Formation dynamique d'équipes dans les DEC-POMDPS ouverts à base de méthodes Monte-Carlo

Jonathan Cohen

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité INFORMATIQUE

Préparée au sein de l'Université de Caen Normandie

Formation dynamique d'équipes dans les DEC-POMDPS ouverts à base de méthodes Monte-Carlo

Présentée et soutenue par
Jonathan COHEN

Thèse soutenue publiquement le 13/06/2019 devant le jury composé de		
M. FRANÇOIS CHARPILLET	Directeur de recherche à l'INRIA, Université Nancy 1 Henri Poincaré	Rapporteur du jury
M. FRÉDÉRIC KORICHE	Professeur des universités, Université d'Artois	Rapporteur du jury
Mme AURÉLIE BEYNIER	Maître de conférences, Université Paris 6 Pierre et Marie Curie	Membre du jury
M. JILLES STEEVE DIBANGOYE	Maître de conférences, INSA Lyon	Membre du jury
M. NICOLAS MAUDET	Professeur des universités, Université Paris 6 Pierre et Marie Curie	Président du jury
M. REGIS SABBADIN	Directeur de recherche à l'INRA, INRA de Toulouse	Membre du jury
M. ABDEL ILLAH MOUADDIB	Professeur des universités, Université Caen Normandie	Directeur de thèse

Thèse dirigée par ABDEL ILLAH MOUADDIB, Groupe de recherche en informatique, image, automatique et instrumentation

UNIVERSITÉ CAEN NORMANDIE

MIIS

GREYC

« Mon truc, William Potter, c'est de ne pas faire attention quand ça fait mal. »

Peter O'Toole
Lawrence d'Arabie

Merci à Safwane et Simon.

Cette thèse a été financée par la Région Normandie et l'Union Européenne dans le cadre du programme opérationnel FEDER-FSE 2014-2020. Ces travaux font partie du projet GARDES (ANR-14-ASTR-0018), collaboration entre le laboratoire GREYC de l'Université de Caen Normandie, la Direction générale de l'Armement et Nexter Robotics.

Résumé

Cette thèse traite du problème où une équipe d'agents coopératifs et autonomes, évoluant dans un environnement stochastique partiellement observable, et œuvrant à la résolution d'une tâche complexe, doit modifier dynamiquement sa composition durant l'exécution de la tâche afin de s'adapter à l'évolution de celle-ci. Il s'agit d'un problème qui n'a été que peu étudié dans le domaine de la planification multi-agents. Pourtant, il existe de nombreuses situations où l'équipe d'agent mobilisée est amenée à changer au fil de l'exécution de la tâche. Nous nous intéressons plus particulièrement au cas où les agents peuvent décider d'eux-même de quitter ou de rejoindre l'équipe opérationnelle. Certaines fois, utiliser peu d'agents peut être bénéfique si les coûts induits par l'utilisation des agents sont trop prohibitifs. Inversement, il peut parfois être utile de faire appel à plus d'agents si la situation empire et que les compétences de certains agents se révèlent être de précieux atouts. Afin de proposer un modèle de décision qui permette de représenter ces situations, nous nous basons sur les processus décisionnels de Markov décentralisés et partiellement observables, un modèle standard utilisé dans le cadre de la planification multi-agents sous incertitude. Nous étendons ce modèle afin de permettre aux agents d'entrer et sortir du système. On parle alors de système ouvert. Nous présentons également deux algorithmes de résolution basés sur les populaires méthodes de recherche arborescente Monte-Carlo. Le premier de ces algorithmes nous permet de construire des politiques jointes séparables via des calculs de meilleures réponses successives, tandis que le second construit des politiques jointes non séparables en évaluant les équipes dans chaque situation via un système de classement Elo. Nous évaluons nos méthodes sur de nouveaux jeux de tests qui permettent de mettre en lumière les caractéristiques des systèmes ouverts.

Table des matières

Résumé **v**

I Introduction **3**

1 Introduction **5**

1.1 Contexte 5

1.2 Thèse 7

1.3 Sommaire 7

II État de l'art **9**

2 Définitions générales **11**

2.1 Concepts de base 12

2.1.1 Agents 12

2.1.2 Rationalité 14

2.1.3 Environnement 14

2.2 Planification 15

2.2.1 Mesure de performance 15

2.2.2 Résolution 16

2.3 Conclusion 18

3 Planification sous incertitude **19**

3.1 Introduction 20

3.2 Processus décisionnels de Markov 20

3.2.1 Définition du modèle 20

3.2.2 Exemple 22

3.2.3 Politiques 24

3.2.4 Principe d'optimalité 25

3.3 Observabilité partielle 28

3.3.1 Définition du modèle . . . 29
3.3.2 État de croyance . . . 32
3.3.3 Belief MDP . . . 34
3.3.4 Complexité . . . 37
3.4 Contrôle décentralisé . . . 38
3.4.1 Définition du modèle . . . 38
3.4.2 Historiques et politiques . . . 40
3.4.3 Complexité . . . 43
3.4.4 Méthodes de résolution . . . 44
3.5 Conclusion . . . 46

4 Équipes flexibles **47**
4.1 Introduction . . . 48
4.2 Allocation de rôles . . . 49
4.2.1 Problème d'affectation . . . 50
4.2.2 Modèle de décision pour l'allocation de rôles . . . 52
4.2.3 Stratégies de ré-allocation de rôles . . . 54
4.3 Formation de coalitions . . . 55
4.3.1 Définitions . . . 56
4.3.2 Propriétés . . . 58
4.4 Contrôle partiel de systèmes multi-agents ouverts . . . 59
4.4.1 Travail d'équipe ad-hoc . . . 60
4.4.2 Agents non dévoués . . . 61
4.5 Conclusion . . . 62

III Modèles **63**

5 POMDPs décentralisés & ouverts **65**
5.1 Introduction . . . 66
5.2 Vers un DEC-POMDP ouvert . . . 67
5.2.1 Le problème TARS . . . 67
5.2.2 Analyse de benchmarks . . . 68
5.3 DEC-POMDPs ouverts . . . 69
5.3.1 Définition du modèle . . . 70
5.3.2 Niveaux de contrôle du processus d'entrées et sorties . . . 71
5.3.3 Coûts des transitions d'équipes . . . 73
5.3.4 Historiques, politiques et résolution . . . 73

5.3.5 Avantages et inconvénients . . . 74
5.4 L'algorithme BRD . . . 74
5.4.1 Équilibre de Nash . . . 74
5.4.2 Best Response Dynamics . . . 75
5.4.3 Sélection des agents . . . 76
5.5 L'algorithme POS-UCT . . . 77
5.5.1 Présentation . . . 78
5.5.2 Recherche arborescente Monte-Carlo dans les domaines partiellement observables . . . 78
5.5.3 Historiques rares . . . 84
5.6 Conclusion . . . 85

6 Éléments de théorie des jeux pour le problème de formation dynamique d'équipes sous incertitude 87

6.1 Introduction . . . 88
6.2 POMDP d'équipes . . . 89
6.2.1 Définition du modèle . . . 89
6.2.2 Historiques et politiques partiels . . . 90
6.3 Propriétés structurelles . . . 92
6.3.1 Concepts de base . . . 92
6.3.2 Monotonie, suradditivité et convexité . . . 95
6.3.3 Illustrations . . . 97
6.3.4 Complexité algorithmique . . . 98
6.4 Indices de pouvoir . . . 99
6.4.1 Utilité différentielle . . . 100
6.4.2 Le système de classement Elo . . . 102
6.4.3 Matchs . . . 103
6.4.4 Volatilité et stabilité . . . 104
6.5 L'algorithme MELO . . . 104
6.5.1 Recherche arborescente Monte-Carlo avec classement Elo . . . 105
6.5.2 Matchs des meilleures équipes . . . 109
6.5.3 Justification des classements Elo . . . 109
6.5.4 Séparabilité . . . 110
6.6 Conclusion . . . 112

IV Évaluation expérimentale **113**

7 Benchmarks **115**
7.1 Introduction . . . 116
7.2 Le problème de la zone sinistrée . . . 116
7.2.1 Description . . . 116
7.2.2 Modèle . . . 117
7.2.3 Objectifs . . . 119
7.3 Le problème de l'entrepôt de stockage . . . 119
7.3.1 Description . . . 119
7.3.2 Modèle . . . 120
7.3.3 Objectifs . . . 121
7.4 Conclusion . . . 121

8 Résultats **123**
8.1 Introduction . . . 124
8.2 Évaluation de l'algorithme BRD-POS-UCT . . . 125
8.2.1 Comparaison avec les environnements fermés . . . 126
8.2.2 Gestion de la taille de l'équipe opérationnelle . . . 127
8.2.3 Influence de l'état de présence . . . 128
8.3 Évaluation de l'algorithme MELO . . . 128
8.3.1 Mise en évidence de la sous-modularité . . . 129
8.3.2 Valeurs non-monotones . . . 131
8.3.3 Pertinence des classements Elo . . . 133
8.3.4 Stratégies de descente . . . 138
8.3.5 Définitions de nœuds extensibles . . . 139
8.3.6 Matchs des meilleures équipes . . . 141
8.4 Conclusion . . . 141

V Conclusion **143**

9 Conclusion **145**
9.1 Bilan . . . 145
9.2 Perspectives . . . 146

Bibliographie **149**

Première partie

Introduction

Chapitre 1

Introduction

> « J'ai une politique très stricte de contrôle des armes à feu : s'il y a une arme à feu, je veux en avoir le contrôle. »
>
> Clint Eastwood

Capital City, États-Unis. Lorsque Morpheus lui propose de choisir entre la pilule bleue et la pilule rouge, Thomas A. Anderson, humble programmeur Américain dans une respectable entreprise de logiciels informatiques, sait qu'il se trouve face à la décision la plus importante de son existence. S'il choisit la pilule bleue, il oublie tout et se réveille chez lui, prêt à reprendre le cours de sa vie monotone, ignorant de la nature réelle des choses. S'il choisit la pilule rouge, alors il s'apprêtera à découvrir que le monde dans lequel il vit n'est en réalité pas tout à fait ce qu'il pensait.

1.1 Contexte

Si *Matrix* [Wachowskis, 1999] restera à jamais une œuvre marquante dans l'histoire du septième art, c'est notamment parce que le film comporte plusieurs scènes devenues cultes. Parmi celles-ci, la scène du choix entre la pilule bleue et la pilule rouge constitue l'une des séquences les plus mémorables du cinéma. Le choix fait par le héros à cet instant est crucial puisque derrière son apparence si simple vont découler des effets à long terme d'une ampleur considérable. La théorie de la décision [Binmore, 2008] est le domaine scientifique qui vise à décrire et à optimiser la capacité d'un agent à prendre ce genre de décisions où les conséquences futures doivent être prises en considération. Cette théorie se découpe en deux branches. D'une part, la théorie de la décision descriptive, qui vise à analyser pourquoi et comment les décisions sont prises. Autrement dit, cela revient à se demander *pourquoi* M. Anderson a choisi la pilule rouge. D'autre part, la théorie de la décision normative, qui s'occupe de déterminer les meilleures décisions à prendre compte tenu des informations imparfaites et incomplètes dont on dispose. Cette fois, cela revient à se demander quelle pilule M. Anderson *aurait dû* prendre. C'est de cette branche dont il sera question dans cette thèse.

Savoir prendre les bonnes décisions en analysant les conséquences futures de ses choix est un processus qui intervient quotidiennement dans de nombreuses situations : de l'astronaute qui doit être capable de réagir en quelques secondes lorsqu'une série d'alarmes se déclenche à bord de sa navette spatiale, à l'entraîneur sportif qui, depuis le bord du terrain, coordonne ses joueurs et leur donne continuellement les directives à suivre, en passant par la voiture autonome qui, percevant ce qui semble être un obstacle, doit décider de freiner brusquement ou non. Avec

l'automatisation croissante des technologies dans de nombreux secteurs de l'industrie, est venue la nécessité de concevoir des systèmes fiables, sur lesquels on peut exercer un contrôle optimal.

Malheureusement, au moins deux types de difficultés se présentent lorsqu'il s'agit de concevoir des systèmes capables de prises de décision autonomes. La première d'entre elles concerne l'incertitude liée au milieu dans lequel le système est implémenté. Imaginons une application militaire, où une flotte de drones autonomes est déployée sur un site sensible (par exemple, une zone de stockage d'armement nucléaire) dont elle doit en assurer la sécurité en signalant, le cas échéant, la présence d'unités ennemies. Les drones, équipés du matériel de défense nécessaire à la réussite de leur mission (caméras, radars, radios), survolent un périmètre, effectuent des rondes, envoient des messages d'alerte à un opérateur humain distant lorsqu'ils pensent avoir identifié une menace, ou décident de l'envoi d'un missile d'interception sol-air. Faire la différence entre un aéronef ennemi en approche du site surveillé et un aéronef qui ne représente pas une menace n'est pas toujours une tâche facile, surtout si l'aéronef ennemi tente délibérément de masquer sa présence et son identité. Suivant son altitude, ses messages radio ou encore sa vitesse de déplacement, il se peut que l'aéronef ennemi ne soit pas détecté en tant que tel par l'équipe de drones, pouvant ainsi mener à de graves brèches de sécurité. Inversement, attaquer un aéronef allié n'est évidemment pas souhaitable. L'incertitude induite par l'inexactitude des capteurs des drones peut également être un facteur rendant difficile la tâche de surveillance. Ces différentes formes d'incertitude sur l'environnement peuvent mener la flotte de drones à prendre des décisions non-optimales (signalements de faux positifs ou de faux négatifs).

La seconde difficulté qui émerge lorsque l'on conçoit un système autonome composé de plusieurs unités de calculs (comme c'est le cas dans cet exemple de drones de surveillance), est de réussir à coordonner efficacement les unités (ou agents) pour que le comportement global de l'équipe soit optimal. Pour des raisons de sécurité, les communications entre les drones peuvent être réduites au minimum, voire complètement interdites. Les drones doivent ainsi s'organiser pour ne pas que l'un empiète sur la zone de surveillance de l'autre, mais également pour qu'il n'y ait pas de zone hors de surveillance à un instant donné. En l'absence de communications, cette synchronisation doit avoir lieu en amont de l'exécution de la mission, de telle sorte que chaque agent sache quoi faire sans avoir besoin de savoir ce que font les autres agents au même moment. De plus, planifier pour un grand nombre d'agents devient vite un problème théorique complexe, puisque le nombre de stratégies possibles pour l'équipe sur le terrain va augmenter exponentiellement avec le nombre d'agents.

