p_h < p_{A'_{\vec{\omega}}}$. D'après le principe des tiroirs, h a au moins un morceau qui est un minorant linéaire approximant $A_{\vec{\omega}}$ sur $[A, C]$ avec une erreur inférieure ou égale à ε, pour lequel $C - A > x_2 - x_0$. Soit $B = A + x_2 - x_0 < C$. D'après le Lemme 2, $d_{\min}^{A,B}$ est le minorant linéaire par morceaux approximant $A_{\vec{\omega}}$ sur $[A, B]$ avec une erreur inférieure ou égale à ε. Par construction, $d_{\min}^{A,B}(x) < (1-\varepsilon)A_{\vec{\omega}}(x)$, pour tout $x \in]B, C]$ il n'y a donc pas de minorant linéaire approximant $A_{\vec{\omega}}$ sur $[A, C]$ avec une erreur inférieure ou égale à ε. □

3.5 Méthode de résolution

Dans cette section, nous présentons un algorithme pour résoudre le problème introduit dans ce chapitre. La complexité d'une instance moyenne est telle qu'une méthode exacte pour produire des solutions optimales n'est pas considérée car son temps d'exécution ne serait pas acceptable. En effet, comme on peut le voir dans le modèle 3.1, il y a de nombreuses contraintes (O($|C|^2n^2m$)) et de nombreuses variables de décision (O($|C|^2nm$)). De plus, le modèle est non linéaire avec une fonction objectif non linéaire difficile. Nos tentatives de ré-

solution d'un modèle relaxé, pourtant plus simple, d'instances de difficulté moyenne à l'aide de solveurs exacts se sont avérées beaucoup trop lentes, et n'ont pas pu produire de solution optimale avec un budget temps de huit heures. Nous proposons donc une méthode heuristique qui produit un ensemble de solutions suivant une approche gloutonne. Pour construire une solution, l'algorithme exécute plusieurs fois les trois étapes suivantes : allocation des ressources des capteurs, calcul des routes à l'aide d'une heuristique basée sur des utilités, puis amélioration locale de la solution (c'est-à-dire sans modifier les routes des capteurs). La méthode proposée peut s'apparenter à l'algorithme de Greedy Randomized Adaptive Search Procedure (GRASP) [71].

Dans cette section, nous présentons d'abord la stratégie globale de construction d'une solution, avant de détailler chaque composant de l'algorithme.

3.5.1 La construction d'une solution

Une solution est créée en répétant plusieurs fois les étapes suivantes jusqu'à ce qu'aucune ressource ne soit plus disponible :

1. Calcule les utilités pour les variables de décision
2. Sélection sur la base des utilités, d'un capteur, d'une cellule et d'une période de temps, et incrémention de pas de la variable de décision correspondante, où pas est la quantité de ressource allouée au cours d'une itération
3. Insertion de cette cellule dans la route du capteur au cours de cette période, si elle n'en fait pas déjà partie

Quand l'algorithme commence, la solution courante est vide : il n'y a aucune ressource allouée, et aucune route n'existe encore. Cette solution évolue à travers ces trois étapes, en augmentant toujours les ressources allouées de la valeur pas, jusqu'à ce qu'il n'y ait plus de ressource disponible pour chaque capteur à chaque période. La première étape consiste à calculer une utilité de manière gloutonne pour chaque variable de décision $\varphi_{ic}^{bt}, \forall t \in, \forall i \in I, \forall c \in C$. Les détails sur les utilités sont présentés dans la section 3.5.2. Dans la deuxième étape, l'algorithme sélectionne une variable de décision et augmente sa valeur de pas. Il alloue ainsi plus de ressources dans une cellule c pendant une période t pour un capteur i. Le mode de sélection de la variable de décision et le nombre de ressources ajoutées sont présentés dans la section 3.5.3. La dernière étape (section 3.5.4) ajoute la cellule c à la route du capteur i au cours de la période t, avec c, i, t correspondant à la variable de décision, sélectionnée lors de l'étape précédente.

3.5.2 Utilité

L'utilité est une partie importante de cet algorithme. Elle doit refléter, pour chaque variable de décision φ_{ic}^{bt}, son impact immédiat sur la fonction objectif mais aussi tenir compte de l'impact sur les ressources disponibles. En effet, l'allocation de ressources d'un capteur i à une cellule c pendant une période t, signifie que c doit être ajoutée à la route de i pendant t si elle n'en fait pas déjà partie. Il y a donc un coût en ressources supplémentaire à prendre en compte. Cela rend l'impact d'une telle décision (section 3.5.3) difficile à évaluer.

L'approximation de la fonction objectif, (AP) présentée dans la section 3.3, a la forme suivante :

$$\text{Minimiser} \quad -\sum_{\vec{\omega}\in\Omega}\sum_{i\in I}\sum_{t\in T} D_{i\vec{\omega}}^{t}\varphi_{i\vec{\omega}(t)}^{bt}$$

Où $\forall\vec{\omega}\in\Omega,\forall t\in T,\forall i\in I, D_{i\vec{\omega}}^{t}=\alpha(\vec{\omega})w_{\vec{\omega}(t)}^{i}e^{\sum_{i'\in I}\sum_{t'\in T}-w_{\vec{\omega}(t')}^{i'}\varphi_{i\vec{\omega}(t)}^{at}}$ est une constante. On peut reformuler cette fonction objectif comme suit :

$$\text{Minimiser} \quad -\sum_{c\in C}\sum_{i\in I}\sum_{t\in T}\left(\varphi_{i\vec{\omega}(t)}^{bt}\sum_{\vec{\omega}\in\Omega|\vec{\omega}(t)=c} D_{i\vec{\omega}}^{t}\right)$$

Cette approximation de la fonction objectif (qui est aussi un minorant) est recalculée à chaque itération de l'algorithme, en considérant donc les ressources réelles allouées de la solution courante après le dernier pas. De cette manière, l'approximation et les utilités s'adapteront le mieux possible à la fonction objectif réelle. Lors du calcul des utilités, les $D_{i\vec{\omega}}^{t},\forall t\in T,\forall i\in I,\forall\vec{\omega}\in\Omega$ sont constants. Ainsi, on peut proposer une première définition simple des utilités, dans laquelle les coûts de déplacement sont ignorés :

$$(U_1^{tic})=\sum_{\vec{\omega}\in\Omega|\vec{\omega}(t)=c} D_{i\vec{\omega}}^{t},\forall i\in I,\forall t\in T,\forall c\in C$$

Cette utilité est donc égale à la somme des $D_{i\vec{\omega}}^{t}$, pour toutes les trajectoires $\vec{\omega}$ telles que la cible est en c pendant t. Cette utilité est simple car elle évalue directement l'impact de toutes les variables de décision sur l'approximation linéaire (la pente de sa dérivée). Plus l'utilité est grande, plus il est intéressant d'augmenter la variable de décision correspondante.

Pour plus de précision, nous pénalisons maintenant l'utilité U_1^{tic} afin d'inclure une considération sur les coûts de déplacement engendrés par la sélection de la variable pour un pas. Nous proposons de multiplier cette utilité par la différence entre les ressources disponibles et une estimation heuristique du coût additionnel. Cette différence est $\rho_i^t - \text{add}_i^t(c)$ où $\text{add}_i^t(c)$ retourne une estimation du coût induit par l'ajout de c dans la route de i à la période t, et où ρ_i^t désigne les ressources restantes à la période t pour le capteur i. On obtient :

$$(U_2^{tic}) = \left(\sum_{\vec{\omega} \in \Omega | \vec{\omega}(t) = c} D^t_{i\vec{\omega}} \right) \times \left(\rho_i^t - \mathsf{add}_i^t(c) \right), \forall i \in I, \forall t \in T, \forall c \in C$$

L'estimation du coût d'ajout doit garantir d'être supérieur au coût d'ajout réel obtenu dans la section 3.5.4. Dans notre algorithme, ces estimations sont obtenues avec une méthode de meilleure insertion, décrite en détails à la section 3.5.4.

Cette utilité n'est négative que lorsque l'estimation du coût d'ajout est supérieure aux ressources disponibles. U_2^{tic} est également une évaluation de l'impact sur l'approximation linéaire actuelle si toutes les ressources de i disponibles à t sont consacrées à l'exploration de la cellule c.

3.5.3 Valeur du pas

Dans cette sous-section, nous discutons de la valeur affectée à un pas à chaque itération pour construire une solution. Chaque pas correspond à l'augmentation des ressources déjà allouées dans une cellule c, par un capteur i à la période t. Le triplet choisi $(c, i, t), c \in C, i \in I, t \in T$, correspond à une variable de décision φ_{ic}^{bt}. Cela signifie que la solution actuelle est modifiée de telle sorte que $\varphi_{ic}^{at} \leftarrow \varphi_{ic}^{at} + \varphi_{ic}^{bt}$, où φ_{ic}^{bt} est égal au minimum entre les ressources disponibles non allouées et la taille z d'un pas.