Les processus décisionnels de Markov décentralisés partiellement observables (DEC-POMDP) dont il est question dans le titre de cette thèse constituent un modèle mathématique formel visant à planifier le comportement optimal d'une équipe d'agents coopératifs qui travaillent ensemble à la résolution d'une tâche dans un environnement complexe, dynamique et incertain. On parle de planification multi-agents sous incertitude. Il s'agit d'un domaine qui a été beaucoup étudié par la recherche en intelligence artificielle, et les DEC-POMDPs constituent le standard *de facto* lorsqu'il s'agit de résoudre les problèmes de prise de décisions séquentielles décentralisées dans l'incertain. C'est le modèle que nous allons utiliser.

Toutefois, cette thèse ne cherche pas à proposer de nouvelles méthodes ou de nouveaux algorithmes pour faire de la planification multi-agents sous incertitude. Nous nous attaquons plutôt à une autre question qui, quant à elle, n'a encore été que très peu abordée par la recherche scientifique : comment planifier pour des équipes *ouvertes*, c'est-à-dire où des agents peuvent quitter ou rejoindre l'équipe en place ? Lorsqu'un drone de surveillance tombe en panne (suite à une attaque ou simplement car sa batterie est épuisée), ou lorsqu'il est ré-affecté sur une autre mission, il n'est plus en mesure d'assurer sa tâche de surveillance. Les autres drones de l'équipe doivent être prêts à se réorganiser pour continuer d'assurer la sécurité du site surveillé.

1.2 Thèse

Comment permettre à une équipe d'agents coopératifs autonomes de se reformer afin de s'adapter à l'évolution de la situation ?

L'objectif principal de cette thèse est de répondre à cette question. D'abord, en introduisant le problème de la planification pour les **DEC-POMDPs ouverts**, où le nombre des agents mobilisés est amené à varier au cours de l'exécution de la mission. Puis, en proposant les premières solutions pour résoudre ce problème. Nous le verrons, il existe plusieurs modalités qui font que les agents peuvent entrer ou sortir de l'équipe d'agents actuellement en mission. Comme évoqué ci-dessus, un agent peut tomber en panne et se retrouver indisponible. Ou bien, l'évolution de la situation fait que d'autres agents sont nécessaires à la résolution de la tâche en cours. Inversement, suivant la configuration actuelle de la mission, il se peut qu'avoir trop d'agents mobilisés entraîne des contre-performances. Dans ce cas-là, certains agents vont devoir quitter l'équipe. Notre but est d'établir un modèle qui permette de représenter toutes ces situations différentes, et d'y apporter des solutions. Pour y parvenir, nous introduisons plusieurs contributions.

Le premier axe de ces contributions vise à décrire le modèle d'OPEN-DEC-POMDP, qui permet d'étendre le modèle de DEC-POMDP aux situations où l'équipe d'agents engagée dans la résolution de la tâche est amenée à évoluer. Dans un second temps, nous proposons le modèle de TEAM-POMDP, qui est une restriction des OPEN-DEC-POMDPs aux cas où les entrées et sorties sont gardées sous contrôle. Dans ce nouveau modèle, on ne considère donc plus qu'un agent peut tomber en panne ou quitter l'équipe durant l'exécution de la tâche sans que cela ne soit planifié à l'avance : s'il y a un agent dans le système, alors il est totalement contrôlé et son comportement planifié en amont de l'exécution de la mission. Cette thèse se concentre donc sur la **formation dynamique d'équipes**. Notre objectif va être de trouver l'équipe optimale à employer selon la situation et, pour chaque agent de cette équipe, de trouver le comportement optimal qu'il doit suivre. Une part importante de nos contributions vise à identifier certaines sous-classes de TEAM-POMDPs qui respectent certaines propriétés habituellement rencontrées dans le domaine de la théorie des jeux coopératifs. De surcroît, on identifie la classe de complexité de notre modèle de décision comme étant la classe des problèmes NEXP-complets.

Le second axe de nos contributions concerne la résolution de ces modèles par le biais de méthodes de recherche arborescente dites de **Monte-Carlo** [Metropolis and Ulam, 1949]. Le principe des méthodes de Monte-Carlo consiste à estimer la valeur d'une variable numérique par une suite d'échantillonnages aléatoires. Ces méthodes ont été largement étudiées, autant dans le domaine de la recherche fondamentale en mathématiques que dans le domaine de l'informatique appliquée. L'un des algorithmes ayant rencontré le plus de succès est l'algorithme MCTS (*Monte Carlo Tree Search*). Dans nos recherches, nous étendons cet algorithme pour faire de la planification dans les milieux ouverts, stochastiques et partiellement observables – c'est-à-dire dans un contexte adapté à notre problématique. Nous proposons les algorithmes BRD-POS-UCT et MELO, tous deux basés sur MCTS, et qui ont chacun leurs forces et faiblesses. Ces deux algorithmes permettent d'apporter des solutions approximatives à notre problématique en évitant la complexité de résolution habituellement rencontrée par les algorithmes optimaux.

1.3 Sommaire

Ce document est composé de trois parties principales. Dans la partie II, nous replaçons notre problème dans le contexte des processus décisionnels de Markov et nous examinons quelques approches existantes concernant la prise de décision dans les systèmes multi-agents ouverts. La partie III présente nos contributions : modèles, analyses et algorithmes pour faire de la formation

dynamique d'équipes. Enfin, la partie IV décrit les expériences que nous avons menées pour valider nos approches.

Partie II : État de l'art

Cette partie permet de replacer nos travaux dans leur contexte et d'aborder les différentes solutions qui existent déjà pour résoudre le problème que l'on se pose.

- **Chapitre 2.** Ce chapitre introduit la thèse en présentant les concepts de base : agent, environnement et problème de planification. Nous reviendrons sur certaines des notions qui ont été énoncées dans cette introduction.
- **Chapitre 3.** Ce chapitre donne les définitions et notions théoriques nécessaires à la compréhension de cette thèse. Sont abordés les modèles de processus décisionnels de Markov, partiellement observables, et décentralisés. Nous donnerons également les définitions d'historiques d'observations et de politiques qui permettent aux agents de prendre des décisions.
- **Chapitre 4.** La partie II se conclue avec ce chapitre, qui présente un examen des méthodes déjà existantes qui répondent partiellement à notre problématique. Nous y étudierons l'allocation et la ré-allocation de rôles dans les systèmes multi-agents, la formation de coalitions dans les jeux coopératifs, puis nous aborderons brièvement d'autres concepts connexes aux équipes d'agents ouvertes.

Partie III : Modèles

Cette partie correspond à nos contributions principales.

- **Chapitre 5.** Pour ce premier chapitre de contributions, nous définissons formellement le problème de formation dynamique d'équipes et de planification sous incertitude. Nous décrivons ensuite notre modèle théorique pour faire de la planification dans les systèmes multi-agents ouverts. Enfin, nous présentons notre premier algorithme de résolution, nommé BRD-POS-UCT, basé sur une méthode de recherche arborescente Monte-Carlo et le calcul de politiques de meilleures réponses.
- **Chapitre 6.** Dans ce second chapitre, nous présentons une nouvelle version du modèle décisionnel introduit dans le chapitre 5. Après avoir donné quelques définitions et propriétés théoriques inspirées de la théorie des jeux, nous décrivons un nouvel algorithme de résolution, nommé MELO, qui exploite la structure de notre modèle et maintient un classement Elo des équipes afin de trouver rapidement celles qui sont efficaces dans les différentes situations possibles de la tâche à résoudre.

Partie IV : Évaluation expérimentale

Enfin, cette troisième partie permet d'évaluer expérimentalement les idées présentées dans cette thèse.

- **Chapitre 7.** Nous introduisons dans ce chapitre deux jeux de tests pour évaluer nos algorithmes et mettre en lumière certaines propriétés de notre modèle de décision. L'un prend place dans un environnement dynamique et complexe, l'autre dans un environnement statique, plus simple, où nous pourrons considérer un plus grand nombre d'agents.
- **Chapitre 8.** Enfin, le dernier chapitre de ces trois parties principales présente quelques résultats que nous avons obtenus pour évaluer la pertinence et l'efficacité de nos approches.

Nous concluons dans la **Partie V** en résumant les principales contributions de nos recherches, et donnons quelques perspectives pour de futurs travaux.

Deuxième partie

État de l'art

Chapitre 2

Définitions générales

Sommaire

2.1 Concepts de base 12
2.1.1 Agents 12
2.1.2 Rationalité 14
2.1.3 Environnement 14
2.2 Planification 15
2.2.1 Mesure de performance 15
2.2.2 Résolution 16
2.3 Conclusion 18

« Nous ne pourrons pas tout faire dans les cent premiers jours. Ni dans les mille premiers jours, ni pendant toute la durée de notre mandat, ni même peut-être pendant toute notre vie sur cette planète. Mais commençons ! »

John Fitzgerald Kennedy
20 janvier 1961

2.1 Concepts de base

Le principal objectif de cette thèse consiste à construire un modèle pour planifier les entrées et sorties d'agents attelés à la résolution d'une tâche. On peut voir ce modèle comme un coach de basket-ball. Depuis le bord du terrain, il coordonne son équipe, donne les directives à suivre et, chose qui nous intéresse tout particulièrement, s'occupe du *coaching*, c'est-à-dire, décide quel joueur doit entrer sur le parquet et quel autre doit sortir à tel ou tel moment du match. Cette question, souvent ignorée, comme nous allons le voir, dans le domaine de la planification multi-agents, revêt une importance primordiale puisque l'équipe en place doit s'adapter à la situation pour pouvoir être efficace. Imaginons que l'un des joueurs performe en-deçà de ses capacités car il joue contre un adversaire qui parvient à le contenir efficacement. Le coach doit intervenir et procéder soit à un changement de position, soit à un changement de joueur s'il veut optimiser les chances de victoire de son équipe.

Mais avant d'aborder le problème des entrées et sorties d'agents dans un système, il nous faut tout d'abord commencer avec certaines notions essentielles à la compréhension de cette thèse. En premier lieu, qu'est-ce qu'un *agent* ? Ensuite, qu'est-ce-que ce *système* dans lequel cet agent évolue ? Après avoir donné ces définitions primordiales à l'étude de notre sujet, nous aborderons la question de la planification, et la place qu'elle occupe dans le domaine de l'intelligence artificielle.

2.1.1 Agents

Un **agent** est une entité qui interagit avec son environnement [Russell and Norvig, 2009]. L'interaction se fait de deux façons : d'une part, l'agent reçoit des informations de l'environnement. Ce sont ce que l'on appelle des **perceptions**, ou **observations**. Il les reçoit via ses **capteurs**. D'autre part, l'agent agit sur l'environnement par le biais d'**actions**. Il utilise pour cela ses **actionneurs**. Les actions que l'agent choisit d'effectuer sont dépendantes de ses perceptions et de son modèle de raisonnement interne. Ce processus simple est illustré sur la figure 2.1.

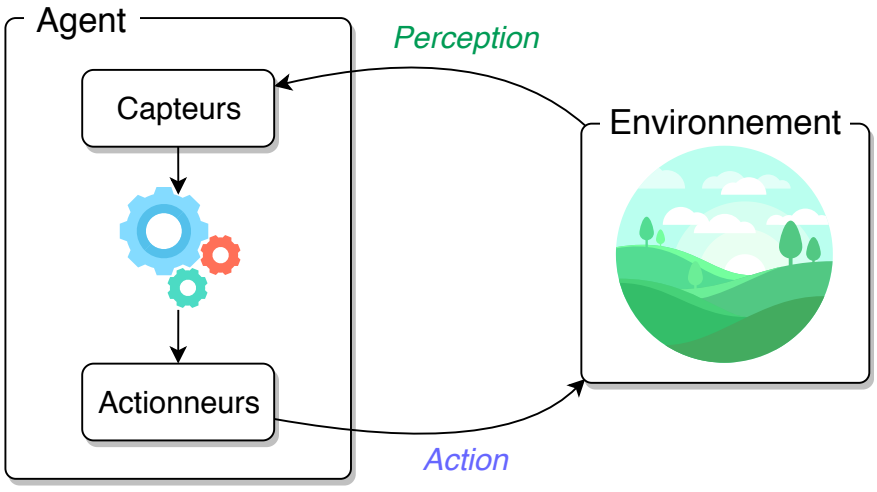


FIGURE 2.1 – Processus de décision d'un agent. L'agent perçoit son environnement et prend des décisions basées sur ses perceptions et ses capacités de raisonnement.

Les exemples d'agents sont nombreux. Un joueur de basket-ball est un agent. Il reçoit des

informations sur ce qu'il se passe sur le parquet via des signaux lumineux, sonores ou tactiles perçus par ses yeux, ses oreilles ou sa peau, et il agit sur son environnement avec ses mains, ses jambes ou sa voix pour effectuer une passe, une feinte, un appel de ballon, etc. Un drone de surveillance est également un agent. Il peut disposer d'une caméra noir et blanc et d'un laser pour effectuer des observations, et il peut agir sur son environnement en émettant des signaux radiophoniques pour communiquer avec sa centrale de sécurité. Une entreprise commerciale aussi est un agent. Elle observe les prix et les produits du marché ainsi que ses compétiteurs, et décide d'actions commerciales, de lancements de produits, d'une éventuelle entrée en bourse, etc., modifiant ainsi l'écosystème économique. Dans le domaine de la recherche en informatique, qui nous intéresse plus particulièrement ici, le terme d'agent permet de désigner de façon générique une unité dotée d'un certain degré d'autonomie et de capacités de calculs, d'une forme de raisonnement.

Un **système multi-agents** [Wooldridge, 2009] est une extension directe aux cas où plusieurs agents peuplent le système. Chaque agent est une unité indépendante qui dispose de ses propres capteurs et actionneurs, et qui prend donc ses propres décisions suivant un raisonnement interne. Au sein d'un système multi-agents, les agents peuvent être de différents types. Des agents **coopératifs** agissent dans un but commun. Ils cherchent à résoudre la même tâche de la meilleure façon possible. Par exemple, on peut représenter une équipe sportive comme un système multi-agents coopératif dont le but est de remporter le match. Des agents **compétitifs** en revanche ont des buts distincts et, le plus souvent, contradictoires. On peut également rencontrer des systèmes hybrides, où certains agents sont coopératifs entre eux mais en compétition avec d'autres. La modélisation d'une situation par un système multi-agents dépend à la fois du domaine étudié et du point de vue adopté. Par exemple, cinq joueurs des *Chicago Bulls* peuvent être vus comme cinq agents coopératifs, mais du point de vue d'un *match* entre les *Chicago Bulls* et les *Los Angeles Lakers*, il est nécessaire de considérer les cinq joueurs des *Lakers* comme des agents appartenant également au système, faisant ainsi de ce dernier un système hybride. Si l'on adopte un regard encore différent sur cette même situation, on peut aussi voir le match entre les *Bulls* et les *Lakers* comme une compétition entre deux agents, où chaque équipe, et non pas chaque joueur, est un agent.

Un système multi-agents **ouvert** [Nodine and Unruh, 1997, Eijk et al., 1999] est un système où les agents peuvent entrer et sortir à leur gré, ou sous certaines conditions. S'il n'est pas ouvert, il est dit **fermé**. Les agents dans un système multi-agents ouvert doivent tenir compte du fait que d'autres agents (coopératifs ou compétitifs) peuvent entrer ou sortir à tout moment. Ils doivent donc disposer de moyens leur permettant de réagir à de telles entrées/sorties. Par exemple, et parlons football plutôt que basket-ball cette fois, si un joueur se blesse en plein match, il va être soigné sur le bord du terrain en attendant de savoir s'il pourra retourner jouer ou s'il devra être remplacé. Pendant son absence, son équipe devra évoluer à dix contre onze et probablement adopter une stratégie de jeu plus défensive. S'il advient que le joueur peut regagner le terrain, alors son équipe pourra retrouver son style de jeu antérieur. S'il se trouve toutefois qu'un nouveau joueur doit le remplacer, alors il faudra peut-être que le schéma tactique de l'équipe change pour s'adapter aux spécificités du joueur remplaçant. Toutefois, la sortie ou l'entrée d'un agent n'est pas toujours contrainte, elle peut également être choisie : un pompier dont les ressources en eau sont épuisées a tout intérêt à quitter momentanément le système pour recharger ses réserves et revenir combattre le feu plus tard.

Cette thèse se concentre sur les systèmes multi-agents coopératifs ouverts. L'intérêt d'employer de tels systèmes est multiple. D'une part, il est possible de profiter des caractéristiques uniques de chaque agent pour leur permettre de résoudre chacun une partie de la tâche globale. D'autre part, le système étant distribué, celui-ci est plus robuste aux pannes. Si un agent ve-

nait à être hors-service ou indisponible, les autres agents pourraient tout de même continuer à travailler à la résolution de la tâche.

2.1.2 Rationalité

Un agent est **rationnel** [Russell and Norvig, 2009] si les actions qu'il effectue sont toujours les meilleures compte tenu de l'information dont il dispose, c'est-à-dire des observations qu'il fait. Il va de soi que tous les agents ne sont pas rationnels. Cela dépend de la situation, du problème étudié, voire du domaine de recherche dans lequel on se place. Par exemple, un service municipal qui voudrait étudier le comportement des automobilistes de la ville au moyen d'un simulateur afin d'optimiser les coûts des aires de stationnement pourrait le faire en modélisant les automobilistes comme des agents [Pageaud et al., 2017]. Ils le sont après tout : au sein du simulateur, un automobiliste observe l'état et le prix des parkings, estime sa distance à son lieu de travail, et finalement choisit une place où se garer. Néanmoins, dans une bonne simulation, les automobilistes ne sont pas tous rationnels. Ils peuvent choisir une place selon des critères non-objectifs, qui par exemple ne minimiseraient pas leurs dépenses, et préférer un parking un peu plus cher, plus éloigné mais dans un quartier plus sûr. Les notions de *maximisation*, de *minimisation* et de *critères objectifs* doivent néanmoins être formellement définies. Peut-être que selon ses critères personnels, un automobiliste préférera se garer dans un quartier tranquille plutôt que sur un parking du centre-ville qui serait plus proche de son lieu de travail. Par contre, une entreprise qui souhaiterait commercialiser un nouveau produit sur un marché a tout intérêt à se baser sur les critères les plus objectifs possibles. Dans ce contexte, on parle d'optimisation d'une fonction objectif. Nous reviendrons sur la notion de *critère d'optimalité* lorsque nous parlerons de planification.

Dans cette thèse, nous ne considérons que des agents rationnels. Les actions qu'ils effectuent sont des décisions rationnelles compte tenu des observations qu'ils font de leur environnement et de leurs connaissances internes. En particulier, cela signifie que l'on ne cherche pas à simuler ou étudier un modèle de l'humain, avec tout ce que ce dernier a de subjectif et de non-rationnel. Cela relève du domaine de la théorie de la décision descriptive. Les agents que l'on considère sont ceux de la théorie de la décision normative, ceux sur lesquels on peut exercer un **contrôle** et dont on attend un comportement rationnel. Robots, drones, programmes informatiques sont les exemples les plus évidents. On peut aussi penser à toutes les situations où quelqu'un est subordonné : militaires, policiers, pompiers, agents de sécurité, sportifs professionnels ; en somme nombre de cas où un humain reçoit d'un supérieur, d'un coach ou d'un conseiller des instructions ou des ordres à suivre.

2.1.3 Environnement

L'environnement correspond à l'agent et tout ce qui entoure l'agent. Pour un joueur de basket-ball, il s'agit de lui-même, du terrain, des autres joueurs, du coach, du public, du temps qu'il reste à jouer, du score actuel, de l'arbitre, de l'expression sur le visage de l'arbitre, de la météo à l'extérieur. Évidemment, toutes les informations dans l'environnement d'un agent ne sont pas des informations essentielles à son comportement rationnel. Par exemple, un sportif professionnel se doit de faire abstraction du public s'il veut prendre des décisions optimales en tout temps. De la même façon, peu lui importe la météo à l'extérieur si cela n'influe pas sur le cours du jeu.

Le **système** est défini comme l'ensemble des éléments de l'environnement qui sont pertinents à l'agent pour la résolution optimale de sa tâche. C'est la modélisation formalisée de l'environnement que le concepteur fait du problème. À un instant donné, le système se trouve dans un **état**,

qui est une collection de variables instanciées qui décrivent la situation, ou configuration actuelle du système. En observant directement la valeur de ces variables (ou bien en en faisant une observation partielle, comme nous le verrons plus tard), l'agent peut prendre des décisions optimales. Souvent l'état du système est exprimé selon deux composantes : l'**état interne** à l'agent et l'**état externe**. L'état interne représente les composantes de l'agent lui-même, comme sa position géographique, son état physique ou bien sa quantité de batterie restante. L'état externe est la composante de l'état qui représente les variables d'intérêt de l'environnement autour de l'agent, comme la surface de terrain brûlée par un feu de forêt combattu par des pompiers, la position des pièces sur un échiquier ou bien le score d'un match.

Un agent, en effectuant des actions, va influer sur l'état du système et le faire évoluer. Un joueur peut prendre l'action de *dribbler* (changement d'état interne : modification de sa position géographique) ou bien de *tirer* (changement d'état externe : modification du score). Lorsque le système passe d'un état à un autre, on dit qu'il *transite* vers ce nouvel état. Un système peut transiter d'un état à un autre via les actions des agents dans le système uniquement (par exemple, le score d'un match ne changera pas si les joueurs ne font rien). On dit que le système est **statique**. Si, en revanche, l'état d'un système peut évoluer indépendamment de l'action des agents, alors on dit que le système est **dynamique** (par exemple, un feu de forêt se propage s'il n'est pas activement combattu par des brigades de pompiers).

De plus, l'effet des actions d'un agent sur le système n'est pas toujours **déterministe**. Aux échecs par exemple, les actions ont des effets déterministes (on parle, par extension, d'*actions déterministes*) : un joueur ne peut pas "manquer" un coup par hasard. Après avoir joué une pièce depuis un certain état de l'échiquier, on peut toujours savoir assurément quel sera l'état suivant de l'échiquier. Mais ce déterminisme n'est pas présent dans de nombreux cas, comme au basket-ball par exemple. Un cas parlant est celui d'un joueur tentant un lancer à 3 points. Les probabilités qu'il marque le panier dépendent de nombreux facteurs (joueurs adverses, distance au panier, posture au moment du lancer, etc.). L'état du système ne va donc pas forcément transiter vers un état où le score est augmenté de 3 points pour l'équipe du lanceur. Ce système est dit **stochastique** : les effets des actions sont soumis à une part de hasard, et au moment où l'agent effectue son action, le prochain état du système n'est pas prédéterminé.

L'objectif des agents au sein du système est de sélectionner les actions qui vont amener le système vers des états *profitables*, et ce de façon la plus probable possible. C'est ici que la notion de planification entre en jeu.

2.2 Planification

Nous avons dit que le rôle d'un agent rationnel qui interagit avec son environnement est de résoudre une tâche, mais nous n'avons pas formellement décrit en quoi consistait cette tâche. Pour un joueur de basket-ball, c'est la victoire de son équipe. Pour un automobiliste, c'est de trouver une place de parking répondant à certains critères. Pour un médecin dépêché sur le site d'une catastrophe naturelle, c'est de soigner le plus efficacement possible le plus grand nombre de victimes.

2.2.1 Mesure de performance

La **planification** est l'étape qui consiste à calculer à l'avance un **plan**, ou une **stratégie**, qui permettra à un agent d'accomplir sa tâche. Un exemple simple pour décrire l'importance de la planification est une partie d'échecs. Un joueur d'échecs réfléchit toujours plusieurs coups à l'avance avant de jouer une pièce (du moins s'il est assez bon). Bien sûr il pourrait, à chaque fois

que son tour arrive, jouer le coup qui maximise sa récompense immédiate (par exemple, prendre un pion, voire une pièce de plus grande valeur). Mais bien souvent aux échecs, le coup qui semble le plus avantageux sur le moment ne l'est pas sur le long terme. Par exemple, le gambit, l'action qui consiste à sacrifier l'une de ses propres pièces, peut permettre à un joueur, malgré une perte matérielle, d'obtenir un avantage stratégique (ouverture d'une zone de l'échiquier, mise en danger d'une pièce adverse, etc.) et surtout, pas forcément immédiat.

Un agent rationnel cherche à optimiser une **mesure de performance** (soit la maximiser, soit la minimiser, selon les cas). C'est selon cette mesure que le comportement d'un agent rationnel peut être jugé comme étant optimal ou sous-optimal. Il revient au concepteur du système de bien choisir la mesure de performance s'il souhaite concevoir des agents qui vont agir dans le but de résoudre la tâche d'intérêt. En d'autres termes, la mesure de performance correspond à l'objectif de l'agent. Aux échecs par exemple, le but est de mettre le roi adverse échec et mat, toute autre issue étant indésirable. La séquence d'actions des blancs en elle-même n'est pas importante du moment qu'elle finit par faire transiter le système dans l'état *Échec et mat pour les blancs*. Ici, la mesure de performance est binaire : 0 pour une défaite et 1 pour une victoire. Le tableau 2.1 donne quelques exemples de situations où un agent se retrouve confronté à une tâche et cherche à optimiser une mesure de performance.

	Situation	Mesure de performance
	Un joueur au casino joue à une table de blackjack	Maximiser la somme des gains cumulés
	Une équipe de pompiers s'active à éteindre un feu de forêt qui se propage	Minimiser la surface détruite par les flammes
	Un robot d'accueil dans un centre commercial accompagne un client vers une boutique	Amener le client à sa destination
	Un État entame des négociations avec un autre État pour limiter la prolifération d'armes nucléaires	Éviter un conflit nucléaire

Tableau 2.1 – Exemples de situations où un agent cherche à optimiser une mesure de performance.

Suivant les cas étudiés donc, optimiser la mesure de performance peut consister à effectuer les actions optimales, quel que soit l'état dans lequel le système se trouve, mais peut également consister à amener le système dans un certain état souhaité ou à le maintenir hors de certains états défavorables.

2.2.2 Résolution

L'étape de planification est là pour calculer le plan, ou la stratégie, que les agents devront suivre afin d'optimiser la mesure de performance choisie. On distingue deux approches pour faire de la planification : la planification **hors-ligne** et la planification **en-ligne**. La planification hors-ligne a lieu *avant* la résolution de la tâche. On calcule la stratégie, puis on la donne à l'agent, et celui-ci s'en sert pour agir de façon optimale, sans avoir à refaire le moindre calcul durant

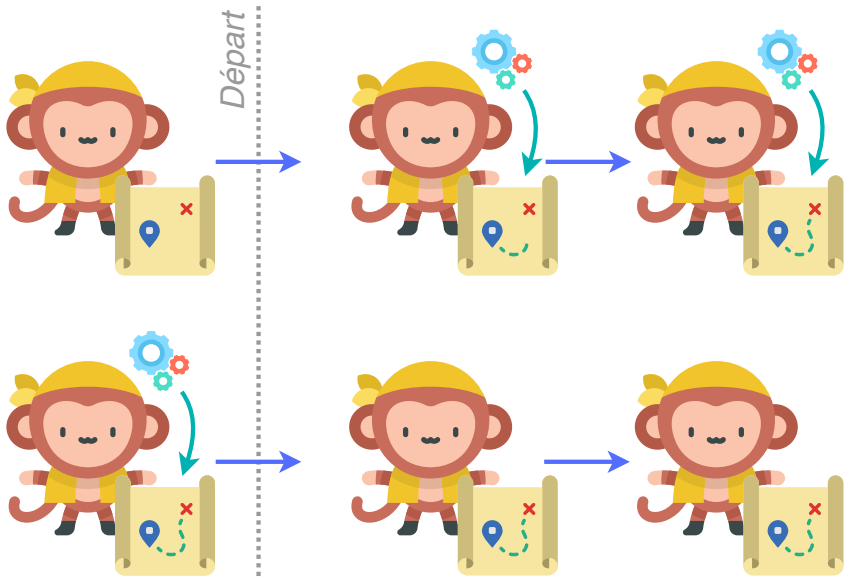


FIGURE 2.2 – Différence entre la planification en-ligne (en haut) et la planification hors-ligne (en bas). L'agent, partant de sa position initiale (le curseur bleu sur sa carte) souhaite se diriger sur le lieu d'un trésor caché (la croix rouge). Il planifie en-ligne lorsqu'il cherche le meilleur itinéraire à emprunter *durant* son expédition, mettant à jour sa position sur sa carte et réfléchissant au chemin à suivre ensuite. Il planifie hors-ligne lorsqu'il calcule *avant* son départ (matérialisé par la ligne verticale pointillée) quel sera le meilleur itinéraire. Après son départ, il ne lui reste alors plus qu'à suivre son plan.

la résolution de sa tâche. À l'inverse, la planification en-ligne a lieu *pendant* la résolution de la tâche. Au fur et à mesure de ses actions et des observations qu'il reçoit, l'agent calcule les meilleures prochaines actions à faire en fonction de l'état actuel du système. La figure 2.2 illustre la différence entre ces deux types de planification.