Dans notre algorithme, le triplet (c, i, t) est choisi aléatoirement parmi tous les triplets avec une utilité U_2^{tic} au moins λ fois aussi bonne que la meilleure utilité, et positive à 0. Ainsi, nous choisissons aléatoirement un triplet $(c, i, t), c \in C, i \in I, t \in T$ parmi ceux qui satisfont :

$$U_2^{tic} \geq \lambda \left(\max_{t' \in T, i' \in I, c' \in C} U_2^{t'i'c'} \right)$$

où $\lambda \in [0, 1]$ est un paramètre de la méthode. Par exemple, $\lambda = 1$ signifie que nous choisissons toujours la meilleure utilité et $\lambda = 0$ signifie que nous choisissons toujours une utilité totalement aléatoire. Par défaut, nous fixons $\lambda = 0.9$ de telle sorte que seulement l'une des meilleures utilités soit choisie tout en randomisant un peu la méthode. Ce comportement aléatoire glouton a été ajouté à notre travail afin d'éviter d'être piégé par des cas problématiques où les coûts de déplacement sont très élevés. Dans de tels cas, les capteurs sont contraints de rester dans des parties restreintes de la zone de recherche, correspondantes aux premières cellules sélectionnées. Plusieurs exécutions de notre algorithme avec $\lambda \neq 1$ sont donc susceptibles de sélectionner différentes premières cellules, explorant ainsi davantage la zone en évitant d'obtenir des solutions explorant systématiquement les mêmes zones de l'espace de recherche. On évite notamment les "pièges", ces cellules qui ont de bonnes utilités initiales mais sont éloignées des autres cellules prometteuses. Typiquement, le cas particulier multisous-zone de notre problème [41, 48], où, une fois affecté à une sous-zone, le capteur est forcé de recher-

cher la cible uniquement cette sous-zone. Plusieurs exécutions de notre algorithme affecteront un capteur à différentes sous-zones, ce qui devrait conduire à des bonnes solutions.

Si aucune utilité ne peut être sélectionnée, ce qui arrive quand $\max_{t\in T, i\in I, c\in C} U_2^{tic} \leq 0$, l'algorithme s'arrête. Cela correspond à une solution où tous les capteurs ont toutes leurs ressources utilisées dans toutes les périodes.

La taille des pas z est un paramètre important. Un grand pas consiste à attribuer beaucoup de ressources dans la cellule sélectionnée, et c'est une décision sur laquelle notre algorithme ne peut pas revenir. Cependant, cette décision est basée sur une utilité gloutonne utilisant la fonction d'approximation. Lorsque les ressources allouées augmentent, les utilités changent et la décision peut s'avérer ne plus être intéressante. En effet, nous avons vu dans la section 3.3 que lorsque les valeurs des variables de décision sont trop importantes, l'approximation linéaire est plus éloignée de la fonction objectif réelle. Ainsi, les grands pas ne sont pas recommandés car ils allouent beaucoup de ressources dans une cellule en suivant une utilité qui en réalité peut ne pas être bonne sur toute la longueur du pas.

En revanche, les petits pas offrent des décisions beaucoup plus prudentes et précises. Étant donné que l'approximation et les utilités sont mis à jour après chaque étape, chaque décision est prise en tenant compte de plus d'un état plus précis et actualisé du problème. Cependant, cela affecte beaucoup plus le temps de calcul car cela signifie plus d'itérations pour allouer toutes les ressources disponibles.

Dans notre algorithme, la taille d'un pas z est un paramètre d'entrée. Il est recommandé de choisir de petites valeurs calculées en fonction des ressources disponibles afin de garder un contrôle sur le nombre de pas. Par défaut (si aucune valeur pour le pas n'est fournie), nous fixons la taille du pas à 1% du minimum des ressources disponibles ($z = 0,01 \times \min_{t\in T, i\in I} \Phi_i^t$).

3.5.4 Ajout de capteurs aux routes

Dans cette sous-section, nous présentons la dernière étape de chaque itération de l'algorithme. Dans l'étape précédente, un triplet (c, i, t) a été sélectionné pour un pas. Il faut alors s'assurer que la cellule c est dans la route du capteur i pendant la période t, afin qu'elle puisse être effectivement explorée.

Une méthode de meilleure insertion est utilisée. C'est une méthode heuristique où la nouvelle cellule est introduite dans la route du capteur, sans modifier l'ordre de visite des cellules déjà présentes. La nouvelle cellule c est insérée à la période qui conduit à l'augmentation minimale du coût de déplacement de la route. Si c est insérée dans la route à la place de l'arc (a, b), alors nous remplaçons cet arc par (a, c) et (c, b). Cela induit un coût d'insertion de $t_{ac} + t_{cb} - t_{ab}$. Ce coût est retiré des ressources disponibles dans toutes les périodes concernées, c'est-à-dire les périodes où le capteur se déplace le long de (a, c) et (c, b). Pour les voyages intra-périodes

(c n'est ni la première ni la dernière cellule visitée de sa période), la méthode retire uniquement des ressources dans la période concernée. Pour les déplacements inter-périodes (c est la première ou la dernière cellule de sa période), la méthode retire ce coût des ressources de toutes les périodes concernées. Une telle méthode produit une nouvelle solution où toutes les périodes concernées par un déplacement inter-périodes disposent de suffisamment de ressources pour faire face à son coût. Un dernier algorithme de réparation partagera l'excédent de ressources épargnées entre les périodes concernées.

Ce sont ces mêmes coûts qui sont utilisés dans le calcul des utilités des variables de décision. On peut remarquer qu'il ne serait pas très difficile d'adapter la méthode de résolution proposée au problème avec dépôt pour les capteurs (les capteurs doivent commencer et finir leur mission de recherche dans un dépôt, ou commencer et finir chaque période dans un dépôt).

3.5.5 Optimisation locale de la solution

Cette sous-section présente la méthode mise en œuvre pour optimiser localement une solution produite par les étapes antérieures. Il s'agit d'une optimisation locale car les routes qui seront utilisées par les capteurs ne sont pas remises en cause, et seule une réallocation des ressources est effectuée. Dans cette méthode, la réallocation a lieu pour chaque paire de périodes $(t_1, t_2), \forall (t_1, t_2) \in T^2$ avec un déplacement inter-périodes commun commençant à t_1 et se terminant à t_2. Toutes les ressources déjà allouées entre ces deux périodes sont réallouées (c'est-à-dire, $\sum_{i \in I, c \in C} \varphi_{ic}^{at_1} + \varphi_{ic}^{at_2}$). De plus, le coût du déplacement inter-périodes est également réalloué (puisque ce coût a été épargné deux fois, une fois dans chaque période). La méthode de réallocation est la suivante :

1. Calculer les utilités $U_1^{tic} \times \rho_i'^t$ des variables de décision
2. Allouer des ressources à la variable de décision associée à la meilleure utilité (si elle est supérieure à 0)

Il s'agit d'une version allégée de l'algorithme précédent, sans aucune considération pour les coûts de déplacement car les routes sont déjà calculées. Cette version affecte à nouveau les ressources, sans jamais allouer plus de ressources que celles qui sont disponibles et sans compromettre les routes existantes.

3.6 Expérimentations numériques

Dans cette section, nous étudions l'efficacité de notre méthode, en présentant des résultats et des temps d'exécution obtenus sur de nombreuses instances. Nous avons conçu plusieurs

expériences, afin de tester l'impact de nombreux paramètres. Les paramètres identifiés comme importants sont le nombre de cellules, de capteurs, de trajectoires et de périodes, dont on s'attend à ce que l'augmentation conduise à un accroissement du temps de calcul, mais avec un impact différent sur la valeur de la fonction objectif. Nous ne comparons pas nos résultats avec ceux des articles de la littérature [41, 48]. En effet, comme indiqué précédemment, le problème abordé dans ces travaux constitue un cas particulier du nôtre et la complexité est bien moindre. Par exemple, une fois qu'un capteur est affecté à une sous-zone, il ne reste plus qu'à allouer les ressources aux cellules, alors que le problème de ce chapitre nécessite le calcul d'une route entre les cellules allouées, même si la route calculée a toujours un coût nul. Dans le cas d'une cible mobile, pour réutiliser leurs instances, il faudrait également multiplier le nombre de capteurs par le nombre de périodes, ce qui conduirait à des problèmes beaucoup plus complexes à résoudre. Dans l'instance utilisée par [48], nous aurions besoin de 24 capteurs au lieu de 6 pour représenter la même instance. Par conséquent, nous considérons qu'une comparaison entre leurs résultats et les nôtres n'aurait pas beaucoup de sens.

Tous les temps d'exécution sont des temps CPU en secondes. Toutes les valeurs pour la fonction objectif sont des valeurs moyennes calculées sur des ensembles d'instances.

3.6.1 Description du générateur d'instances

Afin de tester notre méthode sur de nombreuses instances différentes, nous avons conçu un générateur d'instances. Le générateur prend différents paramètres en entrée et crée une instance discrétisée en sortie. La zone de recherche est représentée par un quadrillage en cellules caractérisé par n_r, le nombre de lignes et n_c le nombre de colonnes. Ainsi, le nombre de cellules est égal à $n_c \times n_r$. Le coût de déplacement entre deux cellules est la distance euclidienne telle que les deux cellules les plus éloignées ont une distance égale à $\min_{t \in T, i \in I} \Phi_i^t$, où Φ est un paramètre relatif aux ressources, décrit plus loin. Nous considérons que ces coûts ont un impact sur les solutions sans être trop restrictifs. Le nombre de capteurs m et de périodes n sont également des paramètres. Pour chaque capteur, une visibilité aléatoire entre 0 et 1 est choisie pour chaque cellule. Les ressources disponibles pour chaque période, pour chaque capteur, sont égales à Φ, un paramètre contrôlant l'énergie disponible.