La planification hors-ligne est particulièrement adaptée lorsqu'il s'agit de coordonner à l'avance une équipe d'agents, surtout lorsque ceux-ci n'ont pas de moyens de communiquer durant la résolution de la tâche. C'est donc l'approche hors-ligne qui sera utilisée tout au long de cette thèse.

Un **plan** correspond à la suite d'actions que l'agent doit effectuer, tandis qu'une **stratégie** (aussi appelée **politique**) lui dit quelle action effectuer selon la situation dans laquelle se trouve l'état du système. Idéalement, l'objectif est d'optimiser absolument la mesure de performance, c'est-à-dire d'assigner des politiques aux agents telles que le comportement global de l'ensemble des agents soit optimal (autrement dit, parfaitement rationnel) et permette d'atteindre la valeur maximale (ou minimale) de la mesure de performance. Prenons l'exemple de la brigade de pompiers œuvrant à l'extinction d'un feu de forêt. Leur mesure de performance est la surface totale brûlée par les flammes, et l'objectif est donc de minimiser cette valeur. Assurément, il n'est pas concevable d'imaginer réduire cette valeur à zéro, puisque les flammes ravageront quoi qu'il en soit une parcelle non-nulle de terrain, aussi petite soit-elle. La valeur théorique optimale que les agents peuvent atteindre n'est pas connue *a priori*. Dans de nombreux problèmes, la valeur optimale de la mesure de performance ne peut pas être aisément trouvée. Cela n'est pas le cas aux échecs, où la valeur minimale est 0 (pour une défaite), et la valeur maximale est 1 (pour une victoire). Lorsque l'on cherche à atteindre exactement la valeur optimale de la mesure

de performance, on parle de **résolution optimale** du problème étudié. Si l'on se contente de vouloir approcher la valeur optimale de la mesure de performance, alors on parle de **résolution approximative**.

Dans cette thèse, on se concentre sur la planification dans les processus à **temps discret**. Cela signifie que l'on suppose que le temps et l'écoulement du temps sont découpés en intervalles, généralement de taille fixe. L'agent prend une décision au bout de chaque intervalle de temps, de façon régulière. De même, l'état du système n'est évalué qu'une seule fois par intervalle de temps. On parle de **pas de temps**, ou d'**étapes de décision**, pour désigner ces intervalles. Cette vision s'oppose aux processus à **temps continu**, où le système peut évoluer de façon continue et où l'agent peut prendre des décisions à n'importe quel moment. Par exemple, un match sportif est un processus à temps continu, en revanche une partie d'échecs est un processus à temps discret. Bien que les processus à temps continu soient généralement les plus courants dans la réalité, il est toutefois possible de *discrétiser* le temps en intervalles. Aussi nous ne considérerons dans cette thèse que des problèmes à temps discret ou à temps continu que l'on peut discrétiser.

Un **problème de planification multi-agents** est le nom donné au problème où l'on se pose la question de savoir quels sont les meilleurs plans, politiques ou stratégies que l'on peut assigner à chaque agent du système étudié. Autrement dit, on parle de planification multi-agents comme du moyen qui consiste à calculer l'activité jointe d'un groupe d'agents pour qu'ils agissent d'une manière coordonnée et optimale. La planification est donc l'étape de conception de la stratégie qui "crée" l'intelligence des agents, et qui fait alors tomber notre travail dans le très vaste domaine de l'**intelligence artificielle** [Russell and Norvig, 2009]. Notre objectif est de concevoir des agents capables de remplir des objectifs de façon *intelligente* – en somme, de concevoir des joueurs d'échecs qui calculent cinq, dix, vingt coups à l'avance. Sûrement ces joueurs sont-ils les plus intelligents !

2.3 Conclusion

Ce chapitre a introduit les concepts de base essentiels à la compréhension de cette thèse. Nous y avons donné les définitions d'agent, d'environnement et de système, ainsi que de système multi-agents. Nous avons fait la différence entre systèmes statiques et dynamiques, déterministes et stochastiques, ainsi que ouverts et fermés. Nous avons également décrit les notions importantes de rationalité et de mesure de performance qui vont permettre de faire de la planification.

Notre sujet d'étude concerne la **planification dans les systèmes multi-agents coopératifs dynamiques, stochastiques et ouverts**. Le prochain chapitre est dédié à introduire les modèles formels qui permettent de faire de la planification dans les systèmes mono et multi-agents. Nous y parlerons également des modèles étendus qui permettent de considérer les cas où les agents n'ont pas une observabilité totale de leur environnement, ainsi que du contrôle décentralisé. En somme, nous allons maintenant présenter les éléments de base pour faire de la planification dans les systèmes multi-agents coopératifs dynamiques et stochastiques. L'aspect ouvert, qui est le verrou scientifique principal de cette thèse, n'y sera pas encore abordé, mais laissé pour le chapitre suivant.

Chapitre 3

Planification sous incertitude

Sommaire

3.1 Introduction 20
3.2 Processus décisionnels de Markov 20
3.2.1 Définition du modèle 20
3.2.2 Exemple 22
3.2.3 Politiques 24
3.2.4 Principe d'optimalité 25
3.3 Observabilité partielle 28
3.3.1 Définition du modèle 29
3.3.2 État de croyance 32
3.3.3 Belief MDP 34
3.3.4 Complexité 37
3.4 Contrôle décentralisé 38
3.4.1 Définition du modèle 38
3.4.2 Historiques et politiques 40
3.4.3 Complexité 43
3.4.4 Méthodes de résolution 44
3.5 Conclusion 46

« La difficulté, avec les labyrinthes, c'est qu'on ne saura qu'à la fin si l'on a choisi le bon chemin ou pas. Et si en fin de compte on s'est trompé, il est en général trop tard pour repartir en arrière et recommencer. C'est le problème avec les labyrinthes. »

Haruki Murakami
L'Étrange Bibliothèque

3.1 Introduction

Dans le domaine de la planification en intelligence artificielle, on distingue principalement deux catégories : la planification dite *classique* [Bylander, 1994] pour les environnements déterministes où les issues des actions et l'état du monde sont parfaitement connus à l'avance (par exemple, la recherche du plus court chemin dans un labyrinthe dont on possède une carte) et la planification dite *sous incertitude*, ou *probabiliste* [Kushmerick et al., 1995] pour les environnements stochastiques, où une part de hasard, d'imprévu, entre en jeu (tels qu'une partie de poker). Dans cette thèse, nous allons nous focaliser sur les environnements stochastiques, donc sur la planification sous incertitude en utilisant les processus décisionnels de Markov.

3.2 Processus décisionnels de Markov

Un **processus décisionnel de Markov** [Puterman, 1994] (*Markov Decision Process*, **MDP**) est un modèle à temps discret pour la planification d'un agent autonome en environnement stochastique.

3.2.1 Définition du modèle

Définition 3.2.1 - *Processus décisionnel de Markov* : Un MDP est un tuple

$$(\mathcal{S},\mathcal{A},T,R),$$

où

- $\mathcal{S}$ est l'ensemble fini des états du système ;
- $\mathcal{A}$ est l'ensemble fini des actions de l'agent ;
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ est la fonction de transition du système, telle que $T(s^t,a^t,s^{t+1}) = Pr(s^{t+1} \mid s^t,a^t)$ est la probabilité d'arriver dans l'état $s^{t+1} \in \mathcal{S}$ une fois que l'agent a effectué l'action $a^t \in \mathcal{A}$ depuis l'état $s_t \in \mathcal{S}$;
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ est la fonction d'utilité, telle que $R(s^t,a^t,s^{t+1})$ est la récompense obtenue par l'agent (ou le coût payé, en cas de récompense négative) après qu'il a effectué l'action $a^t \in \mathcal{A}$ depuis l'état $s^t \in \mathcal{S}$ et que le système a transité dans l'état $s^{t+1} \in \mathcal{S}$.

Initialement, à l'instant $t = 0$, le système est dans un état $s^0 \in \mathcal{S}$. L'agent choisit l'une de ses actions $a^0 \in \mathcal{A}$, et le système passe dans l'état $s^1 \in \mathcal{S}$ avec la probabilité $T(s^0,a^0,s^1)$, où la récompense $R(s^0,a^0,s^1)$ est finalement reçue par l'agent. Ce processus continue séquentiellement soit indéfiniment, c'est-à-dire sans limite de temps préalablement fixée ; soit, jusqu'à ce qu'un **horizon de planification** h soit atteint (c'est-à-dire lorsque $t = h - 1$) autrement dit, après que l'agent a réalisé h actions.

Pour que la fonction de transition T puisse représenter un modèle crédible de l'évolution de l'environnement lorsqu'un agent agit sur celui-ci, il faut qu'elle définisse une **distribution de probabilité** valide, c'est-à-dire :

$$\forall s^t \in \mathcal{S},\ \forall a \in \mathcal{A},\ \sum_{s^{t+1} \in \mathcal{S}} T(s^t,a^t,s^{t+1}) = 1. \tag{3.1}$$

De plus, dans un MDP, la fonction de transition possède la **propriété de Markov**, qui spécifie que, à tout instant t, la probabilité de transiter d'un état $s^t \in \mathcal{S}$ à un état $s^{t+1} \in \mathcal{S}$

après que l'agent a effectué l'action $a^t \in \mathcal{A}$ ne dépend pas des précédents états du système ni des précédentes actions effectuées par l'agent :

$$Pr(s^{t+1} \mid s^0,a^0,s^1,a^1,\ldots,s^t,a^t) = Pr(s^{t+1} \mid s^t,a^t). \tag{3.2}$$

⁎ **Chaînes de Markov.** Le processus qui décrit l'évolution sous-jacente du modèle T est ce que l'on appelle une **chaîne de Markov**. C'est un processus stochastique à temps discret qui possède la propriété de Markov, du nom de son inventeur, Andreï Markov, mathématicien russe du début du XX$^{\text{ème}}$ siècle [Markov, 1906]. Formellement, une chaîne de Markov est une suite de variables aléatoires $(X^t \mid t \in \mathbb{N})$ vérifiant la propriété de Markov, c'est-à-dire telles que

$$Pr(X^{t+1} \mid X^t,\ldots,X^0) = Pr(X^{t+1} \mid X^t). \tag{3.3}$$

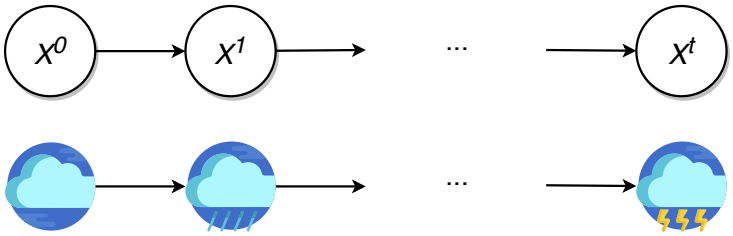


FIGURE 3.1 – Illustration d'une chaîne de Markov (abstraite au-dessus, illustrative au-dessous).

D'une certaine façon, on peut voir une chaîne de Markov comme un MDP avec une seule action : *Regarder* (ou bien *Ne rien faire*), et l'état s^t du système (la variable aléatoire de la chaîne de Markov) évolue de lui-même, passant d'une valeur à une autre naturellement avec le temps. Par exemple, la météo peut être vue comme une chaîne de Markov : aucune action n'exerce de contrôle sur la météo, et celle-ci possède bien la propriété de Markov, puisqu'il n'y a pas besoin de savoir le temps qu'il faisait hier pour déterminer le temps de demain, seule la météo du jour est nécessaire ; toute l'**information suffisante** à une prédiction optimale est contenue dans l'état *présent* de la météo. Certes, des données statistiques passées sur le temps qu'il faisait le même jour les années précédentes peuvent aider à prévoir la météo future, mais d'un point de vue informationnel, si toutes les données (température, pression, vitesse et direction des vents en chaque point de l'atmosphère) étaient mesurables à un instant t (aujourd'hui), alors il s'agirait d'informations suffisantes pour prévoir le temps à l'instant $t+1$ (demain). Un contre-exemple de chaîne de Markov peut être l'évolution de la valeur d'un indice sur un marché boursier. Il n'y a pas d'action directe qui fasse augmenter ou baisser la valeur de l'indice (c'est un phénomène décentralisé et aux causes multiples). La seule valeur de l'indice à l'instant t ne suffit pas à savoir si la tendance est à la hausse ou à la baisse. Un MDP, quant à lui, est une chaîne de Markov sur laquelle on exerce un contrôle, contrôle représenté par l'ensemble d'actions $\mathcal{A}$. On donne un exemple détaillé de MDP dans la section suivante.

La fonction de récompense R assigne une récompense (ou un coût) à l'agent lorsque celui-ci effectue une action depuis un état et que le système transite vers un nouvel état. On rencontre souvent des définitions alternatives de la fonction de récompense, qui peut n'être définie que sur l'action et l'état actuel :

$$R : \mathcal{S} \times \mathcal{A} \to \mathbb{R},$$

ou bien n'être définie que sur les états :

$$R : \mathcal{S} \to \mathbb{R}.$$

L'objectif d'un agent évoluant dans un système modélisé par un MDP est de choisir les actions à effectuer afin d'amener le système dans des états favorables, c'est-à-dire des états où la récompense est élevée. La fonction de récompense peut également être définie de manière à identifier des **états buts**, c'est-à-dire des états où la récompense est très élevée, le reste des états générant une récompense négligeable, voire nulle. La forme de la fonction de récompense peut être choisie en fonction du problème que l'on souhaite modéliser. Une fonction de récompense définie uniquement sur l'espace d'états $\mathcal{S}$ implique que peu importe la manière, l'agent atteint son but en amenant le système dans un (ou des) état(s) particulier(s). Définir la fonction de récompense également sur l'espace d'actions $\mathcal{A}$ implique que la manière d'arriver au but importe. Par exemple, un robot d'escorte devant mener une personne d'un point à un autre n'a pas intérêt à renverser ou bousculer toute personne qui se trouverait sur son passage, mais, au contraire, devrait être récompensé pour tenir compte de la présence d'individus autour de lui.