La génération des trajectoires est une partie importante de ce générateur. D'une part, nous voulons pouvoir générer un nombre donné de trajectoires. Ceci permet, au cours de nos expérimentations numériques, d'étudier librement d'autres paramètres sans se soucier de l'impact sur le nombre de trajectoires. D'autre part, nous voulons générer des trajectoires réalistes, de la même manière que [41, 48] qui représentent un mouvement incertain dans une direction entre chaque période : à partir d'une cellule, la cible a une probabilité de déplacement propre à chaque direction qu'elle peut prendre. À cette fin, nous avons conçu deux manières de générer des trajectoires. La première a un nombre fixe de trajectoires, fixé à 500. Ces trajectoires

sont calculées en choisissant d'abord une cellule de départ aléatoirement, puis en choisissant un déplacement aléatoire entre chaque période, en suivant les probabilités de la matrice de la figure 3.2. Si une transition n'est pas possible (par exemple, la cible est dans un coin), sa probabilité est répartie uniformément entre les autres transitions qui ont des probabilités non nulles. Ainsi, on garde une distribution égale des probabilités parmi les transitions possibles. Les probabilités des trajectoires sont uniformément aléatoirement choisies de sorte que leur somme soit égale à 1. La deuxième manière utilisée pour générer des trajectoires se fait en sélectionnant v cellules de départ aléatoires, les cellules où la cible se trouve à la période 1. v est un paramètre. Toutes les trajectoires possibles à partir de ces cellules sont générées, suivant la matrice de transition de la figure 3.2. Par exemple, avec deux périodes, à partir d'une cellule éloignée des bords, il y a quatre trajectoires avec des probabilités égales. La première vers le haut, la deuxième à gauche, ensuite à droite et la dernière vers la cellule du bas. Nous ne contrôlons pas le nombre de trajectoires générées car il y a moins de transitions possibles lorsque la cellule est adjacente à la limite de l'espace de recherche.

Dans toutes nos expérimentations, le paramètre λ est fixé à 1 et une seule solution est générée.

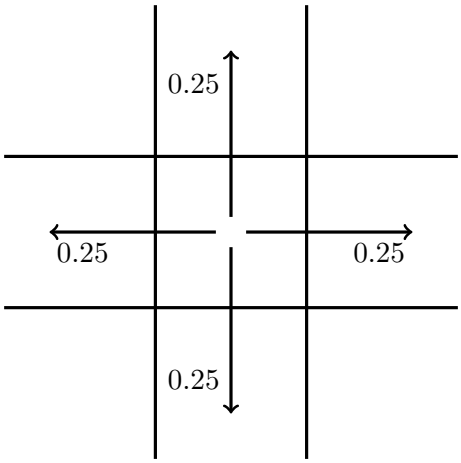

FIGURE 3.2 – Matrice de transition

Les paramètres par défaut sont donnés dans le tableau 3.2.

$n_r \times n_c$	le nombre de cellules	12×10
m	le nombre de capteurs	3
n	le nombre de périodes	4
Φ	la quantité de ressources à chaque période	5

Tableau 3.2 – Les paramètres par défaut

3.6.2 Expérimentation 1 – Cellules

Dans cette expérience, nous mesurons l'impact du nombre de cellules. Plus il y a de cellules, plus la recherche est précise. Cependant, l'ajout de cellules complique certainement le processus de résolution. Les instances de cette expérience ont tous les paramètres par défaut et seul le nombre de cellules varie. Les différentes valeurs du nombre de cellules $|C|$ testées sont $\{5 \times 5, 8 \times 8, 10 \times 10, 15 \times 15, 20 \times 20\}$. Cet ensemble couvre une grande variété d'instances avec des instances simples (5×5 et 8×8), moyennes (10×10 et 15×15) et plus dures (20×20). Pour chaque nombre de cellules testé, un ensemble de 100 instances est généré. Les instances simples (5×5 et 8×8) ont peu de cellules et ont donc une densité de présence de la cible élevée, c'est-à-dire qu'à chaque période l'ensemble des trajectoires possibles est limité à quelques cellules. La recherche de la cible dans ces cellules est alors très efficace et l'on peut s'attendre à de très faibles probabilités de non-détection. La densité diminue avec l'augmentation du nombre de cellules, et ainsi les trajectoires ont beaucoup moins de parties communes. Cela signifie qu'il y a moins d'associations cellule-période qui sont communes à plusieurs trajectoires, et donc l'effort de recherche est beaucoup moins efficace. On peut donc s'attendre a une probabilité de non-détection qui s'accroît. De plus, cela a pour effet d'augmenter le nombre de variables de décision et donc le nombre d'utilités à calculer, ainsi que les chemins des capteurs. L'augmentation du nombre de cellules devrait donc avoir un impact considérable sur le temps d'exécution. Les moyennes des résultats obtenus sont présentées dans le tableau 3.3.

#Cellules	Probabilité de non-détection	Temps CPU (s)
5×5	0.2061	0.204
8×8	0.4699	0.393
10×10	0.5699	0.727
15×15	0.6962	2.827
20×20	0.7534	7.310

Tableau 3.3 – Résultats en moyenne avec différents nombres de cellules

Ces résultats montrent l'importance du nombre de cellules sur le temps CPU. En effet, le temps de résolution augmente rapidement dans notre expérience lorsque nous augmentons le nombre de cellules, et augmente même plus rapidement avec les dernières valeurs testées. Cependant, même pour les plus grandes instances, jusqu'à 400 cellules, le temps de résolution reste acceptable. Cela montre que notre méthode passe à l'échelle, en termes de nombre de cellules. Néanmoins, cela reste irrémédiablement un paramètre important dont dépend fortement le temps de résolution, et qui est donc à ne pas négliger lorsque l'on choisit le nombre de cellules au cours de la discrétisation d'une instance. Ainsi, lors de la division de la zone de recherche E en cellules, le nombre de cellules doit être choisi en connaissance de cause.

Les parties communes entre les trajectoires expliquent pourquoi la valeur de l'objectif se dégrade lorsque le nombre de cellules augmente. Cependant, avoir plus de cellules permet de mieux représenter la recherche réelle lorsque la zone de recherche est hétérogène.

Pour conclure, notre méthode supporte assez bien un nombre élevé de cellules, ce qui est important pour représenter la zone de recherche de manière réaliste.

3.6.3 Expérimentation 2 – Capteurs

Cette deuxième expérience se concentre sur le nombre de capteurs. On s'attend à ce qu'un nombre plus élevé de capteurs tende à améliorer la valeur de l'objectif, au prix d'un allongement du temps de résolution. Un premier ensemble de cent instances avec 6 capteurs est généré et résolu. Ensuite, nous supprimons les capteurs et résolvons à nouveau de sorte que chaque nombre de capteurs $m \in \{1, 2, 3, 4, 5, 6\}$ soit testé pour chaque instance. Ainsi, les trajectoires, les probabilités de présence et les visibilités sont les mêmes pour une instance à travers chaque nombre de capteurs testés. Cela permet aux résultats obtenus de ne montrer que l'impact du nombre de capteurs. Les résultats en moyenne sont présentés dans le tableau 3.4.

#Capteurs	Probabilité de non-détection	Temps CPU (s)
1	0.8427	0.109
2	0.7099	0.433
3	0.5979	0.989
4	0.5028	1.782
5	0.4219	2.808
6	0.3531	4.099

Tableau 3.4 – Résultats en moyenne avec différents nombres de capteurs

Ces résultats montrent que, comme prévu, le problème devient de plus en plus difficile à résoudre quand le nombre de capteurs augmente, mais fournit de meilleures valeurs pour la fonction objectif. Tout d'abord, le temps CPU augmente constamment avec le nombre de capteurs. Plus précisément, il y a une augmentation du temps de résolution chaque fois qu'un capteur est ajouté au problème. L'augmentation est plus importante lorsqu'il y a plus de capteurs (par exemple, ≈ 0.32 s d'augmentation entre 1 et 2 capteurs, et ≈ 1.29 s entre 5 et 6). Cependant, au regard du pourcentage du temps CPU, l'augmentation est plus faible. Il y a, par exemple, ≈ 397 % d'augmentation entre 1 et 2 capteurs, et ≈ 146 % entre 5 et 6. Ces deux comportements sont visibles pour tous les nombres de capteurs testés. Sur cette partie, nous concluons que l'ajout d'un capteur à une instance augmente le temps de résolution, avec une augmentation qui dépend du nombre de capteurs déjà utilisés (dans nos résultats, $m - 1$ fois un coefficient proche de 0.5). Le temps de résolution pour obtenir une solution est vraiment très

faible avec quelques capteurs et reste acceptable pour toutes les valeurs testées.

Deuxièmement, la valeur de l'objectif diminue évidemment lorsque davantage de capteurs sont utilisés. Dans cet ensemble de cas, la diminution semble être constante mais non linéaire, une décroissance de type exponentielle décroissante. En effet, dans notre expérimentation, chaque fois qu'un capteur est ajouté, la valeur de l'objectif perd approximativement 16 % de sa valeur.

Cette expérience montre l'impact du nombre de capteurs, rendant le problème plus complexe à résoudre mais permettant d'atteindre de meilleures résultats. Notre solution évolue bien avec l'augmentation des capteurs, avec des temps CPU toujours acceptables lorsque six capteurs sont utilisés.

3.6.4 Expérimentation 3 – Périodes

Dans cette expérience, nous nous concentrons sur le nombre de périodes. De manière similaire aux expériences précédentes, nous voulons tester notre algorithme sur un nombre de périodes n variant dans l'ensemble $\{1, 2, 3, 4, 5, 6\}$. Nous générons un ensemble de 100 instances pour chaque valeur de n, avec les autres paramètres fixés à leurs valeurs par défaut à l'exception des ressources disponibles. En effet, nous nous intéressons à l'impact sur la fonction objectif de trajectoires plus précises, puisque davantage de périodes (plus courtes) seront utilisées pour représenter la trajectoire d'une cible mobile. Par conséquent, nous devons nous assurer que le budget de recherche global pour chaque capteur ne change pas. Avec les valeurs de $n \in \{1, 2, 3, 4, 5, 6\}$, la quantité de ressources dans chaque période pour chaque capteur est respectivement égale à $\{6, 3, 2, 1.5, 1.2, 1\}$ de telle sorte que la quantité totale sur l'ensemble des périodes pour un capteur soit toujours de 6. Les résultats en moyenne sont présentés dans le tableau 3.5.

#Périodes	Probabilité de non-détection	Temps CPU (s)
1	0.8605	0.022
2	0.8516	0.137
3	0.8474	0.423
4	0.8452	0.972
5	0.8443	1.865
6	0.8462	3.298

Tableau 3.5 – Résultats en moyenne avec différents nombres de périodes

Ces résultats montrent que le nombre de périodes a un impact sur le temps de résolution. Un petit nombre de périodes est résolu très rapidement, en particulier le cas statique (une période). Alors qu'avec beaucoup de périodes, la méthode a besoin de beaucoup plus de temps. Par exemple, six périodes nécessitant 146 fois plus de temps qu'une période, un ratio impor-

tant. L'évolution du temps nécessaire est similaire à ce que l'on a observé à la section 3.6.3 (augmentation de temps plus importante en secondes avec plus de périodes mais plus petit pourcentage). Ces résultats montrent que le nombre de périodes a un impact important sur le temps de résolution et marque vraiment la différence entre les instances faciles (une ou deux périodes) et les instances moyennes à difficiles.

Du côté de la valeur de la fonction objectif, il y a de petites diminutions lors de l'augmentation du nombre de périodes, sauf pour six périodes bien que les valeurs restent relativement proches. Cela est dû au fait qu'avec plus de périodes, il y a une probabilité plus élevée que certaines trajectoires aient des cellules en commun. Ainsi, cela rend plus efficace et rentable un effort de recherche dans ces cellules.

Pour conclure, le nombre de périodes n'a qu'un effet peu sensible sur la valeur de la fonction objectif, mais il impacte significativement le temps de résolution. Le comportement de notre méthode évolue assez bien et conserve des temps de calcul faibles même si l'on considère jusqu'à six périodes.

3.6.5 Expérimentation 4 – Trajectoires

Dans cette dernière expérience, nous étudions l'impact du nombre de trajectoires. Comme mentionné dans la littérature [41], cela a un impact considérable sur la complexité combinatoire du problème. Nous visons ici à tester son effet sur les temps de résolution de notre méthode et la qualité des solutions produites. Pour cela, nous considérons deux ensembles d'instances, chacun recourant à un processus différent pour générer les trajectoires (voir la section 3.6.1). Le premier ensemble, noté Q_1, a toutes les trajectoires possibles (suivant la matrice de transition) faites à partir d'un nombre donné de cellules de départ fixé. Cela produit des cas réalistes, avec des trajectoires très similaires. Le paramètre pour cette génération est le nombre de cellules de départ. Le deuxième ensemble, noté Q_2, a les trajectoires cibles tracées de manière similaire aux expériences précédentes, où chaque trajectoire est obtenue en choisissant une cellule de départ aléatoire et une transition aléatoire entre chaque période. Le nombre de trajectoires à tracer est un paramètre d'entrée. Ces instances sont susceptibles d'avoir des trajectoires disjointes, avec seulement quelques similitudes entre elles. Afin de tester approximativement le même nombre de trajectoires, le premier ensemble est d'abord généré, avec 100 instances pour chaque nombre de cellules de départ dans $\{5, 10, 20, 30, 40, 50\}$. Ensuite, le deuxième ensemble est généré en utilisant le nombre moyen de trajectoires arrondi comme paramètre sur chaque ensemble d'une centaine d'instances. Les résultats du premier ensemble sont présentés dans le tableau 3.6 et ceux qui sont relatifs au second ensemble apparaissent dans le tableau 3.7.

Ces expérimentations numériques montrent que le nombre de trajectoires a un impact sur le temps d'exécution de notre méthode, mais qu'il est assez faible. En effet, dans les deux

#Cellules de départ	#Trajectoires	Probabilité de non-détection	Temps CPU (s)
5	256.25	0.0313	0.484
10	498.07	0.1290	0.732
20	954.94	0.2483	1.015
30	1385.12	0.3058	1.180
40	1793.81	0.3274	1.298
50	2166.32	0.3285	1.391

Tableau 3.6 – Résultats en moyenne sur l'ensemble d'instances Q_1 avec le nombre de cellules de départ en paramètre

#Trajectoires	Probabilité de non-détection	Temps CPU (s)
256	0.5235	0.880
498	0.5968	0.989
955	0.6457	1.099
1385	0.6651	1.210
1794	0.6774	1.307
2166	0.6847	1.402

Tableau 3.7 – Résultats en moyenne sur l'ensemble d'instances Q_2 avec le nombre de trajectoires à tracer en paramètre

ensembles, il y a une augmentation du temps CPU, mais il reste modéré. Même avec le plus grand nombre de trajectoires testé, le temps d'exécution est inférieur à 1.5 secondes. Une trajectoire n'ajoute pas nécessairement de variable de décision et n'ajoute donc pas plus d'utilité à calculer. C'est pourquoi notre méthode est peu impactée par le nombre de trajectoires, car il y a déjà beaucoup de trajectoires par rapport à la taille de la zone explorée. De toute évidence, le temps d'exécution du premier ensemble est le plus impacté. En effet, la génération de la trajectoire dans cet ensemble est telle qu'elle ajoute beaucoup plus de variables de décision que dans le cas du second ensemble.

L'évolution de la fonction objectif montre, comme prévu, qu'avec plus de trajectoires, la qualité des solutions obtenues diminue. En outre, la comparaison des tableaux 3.6 et 3.7 montre que la probabilité de non-détection est plus facile à minimiser lorsque les trajectoires ont beaucoup de cellules en commun. Cependant, concernant le temps de résolution, notre méthode est plus rapide pour résoudre les cas où les trajectoires sont toutes aléatoires, donc avec moins de parties en commun. Ce temps plus rapide peut être causé par des routes de capteurs plus petites en termes de nombres de cellules recherchées, ce qui entraîne moins de calculs pour les coûts d'ajouts et insertions. En effet, on peut s'attendre dans ce cas à ce que les trajectoires soient plus éloignées les unes des autres, puisqu'elles sont aléatoirement disséminées dans la zone et non pas tracées autour de cellules de départ. Il y a alors moins de cas où de petits déplacements avec faible coût entre cellules voisines permettent de rechercher la cible sur de

nombreuses trajectoires.

Pour conclure, le nombre de trajectoires n'affecte pas beaucoup le temps de résolution de notre méthode. Cependant, l'augmentation du nombre de trajectoires affecte négativement la qualité des solutions au problème.

3.6.6 Résultats globaux

Nous rapportons dans cette dernière sous-section des résultats globaux sur toutes nos instances. Tout d'abord, nous abordons l'écart moyen avec la borne inférieure. Pour chaque instance générée, nous avons calculé la borne inférieure avec une erreur ϵ fixée 10^{-3}. L'écart moyen (c'est-à-dire, $\frac{f}{l} - 1$ avec f la valeur de la fonction objectif de la solution produite, et l la borne inférieure) est égal à 15% de la borne inférieure. Toutefois, pour le premier ensemble de trajectoires, les bornes inférieures sont très proches de 0. Étant donné que les coûts de déplacement ont un impact sur la solution (juste assez d'énergie pour voyager entre les deux cellules les plus lointaines), la limite inférieure devrait être un peu éloignée de la valeur de la fonction objectif à l'optimum. Par conséquent, nous concluons que notre algorithme produit des solutions dont la qualité est acceptable.

Deuxièmement, nous avons testé l'exécution de notre algorithme avec $\lambda = 0.9$ (voir section 3.5.3) et dix solutions à générer. Les meilleures valeurs de la fonction objectif renvoyées par notre algorithme sont similaires à celles obtenues avec $\lambda = 1$, avec un temps de calcul environ dix fois plus long. Nous en concluons que la génération de solutions multiples n'apporte aucun avantage ici. Nous avons aussi effectué des expérimentations complémentaires, en comparant les résultats obtenus avec $\lambda = 0.9$ et avec $\lambda = 1$ sur les instances les plus simples de [40, 41]. Les résultats obtenus montrent alors que $\lambda = 0.9$ produit des solutions de meilleure qualité lorsque nous avons des coûts prohibitifs de déplacement qui rendent inaccessible une grande partie de la zone de recherche une fois la première cellule sélectionnée.

3.7 Conclusion du troisième chapitre

Dans ce chapitre, nous avons proposé une nouvelle formulation du problème de la planification d'une recherche multicapteurs pour une cible en mouvement. Nous avons généralisé les travaux précédents de la littérature en ajoutant des coûts de déplacement pour les capteurs entre les cellules à explorer. Cette généralisation rend le problème beaucoup plus complexe à résoudre car une solution doit déterminer une route pour chaque capteur, afin de lui permettre d'explorer les cellules qui lui ont été allouées. Nous avons présenté la formulation mathématique du problème et une méthode pour calculer efficacement une borne inférieure dont l'erreur est maîtrisée. Ensuite, nous avons présenté un nouvel algorithme pour résoudre

notre problème qui, à la différence des travaux précédents, ne repose pas sur une méthode de type progressif-rétrogressif contrairement aux méthodes de la littérature. Nous avons analysé l'efficacité de notre algorithme avec des milliers d'instances et mis en évidence les principaux paramètres ayant le plus d'impact sur le temps d'exécution et la qualité des solutions. De nombreuses extensions sont envisagées pour ce problème. La première consiste à considérer plusieurs cibles à trouver. Adapter notre méthode à ce problème nécessitera donc de modifier les utilités uniquement. Une autre extension également envisagée est de coupler recherche et suivi de cible. Dans ce problème, une fois la cible trouvée, il est nécessaire de la suivre, et une méthode dynamique pour la partie suivi peut être recherchée.