Comme nous le verrons, il ne suffit pas de sélectionner, dans un état s^t, l'action

$$a^t = \max_{a \in \mathcal{A}} R(s^t, a) \tag{3.4}$$

qui maximise de façon gloutonne la récompense dans l'état courant, puisque cela pourrait amener le système dans un état défavorable où les récompenses futures seraient très faibles. Cela reviendrait à explorer un labyrinthe en empruntant systématiquement des chemins prometteurs à première vue, nous rapprochant de la sortie en terme de distance, pour finalement être bloqué et devoir faire demi-tour. Cela s'illustre également bien dans l'exemple du joueur d'échecs. La prise immédiate et irréfléchie d'une pièce apporte toujours un gain strictement positif au joueur, mais ce gain peut se révéler être une perte sur le long terme si l'adversaire avait anticipé la prise de sa pièce et mis en place une stratégie efficace de contre-attaque qui lui permet de remporter la partie. Remarquons finalement, sans perte de généralité et par souci de simplicité, que l'on parlera de récompense même si les valeurs prises par la fonction R sont négatives et représentent en fait des coûts.

Illustrons à présent les notions décrites jusque là dans un exemple qui nous servira de fil rouge tout au long de cette partie.

3.2.2 Exemple

Considérons la situation suivante. Un vaillant explorateur vient de percer à jour les mystères d'anciennes ruines d'une civilisation précolombienne depuis fort longtemps disparue. Les ruines possèdent une entrée secrète qui mène à une vaste galerie de couloirs sombres et dangereux. À l'intérieur de ce labyrinthe souterrain se trouvent une multitude de trésors de très grande valeur, mais également des pièges pouvant se révéler être des dangers de mort pour notre aventurier. Téméraire, il décide toutefois de s'engouffrer à ses risques et périls dans l'entrée des ruines. Les récompenses qu'il espère obtenir valent largement les risques encourus. Ou alors, pas vraiment ?

Une modélisation simplifiée de ce scénario est représentée sur la figure 3.2. Les nœuds correspondent aux états du système. Ici, le "système" est l'agent lui-même – c'est-à-dire l'aventurier. Il y a trois états possibles : *En sécurité*, lorsque l'agent se trouve dans les dédales souterrains sans que sa vie ne soit menacée ; *En danger*, lorsqu'au contraire, dans le souterrain, il se trouve confronté à un danger qui met sa vie en péril ; et *À l'extérieur*, lorsqu'il décide finalement de sortir des ruines. On a donc $\mathcal{S} = \{$ *En sécurité, En danger, À l'extérieur* $\}$. L'état initial du système, spécifié par la courte flèche verticale, est l'état $s^0 =$ *En sécurité*.

L'agent possède deux actions : *Explorer* et *Sortir*. Si l'agent effectue l'action *Explorer*, il passera de l'état *En sécurité* à l'état *En danger* avec une probabilité 0,2. Sinon, avec une probabilité 0,8, il restera dans le même état. Cela signifie que les ruines sont relativement peu dangereuses,

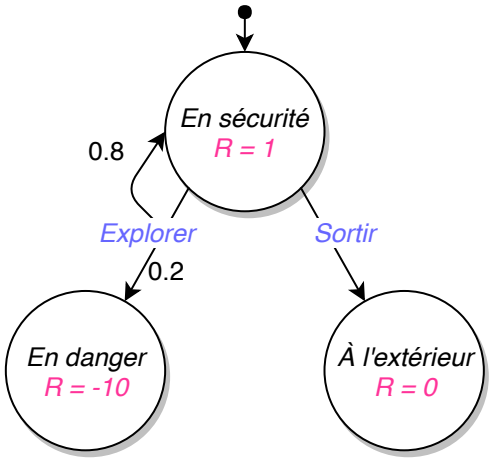


FIGURE 3.2 – Un exemple de MDP simple.

puisqu'il n'y a qu'une chance sur cinq pour que l'agent tombe dans un piège et se retrouve en danger. Si l'agent effectue l'action *Sortir*, alors il finira assurément (avec probabilité de 1) dans l'état *À l'extérieur*. Lorsque l'agent se retrouve dans l'état *En danger* ou l'état *À l'extérieur*, on suppose que le processus s'arrête immédiatement.

Enfin, l'agent reçoit une récompense fixe de *1* unité lorsqu'il se trouve *En sécurité* dans les ruines. Cela correspond aux trésors qu'il trouve et amasse sur sa route en descendant toujours plus profondément dans le souterrain. En revanche, il perd *10* unités lorsqu'il se trouve dans l'état *En danger*, ce qui représente le fait qu'il se blesse dangereusement, qu'il se retrouve coincé durant un long moment dans un piège, etc. Lorsqu'il sort du souterrain, il ne reçoit aucune (*0*) récompense.

Pour résumer, le MDP qui correspond à cette situation est le tuple $(\mathcal{S},\mathcal{A},T,R)$ avec :

— $\mathcal{S} = \{$ *En sécurité*, *En danger*, *À l'extérieur* $\}$

— $\mathcal{A} = \{$ *Explorer*, *Sortir* $\}$

— $T($*En sécurité*, *Explorer*, *En sécurité*$) = 0{,}8$

— $T($*En sécurité*, *Explorer*, *En danger*$) = 0{,}2$

— $T($*En sécurité*, *Sortir*, *À l'extérieur*$) = 1$

— $R($*En sécurité*$) =$ *1*

— $R($*En danger*$) =$ *-10*

— $R($*À l'extérieur*$) =$ *0*

Remarquons que la fonction de récompense R est de la forme $R : \mathcal{S} \to \mathbb{R}$. Toutefois, dans cet exemple, l'agent ne cherche pas à atteindre un *état but* dans lequel il se complairait, car même s'il reste dans l'état *En sécurité*, il devra ensuite forcément faire le choix soit de continuer, soit de sortir. Notons également que toutes les probabilités non spécifiées dans le modèle T sont des probabilités nulles. Par exemple, $T($*À l'extérieur*, *Sortir*, *À l'extérieur*$) = 0$. En effet, une telle situation n'a pas de raison d'exister relativement au sens que l'on donne à notre modélisation du problème. Même si la fonction T est donc définie correctement sur tout l'ensemble des états et des actions, il n'est donc pas nécessaire d'énumérer de telles situations impossibles.

3.2.3 Politiques

Nous l'avons vu, un agent agit sur le système en effectuant des actions, modifiant ainsi l'état du système dans lequel il évolue. Le comportement d'un agent, sa stratégie – le mécanisme qui lui dit quelle action effectuer dans quel état – est encodé dans une **politique**.

Définition 3.2.2 - *Politique stochastique* : Soit M un MDP tel que défini en 3.2.1. Une *politique stochastique* π pour un agent dans le MDP M est une fonction

$$\pi : \mathcal{S} \times \mathcal{A} \to [0,1],$$

telle que

$$\forall s \in \mathcal{S}, \sum_{a \in \mathcal{A}} \pi(s,a) = 1. \tag{3.5}$$

La valeur $\pi(s,a)$ est la probabilité d'effectuer l'action $a \in \mathcal{A}$ si le système est dans l'état $s \in \mathcal{S}$. Le terme **stochastique** est relatif au fait que la fonction n'associe pas nécessairement la même action à un même état, mais plutôt une distribution de probabilité sur les actions. Si, dans un état $s \in \mathcal{S}$, la même action $a \in \mathcal{A}$ est toujours prescrite par la politique stochastique, autrement dit si

$$\forall s \in \mathcal{S}, \exists a \in \mathcal{A}, \pi(s,a) = 1, \text{ et } \forall a' \in \mathcal{A}, a' \neq a, \pi(s,a') = 0, \tag{3.6}$$

alors la fonction π est appelée **politique déterministe**. Par souci de simplicité, son domaine d'application est ainsi noté

$$\pi : \mathcal{S} \to \mathcal{A},$$

de sorte que $\pi(s) \in \mathcal{A}$ désigne l'action à effectuer par l'agent lorsque le système est dans l'état $s \in \mathcal{S}$.

Notons qu'une politique ainsi définie possède elle-même la propriété de Markov telle que décrite par l'équation 3.2 : l'action à effectuer dépend uniquement de l'état courant du système, et non des états et actions passés. De plus, une telle politique est dite **stationnaire**, car elle est indépendante de l'instant t. C'est-à-dire,

$$\forall t_1 \neq t_2 \in \mathbb{N}, \pi^{t_1}(s) = \pi^{t_2}(s), \tag{3.7}$$

où π^t désigne la politique lorsqu'elle est appliquée à l'instant $t \in \mathbb{N}$. Une politique non stationnaire pourrait, pour un même état, prescrire deux, voire plusieurs actions différentes en fonction de l'instant où la politique est appliquée. Dans la suite de ce manuscrit, nous n'allons considérer que des politiques stationnaires et, sauf mention contraire, le terme *politique* désignera une *politique déterministe stationnaire*. On peut toutefois remarquer que, intuitivement, une bonne politique pour l'exemple de la section 3.2.2 se doit d'être non-stationnaire, puisque l'exemple décrit pose en quelque sorte la question : *quand* faut-il sortir ? Le but est en effet de ne pas tomber dans l'état *En danger* tout en effectuant un maximum de fois l'action *Explorer* afin de récolter le plus de récompenses. Mais qu'est-ce qu'une *bonne* politique ?

Une politique π, lorsqu'elle est exécutée à l'instant $t = 0$ depuis un état s_0, induit une séquence de récompenses

$$R(s^0,\pi(s^0)) \to R(s^1,\pi(s^1)) \to R(s^2,\pi(s^2)) \to \dots$$

où $R(s^t,\pi(s^t))$ est la récompense émise à l'instant t, et s^t est l'état à cet instant. Cet état est généré de façon stochastique via la fonction de transition T lorsqu'une action est effectuée

depuis l'état précédent. En effet, une même politique, exécutée deux fois de suite depuis un même état initial et sur une même durée de temps, ne va pas produire la même séquence de récompenses, car, même si la politique est déterministe, l'effet de ses actions sur le système, lui, est stochastique. Qui plus est, deux politiques différentes (qui, pour un même état, associent deux actions différentes) ne vont pas non plus induire chez l'agent le même comportement. En pratique, cela va se traduire par le fait que, pour un état initial identique, le système ne va probablement pas visiter les mêmes états si l'on applique l'une ou l'autre des politiques. Cela signifie de plus que les récompenses générées ne seront pas nécessairement les mêmes. Il est donc possible d'analyser la qualité d'une politique en examinant les récompenses que celle-ci génère en moyenne. C'est ce que l'on appelle la *valeur* d'une politique.

Formellement, la valeur $V^\pi(s^t)$ d'une politique π dans un état s^t, suivie jusqu'à un horizon h, est la somme espérée des récompenses obtenues lorsque l'on exécute cette politique depuis cet état sur un nombre fixe de h étapes :

$$V^\pi(s^t) = \mathbb{E}\left[\sum_{t=0}^{h-1} \gamma^t R^t \mid \pi,\, s = s^t\right] \tag{3.8}$$

$$= R(s^t,\pi(s^t)) + \gamma \sum_{s^{t+1}\in\mathcal{S}} T(s^t,\pi(s^t),s^{t+1})V^\pi(s^{t+1}) \quad \forall s^t \in \mathcal{S}, \tag{3.9}$$

où $\mathbb{E}\{\cdot\}$ désigne l'opérateur de l'espérance mathématique, $R^t = R(s^t,\pi(s^t))$ est la récompense générée par la politique lorsque le système est dans l'état s^t à l'instant t, et $0 < \gamma \leqslant 1$ est un *facteur d'atténuation*. Si l'horizon de planification h est fini, alors on fixe $V^\pi(s^h) = 0$, pour tout état $s^h \in \mathcal{S}$.

La fonction de valeur V^π est ainsi définie de manière à estimer la somme moyenne des récompenses que l'agent peut espérer obtenir dans chaque état s'il suit la politique π. Une *bonne* politique est donc une politique ayant une valeur aussi grande que possible. L'équation 3.9 donnant la valeur d'une politique aura plus de sens lorsque sera introduit le concept d'*optimalité*, ci-après.

La constante γ est le **facteur d'atténuation** qui règle l'attention que l'agent doit porter aux récompenses éloignées dans le temps. Lorsque $\gamma \to 1$, alors l'agent ne considère pas les récompenses proches de lui comme plus importantes. En revanche, quand $\gamma \to 0$, alors l'agent accorde moins d'importance aux récompenses futures. Paramétré de façon adéquate, le facteur d'atténuation est utile pour négliger l'effet d'actions qui seront effectuées dans un lointain futur. Cela est particulièrement utile lorsque l'on considère des horizons de planification grands, voire infinis. Dans la majorité des cas, lorsque l'on considère des horizons de planification finis, on prend $\gamma = 1$.

3.2.4 Principe d'optimalité

Résoudre un MDP, c'est trouver une politique π^* telle que, pour tout état $s \in \mathcal{S}$ et pour toute autre politique $\pi \neq \pi^*$,

$$V^{\pi^*}(s) \geqslant V^\pi(s). \tag{3.10}$$

Une politique π^* vérifiant une telle propriété est appelée **politique optimale**. Ce n'est donc pas seulement une bonne politique, c'est aussi la meilleure. Notons toutefois qu'elle n'est pas nécessairement unique. Deux politiques différentes peuvent avoir la même valeur. La fonction de valeur d'une politique optimale π^* est généralement notée V^* plutôt que V^{π^*} [Bellman, 1954]. La solution d'un MDP est une politique optimale, calculée en amont de l'exécution, et donnée à

l'agent pour que celui-ci puisse l'utiliser en temps réel sans avoir besoin de faire un quelconque calcul durant l'exécution. C'est l'étape de planification.

Trouver une telle politique optimale a été le coeur des recherches du mathématicien américain Richard Bellman au début des années 1950 à la RAND Corporation, en Californie. Ses travaux ont été les fondements de ce qu'il a baptisé la **programmation dynamique**. Le terme *programmation*, s'il semble étrange de nos jours, à l'ère de l'informatique généralisée, avait à l'origine le sens de *planification*, *prise de décision*. Le terme *dynamique* quant à lui, soulignait l'aspect temporel et stochastique des problèmes que la méthode était vouée à résoudre. L'histoire raconte que Bellman aurait trouvé ces termes pour désigner son travail afin de ne pas s'attirer les foudres de son employeur, le secrétaire à la Défense, qui ne voulait surtout pas entendre parler de mathématiques et qui n'imaginait pas que quiconque sous sa direction puisse travailler dans ce domaine. La confusion sur le sens des termes était donc déjà justifiée à l'époque [Bellman, 1984].

L'idée de la programmation dynamique repose sur ce que Bellman a nommé le **principe d'optimalité**, et qui peut être retranscrit, dans le cas d'un MDP, de la façon suivante.

> **Principe d'optimalité.** Une politique optimale a la propriété que, quels que soient l'état initial et l'action initiale prise dans cet état, le reste des décisions qui seront prises doivent constituer une politique optimale au regard de l'état résultant de cette première décision.