CONCLUSION

Résumé des travaux

Dans cette thèse nous avons abordé plusieurs problèmes d'optimisation d'ensembles de capteurs. Le premier problème considéré est celui du suivi de cible à l'aide d'un ensemble de capteurs qui forment un réseau sans fil. Les capteurs qui le constituent ont des positions fixes dans l'espace résultant d'un déploiement aléatoire. L'objectif est de couvrir un ensemble de cibles dont la trajectoire spatiale est connue, à l'aide des capteurs, qui doivent récolter des données les concernant et les retransmettre à une station de base présente dans la zone. Les difficultés du problème abordé sont les possibles retards et avances des cibles. Dans le premier chapitre, ce problème est résolu à l'aide d'une approche robuste, où l'on calcule hors-ligne un ordonnancement robuste qui maximise un rayon de stabilité. Cet ordonnancement garantit la couverture des cibles tant que leurs retards et avances ne dépassent pas la valeur du rayon de stabilité. Dans le deuxième chapitre, nous étendons cette approche robuste à l'aide d'une méthode dynamique. La méthode dynamique permet d'adapter l'ordonnancement aux retards et avances effectifs des cibles au-delà du rayon de stabilité, et donc d'étendre la garantie de couverture. La garantie de performance est mesurée par un rayon de stabilité dynamique qui est maximisé dynamiquement. Le troisième chapitre aborde un problème différent, où les capteurs sont utilisés pour trouver une cible. Cette fois-ci, les capteurs sont mobiles et recherchent une cible dans une zone discrétisée. L'approche proposée est probabiliste, elle vise à minimiser la probabilité de ne pas découvrir la cible. Dans le reste de cette section, nous présentons quelques perspectives ouvertes par ces travaux.

Perspective avancée : le problème de robustesse spatiale

Une perspective envisagée, mentionnée dans la conclusion du chapitre 1, est le traitement du problème d'incertitude spatiale. On reprend ici le même problème que dans le chapitre 1, ainsi que les même notations, et on considère que les trajectoires des cibles peuvent être décomposées en segments de droites. La particularité de ce problème est que désormais, les cibles peuvent s'écarter de leurs trajectoires spatiales estimées. C'est-à-dire qu'à un instant t donné, la cible j peut se trouver en un point de l'espace n'appartenant pas à sa trajectoire estimée. Ce n'est pas le cas avec l'incertitude temporelle où la cible se trouve toujours sur sa trajectoire estimée mais potentiellement plus tôt ou plus tard. C'est un problème très intéressant

que l'on retrouve dans de nombreuses applications industrielles ou militaires.

Dans cette perspective avancée, nous travaillons toujours sur une approche robuste pour couvrir les incertitudes. La méthode recherchée doit produire un ordonnancement réalisable maximisant un rayon de stabilité spatial R. Plus précisément, on garantit qu'une cible $j \in J$ est couverte à un instant t pourvu qu'elle se trouve à une distance d'au plus moins de R de $\tau_j(t)$ (la position estimée de j à t). Dans la méthode de résolution proposée, l'ordonnancement est produit à l'aide d'une approche analogue à celle qui a été proposée pour l'incertitude temporelle, avec une discrétisation en faces et fenêtres de temps délimitées par des ticks. Le problème revient donc à surveiller chaque cible en utilisant les capteurs candidats de ses fenêtres de temps. Un capteur est dit candidat s'il est capable de couvrir toutes les faces où peut se trouver la cible à l'instant considéré. Les faces où la cible j peut se trouver, à un instant t pour un rayon de stabilité R, sont toutes celles qui ont une intersection non vide avec le disque d'incertitude de rayon R centré en $\tau_j(t)$. Les faces qui ne font qu'ajouter des capteurs par rapport à l'ensemble des capteurs couvrant la position estimée (c'est-à-dire celles que rencontre la cible quand elle arrive à portée d'un nouveau capteur) ne sont pas considérées. En effet, ces nouveaux capteurs ne peuvent pas être candidats puisqu'ils ne couvrent pas la cible quand elle se trouve sur sa trajectoire prévisionnelle. Ainsi, pour déterminer les capteurs candidats pour j à t avec un rayon R, il suffit de sélectionner tous ceux qui couvrent entièrement le disque d'incertitude centré en $\tau_j(t)$ et de rayon R. Autrement dit, ce sont tous les capteurs qui sont à portée de $\tau_j(t)$ et dont la portée de surveillance est supérieure à la distance entre la position du capteur et $\tau_j(t)$ plus le rayon R.

Un exemple est donné avec la figure Conc.1 qui reprend l'exemple de la figure 1.2 avec un rayon de stabilité R. Dans cet exemple, pour l'instant t donné, la cible doit être couverte qu'importe où elle se situe dans le disque rouge représentant le disque d'incertitude de rayon R et centré sur la position estimée de la cible à l'instant t.

On a initialement les mêmes fenêtres de temps et les mêmes capteurs candidats pour $R = 0$ que pour $\rho = 0$ dans le problème du premier chapitre. Il s'agit des fenêtres de temps délimitées par les ticks initiaux qui correspondent aux instants où la trajectoire estimée croise la limite de la portée d'un capteur. En augmentant R, on décale ces ticks. Cependant, avec R croissant, un capteur peut ne plus être candidat à la surveillance d'une cible, sans que cela corresponde au déplacement d'un tick initial. C'est le cas lorsque le disque d'incertitude quitte la portée du capteur par un autre point qu'un croisement entre la limite de la portée d'un capteur et la trajectoire estimée. Ce nouveau problème est donc plus difficile en raison de l'apparition de nouveaux ticks (et donc de nouvelles fenêtres de temps) lorsque R atteint certaines valeurs.

Ces ticks et fenêtres de temps n'apparaissent qu'aux extrémités des segments de droites constituant la trajectoire estimée des cibles, donc au début et à la fin de chaque trajectoire, ainsi qu'à tous ses changements de direction. Cet ensemble d'extrémités est noté Q. Pour

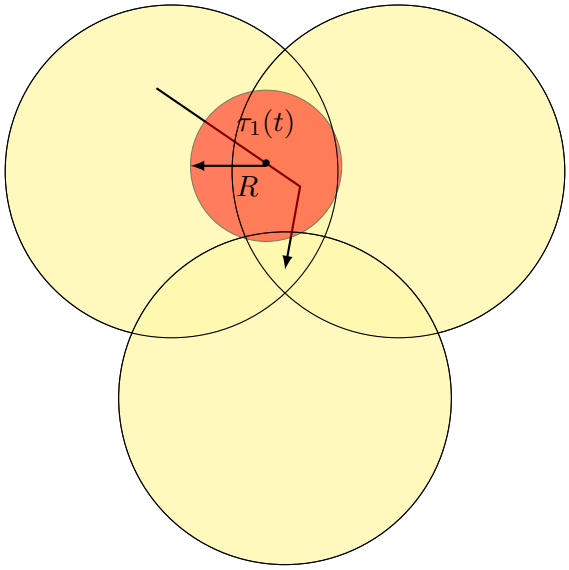

FIGURE Conc.1 – Exemple avec trois capteurs

Extrémité	Capteur	Valeur de R	Nature
e_1	1	10.8	entrant
e_2	1	4.8	entrant & sortant
e_2	2	8.8	entrant & sortant
e_3	1	2.2	sortant
e_3	2	2.2	sortant
e_3	3	5	sortant

Tableau Conc.1 – Liste des ticks pouvant apparaître dans la figure 1.2

chaque extrémité $e \in Q$, des ticks peuvent apparaître pour chaque capteur i à portée de e quand R est égal à v_e, la portée de i moins la distance entre e et i. Au début d'une trajectoire (respectivement, à la fin) il n'y a qu'un seul tick qui peut apparaître par capteur, un tick entrant (respectivement, sortant). Pour les autres extrémités, c'est-à-dire pour les changements de direction, il y a deux ticks par capteur à portée de l'extrémité, un tick entrant et un tick sortant. La liste des ticks pouvant apparaître dans l'exemple de la figure 1.2 est indiquée dans le tableau Conc.1, où e_1 est le début de la trajectoire, e_2 le changement de direction et e_3 la fin. Chaque ligne correspond à l'apparition de un ou deux ticks (un pour e_1 et e_3, deux sinon). La colonne *Valeur de* R donne la valeur de R pour laquelle ces ticks apparaissent. La colonne *Nature* indique s'il y a ajout d'un tick entrant, d'un tick sortant ou les deux.

Comme au premier chapitre, l'augmentation de la valeur de R provoque un déplacement des ticks. Ces déplacements sont maintenant décrits par des expressions algébriques impliquant la position des capteurs, les cordonnées des extrémités des segments des trajectoires

et la portée des capteurs. Ces formules permettent d'obtenir des positions sur les trajectoires estimées des cibles, auxquelles on fait correspondre des dates (induisant de nouvelles positions pour les ticks). Cependant, les formules pour le calcul des positions des ticks ne sont pas des expressions linéaires. Enfin, quand un tick entrant associé à un capteur i croise un tick sortant lui aussi associé à i alors ces deux ticks sont supprimés.

Pour conclure sur la discrétisation, avec une valeur R donnée, on est capable de déplacer, ajouter et supprimer des ticks. Les fenêtres de temps et les ensembles de capteurs candidats peuvent alors être déduits de ces ticks. L'ensemble des valeurs de R créant de nouvelles fenêtres de temps (apparition de ticks), ou modifiant les capteurs candidats (croisement de ticks) peut être calculé avec le problème initial.