Reprenons notre exemple pour illustrer simplement l'idée du principe d'optimalité. Oublions un instant l'aspect "MDP" et concentrons-nous sur une modélisation plus simple. Imaginons qu'avant son entrée dans les ruines, l'aventurier ait mis la main sur une carte des lieux qui lui indique la présence et la localisation exacte des nombreuses richesses. Depuis la pièce principale dans laquelle il se trouve, deux voies s'offrent à lui : à droite ou à gauche. En empruntant l'un ou l'autre des couloirs, il se retrouvera dans une nouvelle pièce, contenant un trésor d'une certaine valeur. Dans cette nouvelle pièce, deux nouvelles voies – droite ou gauche – s'offriront à nouveau à lui, donnant elles-mêmes sur de nouvelles pièces, etc. Une représentation de cette structure est donnée sur la figure 3.3.

S'il y a deux choix de couloirs par pièce et que l'aventurier doit faire n choix au total, cela fait un total de 2^n chemins à évaluer pour trouver lequel d'entre eux va lui permettre de récolter le plus de richesses, et donc $2^{n+1} - 2$ calculs à faire au total. Il n'a pas le temps de faire demi-tour, et doit donc emprunter un seul chemin : une fois arrivé au bout des ruines, il doit rapidement quitter les lieux par le chemin emprunté à l'aller s'il ne veut pas se retrouver piégé pour l'éternité. Plutôt que d'énumérer tous les chemins possibles et les évaluer, l'aventurier peut utiliser la programmation dynamique. Le principe d'optimalité lui dit ceci : s'il se trouve dans une des avant-dernières pièces – quelle que soit cette pièce, la valeur des trésors qu'elle contient et le chemin qu'il a fait pour y parvenir – alors l'action optimale qu'il peut faire est de choisir le chemin qui donne sur la pièce ayant le trésor de plus grande valeur. La valeur qu'il peut ainsi espérer obtenir depuis sa pièce actuelle n'est donc pas uniquement la valeur des trésors qu'elle contient, mais également celle de la meilleure prochaine pièce, la dernière. L'aventurier note cela sur son carnet, et réitère la même réflexion pour chaque autre avant-dernière pièce possible, puis pour chaque pièce avant chaque avant-dernière pièce, etc. Avec une telle méthode, il n'aura qu'à effectuer $\frac{n(n+1)}{2}$ calculs.

Dans cet exemple, l'aventurier doit choisir une option parmi deux à chaque niveau, et il doit le faire trois fois s'il veut arriver au bout des ruines. Avec une recherche exhaustive, pour évaluer la valeur de chaque pièce, il devra effectuer $2^{3+1} - 2 = 14$ calculs, et seulement $\frac{3(3+1)}{2} = 6$ calculs s'il décide d'utiliser la programmation dynamique. La complexité lorsque le nombre de

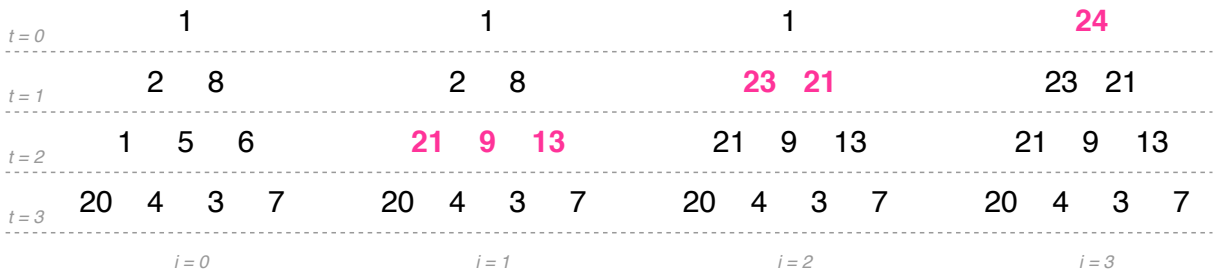


FIGURE 3.3 – Illustration du principe de programmation dynamique sur un problème d'optimisation dans un arbre binaire de profondeur $h = 3$. La procédure de programmation dynamique commence, à l'itération $i = 0$, au niveau $t = h - 1 = 2$, par remplacer la valeur de chaque nœud à ce niveau par la somme de la valeur de ce nœud-là et de la valeur de son meilleur successeur. Par exemple, à $i = 0$, $t = 2$, depuis le nœud de valeur 6, le meilleur successeur est le nœud de valeur 7 (l'autre étant celui de valeur 3). La valeur 6 est donc remplacée par $6 + 7 = 13$, car il s'agit de la "vraie" valeur que l'on peut atteindre au maximum depuis ce nœud-là. En remontant à chaque itération, niveau par niveau, jusqu'à la racine de l'arbre, on obtient la valeur optimale que l'on peut obtenir dans cet arbre (ici, 24) ainsi que le chemin pour y parvenir. Il suffit de partir de la racine de l'arbre à la dernière itération (ici, $i = 3$) et de sélectionner à chaque niveau le nœud ayant la plus grande valeur. Dans ce cas, le chemin optimal dans l'arbre initial est $1 \to 2 \to 1 \to 20$.

pièces et l'horizon augmentent favorise encore plus la programmation dynamique sur la recherche exhaustive.

Le principe d'optimalité dit que la fonction de valeur optimale V^* est solution de l'**équation d'optimalité de Bellman** :

$$V^*(s^t) = \max_{a \in \mathcal{A}} \left(R(s^t,a) + \gamma \sum_{s^{t+1} \in \mathcal{S}} T(s^t,a,s^{t+1}) V^*(s^{t+1}) \right) \quad \forall s^t \in \mathcal{S}. \tag{3.11}$$

Conformément au principe d'optimalité, cette expression, comme l'équation 3.9, se décompose en une somme de deux valeurs. D'un côté,

$$R(s^t,a)$$

est la récompense immédiate que l'agent va recevoir en appliquant l'action a dans l'état s^t. Dans notre exemple, c'est la valeur du trésor dans la pièce actuelle de l'aventurier. De l'autre côté,

$$\gamma \sum_{s^{t+1} \in \mathcal{S}} T(s^t,a,s^{t+1}) V^*(s^{t+1})$$

est la récompense future *espérée* (c'est-à-dire, pondérée par les probabilités de transiter dans les différents prochains états possibles) que l'agent va accumuler suite à l'action a qu'il vient d'effectuer dans l'état s^t. Dans notre exemple, il s'agit de la somme moyenne des valeurs des trésors des prochaines pièces que l'aventurier va visiter. Le processus ainsi décrit par l'équation 3.11 est en tout point identique à celui décrit par la figure 3.3 pour trouver le chemin optimal une fois que l'arbre a été mis à jour par programmation dynamique, à ceci près que, dans la formule générale de l'équation 3.11, la récompense future espérée est pondérée, tandis que dans l'exemple de la figure 3.3, les transitions d'un état à l'autre sont déterministes et les récompenses ainsi espérées non diminuées par un facteur d'atténuation γ.

Malheureusement, l'équation 3.11 ne possède pas de solution analytique (de forme fermée). Mais depuis les années 1950, de très nombreuses techniques et algorithmes basés sur le principe d'optimalité ont été mis au point afin de la résoudre. Nous allons brièvement examiner deux des algorithmes les plus communément utilisés : *Value Iteration* et *Policy Iteration*.

Value Iteration. L'algorithme d'itération sur la valeur (*Value Iteration*) [Bellman, 1956] construit la fonction de valeur de façon itérative en partant d'une fonction de valeur initiale V_0 et en procédant par mises à jour successives :

$$V_{i+1}(s) = \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_i(s') \right) \quad \forall s \in \mathcal{S}. \tag{3.12}$$

En partant à $i = 0$ avec une fonction de valeur V_0 aléatoirement initialisée pour tout état $s \in \mathcal{S}$, et en appliquant continuellement l'équation de mise à jour 3.12 pour tous les états, alors, lorsque $i \longrightarrow \infty$, l'algorithme est garanti de converger vers la fonction de valeur optimale V^*. Dans la pratique, l'algorithme s'arrête lorsque la mise à jour ne modifie plus la fonction de valeur à un $\epsilon \in \mathbb{R}$ près, c'est-à-dire lorsque :

$$|V_i(s) - V_{i-1}(s)| < \epsilon, \quad \forall s \in \mathcal{S}. \tag{3.13}$$

Une fois la fonction de valeur optimale V^* calculée, la politique optimale π^* se dérive naturellement (par définition) en sélectionnant de façon gloutonne l'action qui maximise dans chaque état cette fonction de valeur optimale :

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s') \right) \quad \forall s \in \mathcal{S}. \tag{3.14}$$

Policy Iteration. L'algorithme d'itération de la politique (*Policy Iteration*) [Howard, 1960] plutôt que de calculer la fonction de valeur optimale avant d'en extraire la politique optimale, procède itérativement pour construire et améliorer directement la politique à chaque itération. Initialement, avant la première itération $i = 0$, une politique $\pi_{i=0}$ est initialisée arbitrairement pour chaque état $s \in \mathcal{S}$. Une itération i de l'algorithme consiste à

1. calculer la valeur V^{π_i} de la politique π_i via l'équation 3.9,
2. puis mettre à jour la politique via l'équation 3.14 en se servant de la fonction de valeur V^{π_i} précédemment calculée plutôt que la fonction de valeur optimale V^*.

Ces deux étapes sont répétées en alternance jusqu'à ce que la politique ne puisse plus être améliorée, auquel cas la politique optimale a été trouvée.

Il n'y a pas de consensus sur lequel des deux algorithmes est le plus efficace. *Value Iteration* et *Policy Iteration* ont tous deux la même complexité en temps $O(|\mathcal{S}|^2|\mathcal{A}|)$ [Littman et al., 1995b]. Généralement, leur efficacité respective dépend du problème à résoudre, mais le plus souvent, c'est *Value Iteration* qui est utilisé, du fait de sa simplicité d'implémentation et de son antériorité historique. Il existe d'autres méthodes de résolution de MDP, mais cela est hors de la portée de cette thèse.

3.3 Observabilité partielle

Dans un MDP, comme on l'a vu dans l'exemple de la section 3.2.2, l'agent observe le système dans lequel il évolue. Cela signifie qu'à tout instant t il observe directement et connaît l'état

$s^t \in \mathcal{S}$ réel du système. Cette information lui permet de se servir de sa politique pour prendre une décision à cet instant-là, dans cet état-là. Qu'en serait-il à présent si l'agent n'était pas en mesure d'avoir l'information sur l'état du système ? Ou bien, s'il n'était capable d'en obtenir qu'une partie, voire de n'en faire qu'une observation indirecte, biaisée par ses propres mécanismes de perception ?

Les cas d'observabilité partielle sont très communs en pratique. En fait, ils sont même plus communs que les cas où l'état du système est parfaitement connu. Un robot qui se déplacerait dans un environnement inconnu, comme un nouveau bâtiment dont il ne dispose pas du plan, serait obligé de se servir uniquement de ses perceptions locales pour se diriger. Ces perceptions, ces mesures, peuvent être faites via ses capteurs (caméra couleur, laser de télédétection, microphones, etc.), mais elles seront presque toujours incomplètes et ne lui permettront pas assurément de construire une représentation exacte de son environnement. Cette perfectibilité peut notamment être due aux limites techniques de son matériel, ou bien au fait que l'environnement est complexe, que des personnes marchent autour de lui et brouillent ses signaux. D'où le besoin d'un cadre formel pour permettre le contrôle optimal d'un agent ayant un manque d'informations sur l'état du système.

3.3.1 Définition du modèle

Un **processus décisionnel de Markov partiellement observable** [Smallwood and Sondik, 1973] (*Partially Observable Markov Decision Process*, **POMDP**) est une généralisation du modèle de MDP aux cas où l'agent ne peut pas connaître directement l'état réel du système.

Définition 3.3.1 - *Processus décisionnel de Markov partiellement observable* : Un POMDP est un tuple

$$(\mathcal{S},\mathcal{A},\mathcal{O},T,O,R),$$

où

- $\mathcal{S}$ est l'ensemble fini des états du système ;
- $\mathcal{A}$ est l'ensemble fini des actions de l'agent ;
- $\mathcal{O}$ est l'ensemble fini des observations de l'agent ;
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ est la fonction de transition du système, telle que $T(s^t,a^t,s^{t+1}) = Pr(s^{t+1} \mid s^t,a^t)$ est la probabilité d'arriver dans l'état $s^{t+1} \in \mathcal{S}$ une fois que l'agent a effectué l'action $a^t \in \mathcal{A}$ depuis l'état $s_t \in \mathcal{S}$;
- $O : \mathcal{S}\times\mathcal{A}\times\mathcal{O}\times\mathcal{S} \to [0,1]$ est la fonction d'observation du système, telle que $O(s^t,a^t,o^{t+1},s^{t+1}) = Pr(o^{t+1} \mid s^t,a^t,s^{t+1})$ est la probabilité que l'agent observe $o^{t+1} \in \mathcal{O}$ quand le nouvel état est s^{t+1} une fois qu'il a effectué l'action $a^t \in \mathcal{A}$ depuis l'état $s^t \in \mathcal{S}$;
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ est la fonction de récompense, telle que $R(s^t,a^t,s^{t+1})$ est la récompense obtenue par l'agent après qu'il a effectué l'action $a^t \in \mathcal{A}$ depuis l'état $s^t \in \mathcal{S}$ et que le système a transité dans l'état $s^{t+1} \in \mathcal{S}$.

Dans un POMDP, l'agent fait des observations sur le système. L'incertitude liée au modèle est donc double : d'une part, au même titre qu'un MDP, il y a une incertitude sur les effets d'une action sur le système (modélisée par la fonction stochastique de transition T), et d'autre part, il y a une incertitude sur l'état réel du système due aux informations partielles, incomplètes, dont l'agent dispose (modélisée par la fonction stochastique d'observation O). La fonction

d'observation O est plus habituellement définie seulement sur les observations, actions et états d'arrivée :

$$O : \mathcal{A} \times \mathcal{O} \times \mathcal{S} \to [0,1],$$

voire uniquement sur les états :

$$O : \mathcal{O} \times \mathcal{S} \to [0,1].$$

Un cas particulier de fonction d'observation est le cas où l'observation reçue détermine totalement l'état réel du système. Si,

$$\forall o \in \mathcal{O},\ \exists s \in \mathcal{S} \text{ tel que } \ O(o,s) = 1, \tag{3.15}$$

alors le POMDP est dit **totalement observable**, et il est ainsi possible de se ramener au cas simple d'un MDP. Dans le cas le plus général toutefois, faire une observation ne suffit pas à déterminer assurément l'état du système. Même si la fonction d'observation O est déterministe (c'est-à-dire qu'une et une seule observation est toujours associée à un certain état), alors cela ne suffit pas pour disposer d'un POMDP totalement observable, car deux états peuvent être associés à la même observation.