La méthode de résolution proposée se rapproche de celle du chapitre 1. De la même manière, on ne peut pas résoudre un unique programme linéaire car les ensembles de capteurs candidats et les fenêtres de temps dépendent de la valeur de R. De plus, la taille d'une fenêtre de temps est calculée avec une formule non linéaire. Donc encore une fois, notre stratégie repose sur la résolution d'une séquence de programmes linéaires, mais cette fois-ci uniquement pour des valeurs de R données. Le programme linéaire est le même que le modèle 1.1. Une dichotomie comme dans le chapitre 1 est utilisée sur l'ensemble des valeurs de R créant, supprimant ou modifiant les fenêtres de temps. La dichotomie retourne un intervalle $[a, b[$ de R, où il existe un ordonnancement réalisable pour a mais pas pour b. Contrairement à l'incertitude temporelle, nous ne pouvons pas résoudre un unique et dernier programme linéaire où seule la taille des fenêtres de temps varie, car ces tailles ont une expression non linéaire en fonction de R. Nous optons donc pour une seconde dichotomie, qui continue de résoudre le même programme linéaire avec R donné sur l'intervalle continu $[a, b[$. La méthode itère jusqu'à ce que l'intervalle résultant soit suffisamment petit (taille inférieure à une précision ϵ donnée en paramètre). Ainsi, on obtient un ordonnancement réalisable pour une valeur donnée de R, que l'on sait être à une distance maximale de ϵ de la solution optimale.

Pour réduire le nombre d'itérations de la première dichotomie, nous proposons deux bornes supérieures qui réduisent le nombre de valeurs de R à tester. La première borne retire toutes les valeurs où une fenêtre de temps n'a aucun capteur candidat, ce qui correspond à un problème non réalisable. Pour cela, on doit lister tous les ajouts de tick ou intersections de ticks. Il suffit ensuite de tester si lors d'un tel évènement, la position estimée correspondante n'est plus couverte par aucun capteur. Le test est simple mais la complexité est de l'ordre de $|S| \times |I|^4$ où $|S|$ est le nombre de segments des trajectoires estimées et $|I|$ le nombre de capteurs. La deuxième borne vise à identifier les valeurs de R qui conduisent à un problème non réalisable faute d'énergie suffisante pour couvrir une fenêtre de temps. L'idée consiste à calculer l'énergie minimum requise pour la couverture de chaque fenêtre de temps initiale (obtenue pour $R = 0$), puis de faire évoluer l'énergie disponible avec la valeur de R. L'énergie nécessaire est simple-

ment égale à $(p^S + p^T) \times \Delta$, avec Δ la taille de la fenêtre. L'énergie disponible initialement est la somme des batteries des capteurs candidats. Pour chaque capteur, on calcule la valeur de R à partir de laquelle ce capteur n'est plus candidat à aucun instant de la fenêtre de temps. Pour cela, il suffit de calculer le point de la trajectoire estimée le plus éloigné du capteur, qui se trouve dans la fenêtre de temps. Ensuite, il reste à identifier la plus grande de ces valeurs, pour laquelle l'énergie disponible (somme des batteries des capteurs encore candidats dans cette face) est toujours supérieure à l'énergie nécessaire. Le principe de cette borne peut être appliqué à des découpages plus fins des fenêtres de temps, en les coupant pour chaque segment la constituant. La complexité de cette borne est de $|S| \times |I|^2$ avec $|S| \times |I|$ le nombre maximal de fenêtres de temps.

À noter qu'avant d'exécuter les deux dichotomies, on teste la faisabilité du problème ($R = 0$) et la faisabilité de la borne supérieure.

L'algorithme de discrétisation, le calcul des deux bornes supérieures ainsi que la méthode de résolution ont été implémentés en C++. Quelques essais préliminaires ont été conduits et les premiers résultats sont reportés dans le tableau Conc.2. Chaque ligne correspond à 40 instances réalisables (au moins un ordonnancement pour $R = 0$ est possible) avec un nombre de capteurs donné, et une précision $\epsilon = 10^{-4}$. Ces premiers résultats montrent que le temps de calcul de la méthode est acceptable mais dépend grandement du nombre de capteurs. Ces premiers essais nous ont aussi permis de constater l'importance des bornes supérieures qui peuvent s'avérer très efficace pour résoudre certaines instances. Avec 400 capteurs par exemple, il y a une grande différence de temps de calcul entre les instances où la borne supérieure est réalisable et celles où elle ne l'est pas. Il y a en effet des programmes linéaires plus difficiles à résoudre mais il y a aussi plus de valeurs à tester dans la première dichotomie.

#Capteurs	Temps CPU (s) moyen
50	0.013
100	0.066
200	0.870
400	7.911

Tableau Conc.2 – Résultats préliminaires de l'incertitude spatiale

Perspective avancée : robustesse spatio-temporelle

La perspective avancée présentée à la section précédente, ouvre la voie à une extension du problème d'ordonnancement robuste dans le cas où les incertitudes sont à la fois spatiales et temporelles. L'idée est de proposer un ordonnancement qui permet la couverture des cibles

et qui est robuste à des déviations spatiales, temporelles, ou les deux à la fois. Ses garanties de performances sont ainsi mesurées par un rayon de stabilité spatial R, et par un rayon de stabilité temporel ρ. Ainsi, la couverture d'une cible j à l'instant t est garantie, si elle se trouve dans un disque d'incertitude de rayon R et centrée sur $\tau_j(t+d)$, et que la valeur absolue de son avance ou de son retard d reste inférieure à ρ. Le reste du problème (un capteur par cellule à un instant t, limitation des batteries, envoi des données collectées à la station de base, ...) ne change pas.

Ce nouveau problème est bi-objectif par nature. Ces objectifs peuvent être ordonnés de manière lexicographique, ou non, ce qui impacte grandement la complexité.

Dans le cas d'un ordre lexicographique sur les objectifs, la résolution reste simple quel que soit l'ordre choisi. Il suffit d'appliquer les méthodes de résolution consécutivement sur le même ensemble de ticks. Par exemple, si R est maximisé avant ρ, alors on applique d'abord la méthode décrite dans la section précédente (problème de robustesse spatiale). Ensuite, on utilise la méthode du chapitre 1 pour maximiser ρ sur l'ensemble de ticks résultant de la valeur optimale de R. La seule particularité est l'ajout de ticks pour représenter les débuts et fins des trajectoires des cibles après la maximisation de R. On inverse l'ordre d'application des méthodes lorsque l'on maximise ρ puis R. Cependant, l'ajout d'un tick lors de la maximisation de R doit se faire en prenant en compte la valeur optimale de ρ. Un tick est alors ajouté avec un décalage temporel ($t \pm \rho$ avec $+\rho$ si le tick est entrant, et $-\rho$ sinon).

L'utilisation consécutive de ces deux méthodes permet l'optimisation lexicographique des deux objectifs, et l'on peut supposer que le temps de résolution devrait rester acceptable. Il s'agit en effet approximativement de l'addition des temps d'exécution des deux méthodes.

Avec deux objectifs de même importance, où un front de Pareto est recherché, le problème est plus difficile. Dans un premier temps, on peut appliquer une méthode de type ϵ-contrainte. C'est-à-dire que l'on résout plusieurs fois le problème dans un même ordre lexicographique. Ce faisant, on borne à chaque itération la valeur du premier objectif avec une valeur décroissante. La résolution reste donc la même que celle utilisée pour le problème avec ordre lexicographique, mais appliquée de nombreuses fois. Le temps de résolution est ainsi susceptible d'augmenter fortement. Pour le réduire, lors d'une itération, on peut utiliser la valeur optimale du second objectif trouvée dans l'itération précédente comme borne inférieure. Cette borne peut réduire fortement le nombre de valeurs à tester lors de l'optimisation du second objectif. On peut aussi remplacer la dichotomie par une simple méthode itérative partant de la borne inférieure. Pour résumer, la méthode de type ϵ-contrainte, dans chaque itération, résout le problème dans un ordre lexicographique pour obtenir une solution dont les valeurs des fonctions objectif serviront de bornes pour le calcul de la solution suivante. Par exemple, à l'itération $i > 1$, la solution a un rayon de stabilité spatial R_i tel que $R_i = R_{i-1} - \epsilon$ et un rayon de stabilité temporel optimal ρ_i que l'on calcule en utilisant $\rho_i \geq \rho_{i-1}$ (en testant des valeurs croissantes).

La méthode produit un ensemble de solutions faiblement efficaces pour lesquelles la fonction objectif R ne peut pas être améliorée de plus de ϵ. Ces solutions sont bien réparties entre les valeurs extrêmes des objectifs du front de Pareto, car elles ne sont pas concentrées autour de quelques valeurs de R ou ρ mais sont régulièrement réparties entre les bornes (supérieures et inférieures) du front de Pareto. Ainsi, avec une valeur de ϵ faible, les solutions obtenues représentent fidèlement le front de Pareto. Cette méthode a été implémentée et les expérimentations conduites montrent qu'elle est efficace.

Un exemple d'ensemble de solutions produites est présenté dans la figure Conc.2 avec un ϵ important. La ligne rouge est le front de Pareto, continu, que l'on ne connaît pas. Un ensemble de 8 solutions ($\{s_1, s_2, \ldots, s_8\}$) a été calculé tel que $\forall i \in \{1, .., 7\}, R_{s_{i+1}} = R_{s_i} - \epsilon$ où R_{s_i} est la valeur de la fonction objectif R pour la solution $i \in \{1, \ldots, 8\}$. La solution s_4 est dominée par s_3 et n'est donc pas gardée. Nous savons aussi que chaque autre solution peut être dominée par une solution du front de Pareto se trouvant sur la ligne en pointillés adjacente, dont la longueur est ϵ.

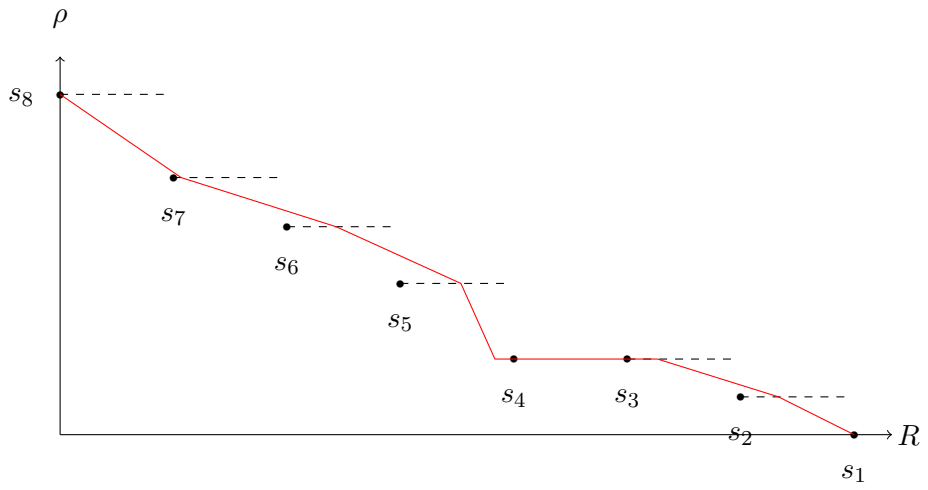

FIGURE Conc.2 – Illustration des solutions obtenues avec l'algorithme spatio-temporel

TABLE DES FIGURES

LISTE DES TABLEAUX

BIBLIOGRAPHIE

[1] Li Da XU, Eric L XU et Ling LI, « Industry 4.0 : state of the art and future trends », in : *International Journal of Production Research* 56.*8* (2018), p. 2941-2962.

[2] Jennifer YICK, Biswanath MUKHERJEE et Dipak GHOSAL, « Wireless sensor network survey », in : *Computer networks* 52.*12* (2008), p. 2292-2330.

[3] Wireless Sensor Networks MARKET, *Wireless Sensor Networks - Forecast to 2023*, `https://www.marketsandmarkets.com/Market-Reports/wireless-sensor-networks-market-445.html`, Accès : 13-03-2020, 2019.

[4] Bernard Osgood KOOPMAN, *Search and screening : general principles with historical applications*, Pergamon Press, 1980.

[5] Michael A GOODRICH, Bryan S MORSE, Damon GERHARDT, Joseph L COOPER, Morgan QUIGLEY, Julie A ADAMS et Curtis HUMPHREY, « Supporting wilderness search and rescue using a camera-equipped mini UAV », in : *Journal of Field Robotics* 25.*1-2* (2008), p. 89-110.

[6] Ian F AKYILDIZ, Weilian SU, Yogesh SANKARASUBRAMANIAM et Erdal CAYIRCI, « A survey on sensor networks », in : *IEEE Communications magazine* 40.*8* (2002), p. 102-114.

[7] Shaimaa M MOHAMED, Haitham S HAMZA et Iman Aly SAROIT, « Coverage in mobile wireless sensor networks (M-WSN) : A survey », in : *Computer Communications* 110 (2017), p. 133-150.

[8] Mohamed ELHOSENY, Alaa THARWAT, Xiaohui YUAN et Aboul Ella HASSANIEN, « Optimizing K-coverage of mobile WSNs », in : *Expert Systems with Applications* 92 (2018), p. 142-153.

[9] Yuzhen LIU et Weifa LIANG, « Approximate coverage in wireless sensor networks », in : *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, IEEE, 2005, p. 68-75.

[10] Matthias HANDY, Marc HAASE et Dirk TIMMERMANN, « Low energy adaptive clustering hierarchy with deterministic cluster-head selection », in : *Mobile and Wireless Communications Network, 2002. 4th International Workshop on*, IEEE, 2002, p. 368-372.

[11] Ossama YOUNIS et Sonia FAHMY, « HEED : a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks », in : *IEEE Transactions on mobile computing* 3.*4* (2004), p. 366-379.

[12] Francesco CARRABS, Raffaele CERULLI, Ciriaco D'AMBROSIO, Monica GENTILI et Andrea RAICONI, « Maximizing lifetime in wireless sensor networks with multiple sensor families », in : *Computers & operations research* 60 (2015), p. 121-137.

[13] Dipesh J PATEL, Rajan BATTA et Rakesh NAGI, « Clustering sensors in wireless ad hoc networks operating in a threat environment », in : *Operations Research* 53.*3* (2005), p. 432-442.

[14] Francesco CARRABS, Raffaele CERULLI, Ciriaco D'AMBROSIO et Andrea RAICONI, « Extending lifetime through partial coverage and roles allocation in connectivity-constrained sensor networks », in : *IFAC-PapersOnLine* 49.*12* (2016), p. 973-978.

[15] Francesco CARRABS, Raffaele CERULLI, Ciriaco D'AMBROSIO et Andrea RAICONI, « An exact algorithm to extend lifetime through roles allocation in sensor networks with connectivity constraints », in : *Optimization Letters* 11.*7* (2017), p. 1341-1356.

[16] Mihaela CARDEI et Ding-Zhu DU, « Improving wireless sensor network lifetime through power aware organization », in : *Wireless Networks* 11.*3* (2005), p. 333-340.

[17] Fabian CASTAÑO, André ROSSI, Marc SEVAUX et Nubia VELASCO, « A column generation approach to extend lifetime in wireless sensor networks with coverage and connectivity constraints », in : *Computers & Operations Research* 52 (2014), p. 220-230.

[18] Abolghasem ALIBEIKI, Homayun MOTAMENI et Hosein MOHAMADI, « A new genetic-based approach for maximizing network lifetime in directional sensor networks with adjustable sensing ranges », in : *Pervasive and Mobile Computing* 52 (2019), p. 1-12.

[19] Hsiang-Tsung KUNG et Dario VLAH, « Efficient location tracking using sensor networks », in : *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, t. 3, IEEE, 2003, p. 1954-1961.

[20] Marjan NADERAN, Mehdi DEHGHAN, Hossein PEDRAM et Vesal HAKAMI, « Survey of mobile object tracking protocols in wireless sensor networks : a network–centric perspective », in : *International Journal of Ad Hoc and Ubiquitous Computing* 11.*1* (2012), p. 34-63.

[21] Charly LERSTEAU, André ROSSI et Marc SEVAUX, « Robust scheduling of wireless sensor networks for target tracking under uncertainty », in : *European Journal of Operational Research* 252.*2* (2016), p. 407-417.

[22] Yanling JIN, Yongsheng DING, Kuangrong HAO et Yaochu JIN, « An endocrine-based intelligent distributed cooperative algorithm for target tracking in wireless sensor networks », in : *Soft computing* 19.*5* (2015), p. 1427-1441.

[23] Yuri N SOTSKOV, Alexandre DOLGUI et Marie-Claude PORTMANN, « Stability analysis of an optimal balance for an assembly line with fixed cycle time », in : *European Journal of Operational Research* 168.*3* (2006), p. 783-797.

[24] Stanley J BENKOSKI, Michael G MONTICINO et James R WEISINGER, « A survey of the search theory literature », in : *Naval Research Logistics (NRL)* 38.*4* (1991), p. 469-494.

[25] James M DOBBIE, « A survey of search theory », in : *Operations Research* 16.*3* (1968), p. 525-537.

[26] Shmuel GAL, « Search games », in : *Wiley Encyclopedia of Operations Research and Management Science* (2010).

[27] Ryusuke HOHZAKI, « Search games : Literature and survey », in : *Journal of the Operations Research Society of Japan* 59.*1* (2016), p. 1-34.

[28] Jess CURTIS et Robert MURPHEY, « Simultaneous area search and task assignment for a team of cooperative agents », in : *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003, p. 5584.

[29] Joel GEORGE, PB SUJIT et João B SOUSA, « Search strategies for multiple UAV search and destroy missions », in : *Journal of Intelligent & Robotic Systems* 61.*1-4* (2011), p. 355-367.

[30] Pablo LANILLOS, Seng Keat GAN, Eva BESADA-PORTAS, Gonzalo PAJARES et Salah SUKKARIEH, « Multi-UAV target search using decentralized gradient-based negotiation with expected observation », in : *Information Sciences* 282 (2014), p. 92-110.

[31] Madhubhashi SENANAYAKE, Ilankaikone SENTHOORAN, Jan Carlo BARCA, Hoam CHUNG, Joarder KAMRUZZAMAN et Manzur MURSHED, « Search and tracking algorithms for swarms of robots : A survey », in : *Robotics and Autonomous Systems* 75 (2016), p. 422-434.

[32] Antonio L ALFEO, Mario GCA CIMINO et Gigliola VAGLINI, « Enhancing biologically inspired swarm behavior : Metaheuristics to foster the optimization of UAVs coordination in target search », in : *Computers & Operations Research* 110 (2019), p. 34-47.

[33] Tüze KUYUCU, Ivan TANEV et Katsunori SHIMOHARA, « Superadditive effect of multi-robot coordination in the exploration of unknown environments via stigmergy », in : *Neurocomputing* 148 (2015), p. 83-90.

[34] James KENNEDY et Russell EBERHART, « Particle swarm optimization », in : *Proceedings of ICNN'95-International Conference on Neural Networks*, t. 4, IEEE, 1995, p. 1942-1948.

[35] John VON NEUMANN, Oskar MORGENSTERN et Harold William KUHN, *Theory of games and economic behavior (commemorative edition)*, Princeton university press, 2007.

[36] Shmuel GAL, « Search games with mobile and immobile hider », in : *SIAM Journal on Control and Optimization* 17.*1* (1979), p. 99-122.