Illustrons le principe d'observabilité partielle avec notre exemple fil rouge que nous allons, pour les besoins de l'illustration, une nouvelle fois modifier. Imaginons que les ruines découvertes par l'aventurier forment un immense labyrinthe au centre duquel se trouve un unique trésor. Et cette fois, aucune carte des ruines n'est disponible à son arrivée sur place. La situation pour l'agent est partiellement observable : il sait où il se trouve lorsqu'il pénètre dans le labyrinthe, mais une fois à l'intérieur, il ne peut connaître assurément sa position. Chaque couloir se ressemble, et plusieurs intersections peuvent être identiques. S'il observe qu'il se trouve à une intersection en forme de L, plusieurs localisations sont possibles, car il y a plusieurs intersections en forme de L dans ce labyrinthe. Nous allons le voir, il existe plusieurs façons pour lui d'estimer sa position dans le labyrinthe.

⁎ **Modèle de Markov caché.** Nous l'avons vu, un MDP s'appuie sur des chaînes de Markov sur lesquelles on exerce un contrôle, c'est-à-dire pour lesquelles on dispose d'actions nommées que l'on (ou plutôt, l'agent) peut explicitement utiliser pour orienter l'évolution des chaînes. Le modèle qui décrit l'évolution d'un POMDP est lui nommé **modèle de Markov caché** (*Hidden Markov Model*, HMM) [Baum and Petrie, 1966]. Dans un HMM, les variables décrivant l'évolution de l'état sont dites **cachées** (ou **latentes**). Des variables **observables** sont émises à chaque instant t par le système et renseignent sur la valeur de l'état actuel.

FIGURE 3.4 – Exemple de modèle de Markov caché. Les variables X sont des variables *latentes*, non observables, et forment une chaîne de Markov. Seules les variables Y, des observations émises par l'environnement, sont perceptibles.

La météo, précédemment décrite comme une chaîne de Markov, peut être modélisée de façon plus réaliste par un HMM. L'état réel du système (l'ensemble des variables météorologiques,

	Observable	*Partiellement Observable*
Pas de contrôle	Chaîne de Markov	Modèle de Markov caché
Contrôle	MDP	POMDP

Tableau 3.1 – Positionnement des modèles décisionnels présentés.

nombreuses et complexes à mesurer) est presque inconnu à proprement parler, du moins pour qui n'est pas météorologue. En revanche, un personne peut faire une *observation* du temps : fait-il beau ? Y'a-t-il beaucoup de vent ? Qu'il *fasse beau*, ce n'est pas l'état actuel de la météo, c'est juste l'information perçue. L'état réel, c'est qu'il fait x degrés Celsius à l'ombre, et qu'il y a un vent soufflant à z km/h dans la direction y. Notons toutefois que ce qui est défini comme l'"état" est décidé par celui ou celle qui conçoit la chaîne de Markov. On peut très bien imaginer une chaîne où les états sont {*Il fait beau*, *Il ne fait pas beau*} et les observations {*Soleil*, *Nuages*, *Pluie*}. Un POMDP, c'est un HMM pourvu d'actions de contrôle qui vont diriger l'évolution des variables latentes du modèle. Le tableau 3.1 résume le positionnement des différents modèles présentés jusqu'ici suivant le contrôle et l'observabilité dont on dispose.

Contrairement à un MDP, dans un POMDP un agent ne peut pas choisir son action en fonction de l'état actuel du système, car il n'observe justement pas cet état. En revanche, la fonction d'observation O donne des informations essentielles à l'agent. En effet, lorsque celui-ci effectue une action et que le système transite d'un état à un autre, l'observation perçue par l'agent l'informe de ce changement d'état, et lui donne donc une *intuition* sur l'état réel du système. Un agent peut donc maintenir une séquence, un **historique**, des observations successives qu'il fait – à la manière de l'explorateur qui pourrait noter sa progression sur son carnet d'aventure au fur et à mesure de son avancée dans le labyrinthe, dessinant ainsi un plan partiel de son environnement, ou bien encore à la façon de l'historique d'un navigateur web qui enregistre, sur une fenêtre d'une certaine longueur, les derniers sites internet visités.

Définition 3.3.2 - *Historique d'observations* : Soit M un POMDP. Un *historique d'observations* $\vec{o}^t$ est une séquence finie de t observations reçues par l'agent depuis le début du processus :

$$\vec{o}^t = (o^1, o^2, \ldots, o^t) \in \mathcal{O}^t. \tag{3.16}$$

On note $\vec{\mathcal{O}}^t$ l'ensemble des historiques d'observations de longueur t de l'agent et $\vec{\mathcal{O}}$ l'ensemble de tous les historiques d'observations de cet agent. Sauf mention contraire, on utilisera en général la notation $\vec{o} \in \vec{\mathcal{O}}$. Comme pour un MDP, on peut alors dériver la définition d'une politique pour l'agent d'un POMDP. La politique d'un agent est une fonction associant une action à chaque historique d'observations possible.

Définition 3.3.3 - *Politique* : Soit M un POMDP. Une *politique déterministe* π pour l'agent du POMDP M est une fonction

$$\pi : \vec{\mathcal{O}} \to \mathcal{A}.$$

Maintenir un historique des observations perçues peut être pratique, car il est possible d'en inférer beaucoup d'informations. On peut se demander : quelle est la probabilité de faire la séquence d'observations $(o^1, o^2, \ldots, o^t)$? Ou bien, étant donnée une séquence d'observations passées

$(o^1,o^2,\ldots,o^t)$, quelle est la probabilité $Pr(s^t \mid o^1,o^2,\ldots,o^t)$ d'être dans l'état $s^t \in \mathcal{S}$? L'historique des observations est une statistique suffisante à la prise de décision optimale car il contient toutes les informations que peut posséder un agent. Un inconvénient de l'historique d'observations est que, si l'horizon de planification h est grand, alors l'historique peut lui-même devenir très long, ce qui donne un nombre d'historiques possibles

$$\sum_{t=0}^{h-1} |\mathcal{O}|^t = \frac{|\mathcal{O}|^h - 1}{|\mathcal{O}| - 1}$$

qui croît rapidement avec la taille de l'horizon h. Un agent ayant 2 observations possibles ($|\mathcal{O}| = 2$) aura $2^{10} - 1 = 1023$ historiques d'observations possibles pour $h = 10$. Cette explosion combinatoire est d'autant plus problématique que, si un agent doit avoir une politique qui lui prescrit l'action à effectuer en fonction de son historique d'observations, alors il aura

$$|\mathcal{A}|^{\frac{|\mathcal{O}|^{h-1}}{|\mathcal{O}|-1}}$$

politiques différentes possibles. Pour 1023 historiques d'observations, un agent avec 2 actions possibles ($|\mathcal{A}| = 2$) pourra suivre $2^{1023} \approx 8{,}98.10^{307}$ politiques différentes. À titre de comparaison, il est estimé que l'univers observable ne contient que 10^{79} atomes [Planck Collaboration et al., 2016].

Une autre information sur laquelle un agent peut se baser pour prendre des décisions est l'*état de croyance*.

3.3.2 État de croyance

Comme un agent dans un POMDP n'observe pas directement l'état du système, il pourrait être pratique pour lui d'estimer l'état *le plus probable* en fonction de son historique d'observations et ainsi se considérer comme dans un modèle totalement observable. Il pourrait de cette façon se baser sur une politique de MDP comme décrite dans la définition 3.2.2 pour prendre ses décisions. Malheureusement, cette solution – choisir l'état le plus probable – présente l'inconvénient que l'agent ne prendrait pas en compte son degré d'incertitude. S'il a des chances équiprobables d'être dans n'importe quel état, alors il va plus ou moins devoir décider au hasard. Au contraire, le fait de *ne pas savoir* devrait l'inciter, par exemple, à utiliser des actions de perception ou de détection de son environnement, qui pourraient le renseigner sur l'état réel du système.

Une statistique suffisante pour la prise de décision optimale dans un POMDP est l'**état de croyance** [Bertsekas et al., 2005], défini, pour tout état $s \in \mathcal{S}$ et à tout instant t, par :

$$b^t(s) = Pr(s^t = s). \tag{3.17}$$

$b^t(s)$ est la probabilité d'être dans l'état $s \in \mathcal{S}$ à l'instant t. L'état de croyance b^t est une distribution de probabilité sur les états, c'est-à-dire

$$\sum_{s \in \mathcal{S}} b^t(s) = 1. \tag{3.18}$$

L'état de croyance b^t peut être vu comme un vecteur de taille $|\mathcal{S}|$. On note $\mathcal{B}$ l'ensemble des états de croyance. Le fait que l'état de croyance soit une statistique suffisante signifie qu'il contient toute l'information possible sur l'état réel du système, et qu'aucune donnée (actions et observations passées) n'apporterait d'information supplémentaire sur l'état réel du système [Åström, 1965]. Un agent n'a pas besoin de stocker l'historique des observations qu'il fait, il lui

suffit de mettre à jour son état de croyance à chaque instant t. Après avoir agi avec l'action $a^t \in \mathcal{A}$ et avoir observé $o^{t+1} \in \mathcal{O}$, le nouveau degré de croyance d'être dans un état $s^{t+1} \in \mathcal{S}$ à l'instant $t+1$ est :

$$\begin{aligned} b^{t+1}(s^{t+1} \mid a^t,o^{t+1}) &= Pr(s^{t+1} \mid b^t,a^t,o^{t+1}) \\ &= \frac{O(s^{t+1},a^t,o^{t+1}) \sum_{s^t \in \mathcal{S}} T(s^t,a^t,s^{t+1}) b^t(s^t)}{Pr(o^{t+1} \mid a^t,b^t)} \end{aligned} \tag{3.19}$$

où le dénominateur

$$Pr(o^{t+1} \mid a^t,b^t) = \sum_{s^{t+1} \in \mathcal{S}} O(s^{t+1},a^t,o^{t+1}) \sum_{s^t \in \mathcal{S}} T(s^t,a^t,s^{t+1}) b^t(s^t) \tag{3.20}$$

est un facteur de normalisation, nécessaire pour s'assurer que l'état de croyance b^{t+1} soit une distribution de probabilité valide. Après chaque nouvelle action et observation, l'agent met à jour son état de croyance en appliquant l'opération de mise à jour via l'équation 3.19 pour chaque état $s \in \mathcal{S}$. L'**estimateur d'état**, noté SE (pour *State Estimator*), est l'opérateur appliquant l'opération de mise à jour de la croyance pour chaque état. Le nouvel état de croyance $b^{t+1} \in \mathcal{B}$ est ainsi noté

$$b^{t+1} = (b^{t+1}(s) \mid \forall s \in \mathcal{S}) = SE(b^t,a^t,o^{t+1}). \tag{3.21}$$

Il s'agit de la croyance mise à jour d'être dans chaque état après avoir fait une observation.

⁎ **Théorème de Bayes.** Il est possible de reconnaître dans l'équation 3.19 une application directe du théorème de Bayes, résultat essentiel de la théorie des probabilités [Bayes and Price, 1763], et qui peut être énoncé comme suit. Soit un espace probabilisé défini sur un ensemble d'évènements $\mathcal{E}$. Pour deux évènements $A,B \in \mathcal{E}$, la probabilité de A sachant B est

$$Pr(A \mid B) = \frac{Pr(B \mid A) Pr(A)}{Pr(B)}. \tag{3.22}$$

Illustrons le théorème de Bayes avec un exemple simple d'enquête policière. Supposons que M. X a été retrouvé mort dans sa chambre d'hôtel le lundi matin par le service d'étage. La thèse du meurtre est très vite avancée par les enquêteurs, et les soupçons se tournent vers Mme Y, une soi-disant amie de la victime. M. Z, un client de l'hôtel où séjournait M. X et qui passait la nuit dans une chambre voisine, dit avoir un témoignage à faire aux enquêteurs. On peut étiqueter les deux évènements suivants :

A) Mme Y est l'auteur du meurtre, et

B) M. Z a vu sortir Mme Y de l'appartement de M. X la veille de la découverte du corps.

Dans le théorème de Bayes, la probabilité $Pr(A)$ est la probabilité **a priori** (ou **marginale**) que A se réalise, c'est-à-dire la croyance initiale que l'on a sur la réalisation de A. Ici, c'est la présomption qu'ont préalablement les enquêteurs sur la culpabilité de Mme Y dans cette affaire. De la même façon, $Pr(B)$ est la probabilité marginale que B se réalise. Dans cet exemple, c'est la probabilité que M. Z voit Mme Y sortir de la chambre de M. X, sans connaissance sur le fait qu'elle soit la meurtrière ou pas. La probabilité $Pr(B \mid A)$ est ce que l'on appelle la **fonction de vraisemblance**, et mesure la plausibilité que B se réalise effectivement si A se réalise. Dans notre cas, c'est donc la probabilité que M. Z ait effectivement été témoin du départ de Mme Y le soir du meurtre si cette dernière est effectivement la meurtrière. Enfin, $Pr(A \mid B)$ est la probabilité **a posteriori** (ou **conditionnelle**) de A sachant B, c'est-à-dire la probabilité que A

se réalise *sachant* que B s'est effectivement réalisé. Dans notre exemple, c'est la probabilité que Mme Y soit l'auteur du crime, sachant que M. Z l'a vue sortir de l'appartement de la victime le soir du meurtre. $Pr(A \mid B)$ est la croyance **mise à jour** des enquêteurs sur la culpabilité de Mme Y suite au témoignage de M. Z.

Supposons que, initialement, les enquêteurs aient peu de raisons d'accuser Mme Y, mais qu'elle soit leur seule piste : $Pr(A) = 0{,}1$. La probabilité que M. Z voit Mme Y quitter la chambre si elle est vraiment la meurtrière est très forte (sûrement a-t-il entendu un bruit qui a attiré son attention et l'a poussé à examiner le couloir de l'hôtel, et son témoignage est jugé digne de confiance par les enquêteurs) : $Pr(B \mid A) = 0{,}8$. À l'inverse, on peut supposer que la probabilité de voir la suspecte quitter la chambre est très faible si celle-ci est innocente (M. Z n'aurait aucune raison de s'enquérir de ce qu'il se passe dans la chambre voisine) : $Pr(B \mid \neg A) = 0{,}01$. On a donc :

$$\begin{aligned} Pr(A \mid B) &= \frac{Pr(B \mid A)Pr(A)}{Pr(B)} \\ &= \frac{Pr(B \mid A)Pr(A)}{Pr(B \mid A)Pr(A) + Pr(B \mid \neg A)Pr(\neg A)} \\ &= \frac{0{,}8 * 0{,}1}{0{,}8 * 0{,}1 + 0{,}01 * 0{,}9} \\ &\approx 0{,}899. \end{aligned} \tag{3.23}$$

Sous réserve de la bonne foi de M. Z, les regards des enquêteurs sont maintenant très fermement tournés vers Mme Y.