[37] Tristan GARREC et Marco SCARSINI, « Search for an immobile hider on a stochastic network », in : *European Journal of Operational Research* 283.*2* (2020), p. 783-794.

[38] Steve ALPERN, « Search for an immobile Hider in a known subset of a network », in : *Theoretical Computer Science* 794 (2019), p. 20-26.

[39] Fedor V FOMIN et Dimitrios M THILIKOS, « An annotated bibliography on guaranteed graph searching », in : *Theoretical computer science* 399.*3* (2008), p. 236-245.

[40] Hoai An LE THI, Duc Manh NGUYEN et Tao Pham DINH, « A DC programming approach for planning a multisensor multizone search for a target », in : *Computers & operations research* 41 (2014), p. 231-239.

[41] Cécile SIMONIN, Jean-Pierre LE CADRE et Frédéric DAMBREVILLE, « A hierarchical approach for planning a multisensor multizone search for a moving target », in : *Computers & operations research* 36.*7* (2009), p. 2179-2192.

[42] Jacques DE GUENIN, « Optimum distribution of effort : An extension of the Koopman basic theory », in : *Operations Research* 9.*1* (1961), p. 1-7.

[43] Reuven Y RUBINSTEIN et Dirk P KROESE, *The cross-entropy method : a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*, Springer Science & Business Media, 2013.

[44] Pham Dinh TAO et Le Thi Hoai AN, « Convex analysis approach to DC programming : theory, algorithms and applications », in : *Acta mathematica vietnamica* 22.*1* (1997), p. 289-355.

[45] Scott Shorey BROWN, « Optimal search for a moving target in discrete time and space », in : *Operations research* 28.*6* (1980), p. 1275-1289.

[46] Lawrence D STONE, « Necessary and sufficient conditions for optimal search plans for moving targets », in : *Mathematics of Operations Research* 4.*4* (1979), p. 431-440.

[47] Ryusuke HOHZAKI et Koji IIDA, « A Concave Maximization Problem with Double Layers of Constraints on the total amount of resources », in : *Journal of the Operations Research Society of Japan* 43.*1* (2000), p. 109-127.

[48] Hoai An LE THI, Duc Manh NGUYEN et Tao Pham DINH, « A Based-DC Programming Approach for Planning a Multisensor Multizone Search for a Moving Target », in : *Modelling, Computation and Optimization in Information Systems and Management Sciences*, Springer, 2015, p. 107-118.

[49] X Rong LI, Ryan R PITRE, Vesselin P JILKOV et Huimin CHEN, « A new performance metric for search and track missions », in : *2009 12th International Conference on Information Fusion*, IEEE, 2009, p. 1100-1107.

[50] Ryan R PITRE, X Rong LI et Donald DELBALZO, « A new performance metric for search and track missions 2 : Design and application to UAV search », in : *2009 12th International Conference on Information Fusion*, IEEE, 2009, p. 1108-1114.

[51] Ryan R PITRE, X Rong LI et R DELBALZO, « UAV route planning for joint search and track missions—An information-value approach », in : *IEEE Transactions on Aerospace and Electronic Systems* 48.*3* (2012), p. 2551-2565.

[52] Wei MENG, Zhirong HE, Rong SU, Pradeep K YADAV, Rodney TEO et Lihua XIE, « Decentralized multi-UAV flight autonomy for moving convoys search and track », in : *IEEE Transactions on Control Systems Technology* 25.*4* (2016), p. 1480-1487.

[53] Luca BENINI, Giuliano CASTELLI, Alberto MACII, Enrico MACII, Massimo PONCINO et Riccardo SCARSI, « A discrete-time battery model for high-level power estimation », in : *Proceedings of the conference on Design, automation and test in Europe*, ACM, 2000, p. 35-41.

[54] Giuseppe ANASTASI, Marco CONTI, Mario DI FRANCESCO et Andrea PASSARELLA, « Energy conservation in wireless sensor networks : A survey », in : *Ad hoc networks* 7.*3* (2009), p. 537-568.

[55] Yuri N SOTSKOV, Vyacheslav S TANAEV et Frank WERNER, « Stability radius of an optimal schedule : A survey and recent developments », in : *Industrial applications of combinatorial optimization*, Springer, 1998, p. 72-108.

[56] Fabian CASTAÑO, Éric BOURREAU, André ROSSI, Marc SEVAUX et Nubia VELASCO, « Partial target coverage to extend the lifetime in wireless multi-role sensor networks », in : *Networks* 68.*1* (2016), p. 34-53, DOI : 10.1002/net.21682, URL : https://hal.archives-ouvertes.fr/hal-01328424.

[57] Malka N HALGAMUGE, Moshe ZUKERMAN, Kotagiri RAMAMOHANARAO et Hai L VU, « An estimation of sensor energy consumption », in : *Progress in Electromagnetics Research* 12 (2009), p. 259-295.

[58] Matthew J MILLER et Nitin H VAIDYA, « A MAC protocol to reduce sensor network energy consumption using a wakeup radio », in : *IEEE Transactions on mobile Computing* 4.*3* (2005), p. 228-242.

[59] Chi-Fu HUANG et Yu-Chee TSENG, « The coverage problem in a wireless sensor network », in : *Mobile Networks and Applications* 10.*4* (2005), p. 519-528.

[60] Charly LERSTEAU, André ROSSI et Marc SEVAUX, « Minimum energy target tracking with coverage guarantee in wireless sensor networks », in : *European Journal of Operational Research* 265.*3* (2018), p. 882-894.

[61] CUPCARBON, *CupCarbon*, http://www.cupcarbon.com/, Accès : 17-07-2019, 2014.

[62] Kamal MEHDI, Massinissa LOUNIS, Ahcène BOUNCEUR et Tahar KECHADI, « Cupcarbon : A multi-agent and discrete event wireless sensor network design and simulation tool », in : *7th International ICST Conference on Simulation Tools and Techniques, Lisbon, Portugal, 17-19 March 2014*, Institute for Computer Science, Social Informatics et Telecommunications Engineering (ICST), 2014, p. 126-131.

[63] Zuhal CAN et Murat DEMIRBAS, « A survey on in-network querying and tracking services for wireless sensor networks », in : *Ad Hoc Networks* 11.*1* (2013), p. 596-610.

[64] Bin LU et Vehbi C GUNGOR, « Online and remote motor energy monitoring and fault diagnostics using wireless sensor networks », in : *IEEE Transactions on Industrial Electronics* 56.*11* (2009), p. 4651-4659.

[65] Ravinder KAUR et Kamal Preet SINGH, « An efficient multipath dynamic routing protocol for mobile wsns », in : *Procedia Computer Science* 46 (2015), p. 1032-1040.

[66] Warren B POWELL, *Approximate Dynamic Programming : Solving the curses of dimensionality*, t. 703, John Wiley & Sons, 2007.

[67] Mandell BELLMORE et George L NEMHAUSER, « The traveling salesman problem : a survey », in : *Operations Research* 16.*3* (1968), p. 538-558.

[68] Clair E MILLER, Albert W TUCKER et Richard A ZEMLIN, « Integer programming formulation of traveling salesman problems », in : *Journal of the ACM (JACM)* 7.*4* (1960), p. 326-329.

[69] Gurobi OPTIMIZATION, *Inc.,"Gurobi optimizer reference manual," 2015*, 2014.

[70] Robert M CORLESS, Gaston H GONNET, David EG HARE, David J JEFFREY et Donald E KNUTH, « On the LambertW function », in : *Advances in Computational mathematics* 5.*1* (1996), p. 329-359.

[71] Thomas A FEO et Mauricio GC RESENDE, « Greedy randomized adaptive search procedures », in : *Journal of global optimization* 6.*2* (1995), p. 109-133.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

université angers

Titre : Optimisation d'ensembles de capteurs pour le suivi et la recherche de cibles

Mot clés : Recherche Opérationnelle ; Capteur ; Suivi de cibles ; Recherche de cible

Résumé : Dans cette thèse, nous abordons deux problèmes de suivi de cibles mobiles et un problème de recherche de cible à l'aide d'un réseau de capteurs. La première contribution est une extension des travaux précédents dans le domaine de la recherche d'un ordonnancement robuste permettant le suivi de cibles mobiles avec un réseau de capteurs sous incertitude temporelle. La nouvelle méthode proposée prend en compte le cas multi-cibles, le transfert des données et des considérations énergétiques. La seconde contribution est une méthode dynamique pour réagir aux déviations que l'ordonnancement robuste ne couvre pas. La troisième contribution de cette thèse relative au problème de recherche de cible, qui généralise un problème de planification de l'effort de recherche, en tenant compte de coûts de déplacement, ce qui donne lieu à un problème plus réaliste et plus général. La méthode de résolution proposée construit une solution à l'aide d'une approximation linéaire de la fonction objectif. Les contributions aux trois problèmes abordés dans cette thèse ont été testées sur de nombreuses instances.

Title : Optimization of sensor sets for target tracking and searching

Keywords : Operations research ; Sensor ; Target Tracking ; Target Search

Abstract : In this thesis, we address two target tracking problems and a search problem, using sensors. The first contribution is an extension of a previous work on target tracking, where the target trajectories are estimated under temporal uncertainty. A robust schedule for cases with more than one target, data transfer to a base station and energetic considerations is computed. A dynamic method is introduced as a second contribution to detect deviations and take online actions when the robust schedule is no longer feasible. It offers dynamic performance guarantees measured by a dynamic stability radius. The third contribution is related to the search problem, where an extension of the planning of the search effort is proposed, but where traveling costs are taken into account. The proposed approach builds a solution step-by-step using a linear approximation of the objective function. The contributions to the three problems addressed in this thesis are tested on numerous instances.