La figure 3.5 apporte une autre illustration de la mise à jour d'un état de croyance dans le cadre d'un POMDP.

Initialement, à $t = 0$, l'état de croyance est b^0. Souvent, il est supposé que b^0 est une donnée intrinsèque du POMDP, et celui-ci est alors noté par le tuple $\langle \mathcal{S},\mathcal{A},\mathcal{O},T,O,R,b^0 \rangle$. Lorsqu'on ne dispose d'aucune information *a priori* sur l'état initial du système, on peut choisir une distribution uniforme pour b^0. Si toutefois on dispose de connaissances expertes liées au domaine étudié, alors on peut donner à b^0 une forme privilégiant certains états initiaux (comme dans la figure 3.5, où l'un des états est supposé impossible à $t = 0$).

3.3.3 Belief MDP

Nous l'avons vu, on peut calculer le prochain état de croyance uniquement à partir de l'état de croyance actuel, de l'action choisie et de l'observation perçue, et ce en appliquant l'équation 3.19 pour chaque état. Ce processus de mise à jour de l'état de croyance est donc *markovien*, et l'état de croyance, au même titre que l'état dans un MDP simple, constitue un **signal markovien**, une information qui est reçue par l'agent et qui possède la propriété de Markov. On peut ainsi définir une fonction de transition sur les états de croyance

$$\tau : \mathcal{B} \times \mathcal{A} \times \mathcal{B} \to [0,1]$$

qui donne, pour un état de croyance b^t et une action a^t, la probabilité de transiter vers un nouvel état de croyance b^{t+1} :

$$\tau(b^t,a^t,b^{t+1}) = \sum_{o^{t+1} \in \mathcal{O}} Pr(o^{t+1} \mid a^t,b^t) Pr(b^{t+1} \mid b^t,a^t,o^{t+1}), \tag{3.24}$$

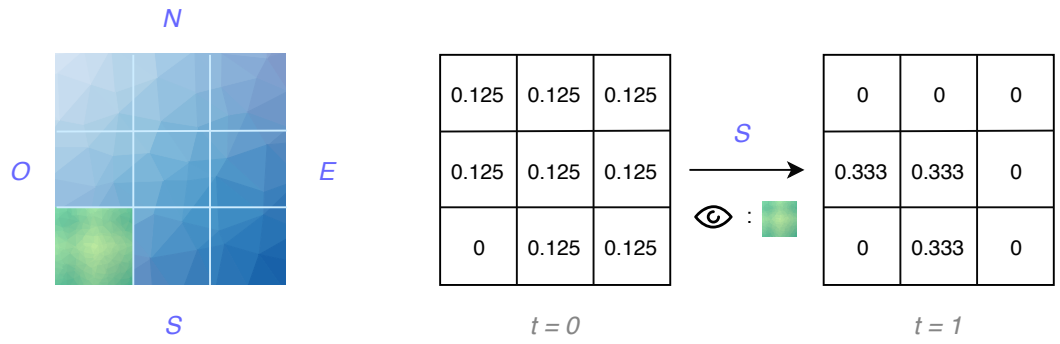


FIGURE 3.5 – Exemple du processus de mise à jour de l'état de croyance. Une frégate voguant vers les Caraïbes s'est perdue en mer et son équipage se retrouve bien embêté. La grille de gauche représente la carte vue de dessus sur laquelle peut se déplacer le navire. Chaque cellule est une position possible. La cellule Terre (■), en bas à gauche, représente la terre ferme, le nouveau continent que le capitaine du navire cherche à atteindre. Le capitaine, l'agent, dispose de quatre actions : $\mathcal{A} = \{ N, E, S, O \}$, correspondant aux points cardinaux vers lesquels il peut se diriger. Lorsque le navire se trouve sur une des trois cases adjacentes à la cellule Terre, il fait l'observation *Terre en vue* (■). Sinon, il fait l'observation *Rien en vue*. L'état du système correspond à la position du navire sur l'une des cellules de la grille. En supposant que le capitaine sait que son navire se trouve perdu en mer, il y a huit positions initiales possibles pour le navire. Cela est représenté sur la grille centrale, où, à $t = 0$, chaque cellule contient la même probabilité initiale de contenir le navire. En effectuant l'action S (Aller au Sud), et en faisant l'observation *Terre en vue*, le capitaine met à jour son état de croyance à $t = 1$, représenté sur la grille de droite. Les états impossibles sont éliminés (probabilité à 0). Les trois seuls états possibles sont ceux directement adjacents à la cellule Terre et, sous réserve que les actions et observations soient déterministes, ils sont tous trois équiprobables.

où $Pr(b^{t+1} \mid b^t,a^t,o^{t+1})$ est défini par :

$$Pr(b^{t+1} \mid b^t,a^t,o^{t+1}) = \begin{cases} 1 & \text{si } b^{t+1} = SE(b^t,a^t,o^{t+1}) \\ 0 & \text{sinon.} \end{cases} \tag{3.25}$$

Remarquons que $\tau(b^t,a^t,b^{t+1})$ vaut toujours soit 0 soit 1 : τ est déterministe. De la même façon, on peut définir une fonction de récompense

$$\rho : \mathcal{B} \times \mathcal{A} \to \mathrm{R}$$

qui donne la récompense pour avoir effectué une action a^t dans l état de croyance b^t :

$$\rho(b^t,a^t) = \sum_{s \in \mathcal{S}} b^t(s) R(s,a^t). \tag{3.26}$$

Avec ces deux nouvelles fonctions τ et ρ, on peut définir ce que l'on appelle un ***belief* MDP**, ou **MDP de croyance** : un MDP dans lequel les états sont des états de croyance. Le tuple

$$\langle \mathcal{B},\mathcal{A},\tau,\rho \rangle$$

est un MDP à espace d'états continu. Remarquons qu'il s'agit bel et bien d'un MDP tel que décrit en définition 3.2.1 : un espace d'états, un ensemble d'actions, une fonction de transition

sur les états et une fonction de récompense. L'intérêt de considérer un *belief* MDP plutôt que le POMDP de base est qu'il a été démontré que la solution de l'un est également celle de l'autre. Autrement dit, il suffit de calculer une politique optimale dans le *belief* MDP pour obtenir une politique optimale pour le POMDP [Åström, 1965].

Définition 3.3.4 - *Politique déterministe d'un belief MDP* : Soit M un *belief* MDP. Une politique déterministe π pour un agent dans le *belief* MDP M est une fonction

$$\pi : \mathcal{B} \rightarrow \mathcal{A}.$$

Une politique est une application des états de croyance vers l'ensemble des actions. On peut calculer la valeur d'une politique de *belief* MDP de façon similaire à ce que l'on a fait pour la valeur d'une politique d'un MDP (équation 3.9) :

$$V^{\pi}(b^t) = \rho(b^t,\pi(b^t)) + \gamma \sum_{o^{t+1} \in \mathcal{O}} Pr(o^{t+1} \mid \pi(b^t),b^t) V^{\pi}(b^{t+1}), \tag{3.27}$$

où $b^{t+1} = SE(b^t,\pi(b^t),o^{t+1})$ est l'état de croyance suivant.

Un *belief* MDP étant un MDP, il serait tentant de se dire qu'on peut appliquer un algorithme comme *Value Iteration* pour le résoudre – et résoudre par là-même le POMDP initial. Malheureusement, il n'est pas trivial d'appliquer *Value Iteration* lorsque l'espace d'états est continu. En effet, cet algorithme met à jour de façon itérative la valeur en chaque état. On comprend donc le problème qu'il en résulte lorsque le nombre d'états est infini.

Heureusement, la fonction de valeur V^* de la politique optimale d'un *belief* MDP possède la propriété d'être **convexe** et **linéaire par partie** (*Piecewise Linear and Convex*, PWLC) [Smallwood and Sondik, 1973]. Plus précisément, V^* correspond à l'enveloppe supérieure d'un ensemble d'hyperplans définis sur l'espace des états de croyance, où chaque hyperplan correspond à la fonction de valeur d'une politique. La figure 3.6 illustre cette propriété.

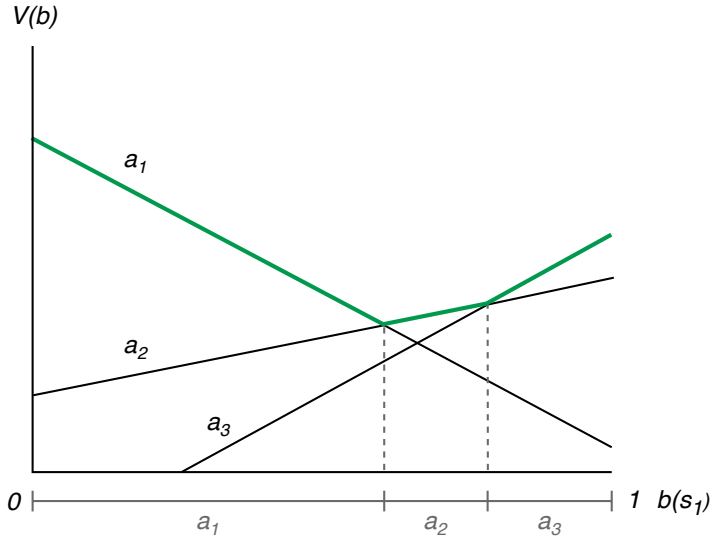


FIGURE 3.6 – Illustration de la propriété PWLC de la fonction de valeur optimale d'un POMDP, ici avec trois actions, deux états, à horizon 1.

Considérons un POMDP simple à horizon 1, ayant deux états $\mathcal{S} = \{s_1,s_2\}$ et trois actions $\mathcal{A} = \{a_1,a_2,a_3\}$. L'état de croyance correspond au couple $(b(s_1),b(s_2))$, où $b(s_i)$ est la probabilité d'être dans l'état $s_i \in \mathcal{S}$. Comme b constitue une distribution de probabilité valide, alors il suffit de considérer $b(s_1)$ pour connaître parfaitement l'état de croyance (en effet, $b(s_2) = 1 - b(s_1)$). On peut ainsi représenter cet état de croyance par le segment [0,1] (l'axe des abscisses sur la figure 3.6) où 0 correspond à une probabilité nulle d'être dans s_1 (donc une probabilité certaine d'être dans s_2) et 1 correspond à une probabilité certaine d'être dans s_1 (donc une probabilité nulle d'être dans s_2). Si l'on considère un problème à horizon 1, alors l'agent ne peut choisir qu'une seule action $a \in \mathcal{A}$. Supposons qu'il effectue l'action a_1. La valeur espérée pour cette politique à horizon 1 est donc de façon triviale la récompense moyenne immédiate que l'agent peut obtenir dans cet état de croyance en effectuant l'action a_1 :

$$V^{a_1}(b) = \rho(b,a_1) = \sum_{s \in S} b(s)R(s,a_1) = b(s_1)R(s_1,a_1) + b(s_2)R(s_2,a_1).$$

La valeur de l'action a_1 est donc linéaire selon l'état de croyance b. Cela est représenté par une droite sur la figure 3.6. On procède de façon similaire pour les actions a_2 et a_3. La valeur de la politique optimale correspond au maximum des valeurs des actions en chaque point de l'état de croyance. Cela forme l'enveloppe convexe surlignée en gras sur la figure 3.6. L'état de croyance est ainsi partitionné en sous-segments, et chaque segment indique l'action optimale à effectuer dans cette partie de l'état de croyance. La propriété de convexité s'explique alors naturellement. Plus on se trouve au centre de l'état de croyance ($b(s_1) \approx 0{,}5$) moins on a de certitude sur l'état réel du système, moins l'action optimale à choisir est certaine, et donc plus la récompense espérée est faible. Inversement, plus on se rapproche des bords de l'état de croyance (0 et 1), plus on est sûr de l'état réel et donc plus à même de choisir une action optimale : c'est là que l'enveloppe est la plus haute.

Ce raisonnement se généralise à des horizons plus grands que 1 et pour des espaces à plus de deux états. Lorsque le système possède plus de deux états, les fonctions de valeurs sont représentées par des hyperplans plutôt que de simples droites, et pour des horizons supérieurs à 1, ces hyperplans correspondent aux valeurs espérées de politiques plutôt qu'aux valeurs espérées de simples actions. De nombreux algorithmes exploitant la propriété PWLC de la fonction de valeur optimale ont ainsi été développés [Littman et al., 1995a,Cassandra et al., 1997,Cassandra, 1998b].

3.3.4 Complexité

Résoudre un POMDP à horizon fini est un problème PSPACE-complet [Papadimitriou and Tsitsiklis, 1987], et résoudre un POMDP à horizon infini est un problème indécidable [Madani et al., 1999]. Toutefois, tout n'est en réalité pas si noir. Les progrès technologiques et les avancées algorithmiques ont permis de résoudre des POMDPs avec des centaines d'états en une poignée de secondes [Hsu et al., 2007], mais également des POMDPs avec plusieurs centaines de milliers d'états [Hoey et al., 2010].

Le lecteur est invité à se référer à [Kaelbling et al., 1998] et [Spaan, 2012] pour des introductions approfondies au sujet des POMDPs. De nombreux exemples d'applications des POMDPs sont présentés dans [Cassandra, 1998a]. Pour une revue de quelques algorithmes de résolution de POMDP, on peut se référer à [Ross et al., 2008] (algorithmes *online*) et [Shani et al., 2013] (algorithmes *point-based*). Parmi les algorithmes ayant rencontré le plus de succès, on peut citer $HSVI$ (*Heuristic Search Value Iteration*) [Smith and Simmons, 2004] et $PBVI$ (*Point-Based Value Iteration*) [Pineau et al., 2006]. Un autre exemple d'algorithme de résolution récent est

décrit dans [Silver and Veness, 2010] et présente une approche basée sur les méthodes de Monte-Carlo, relativement semblable à ce qui sera présenté dans la suite de cette thèse. Enfin, [Ferrari, 2017] propose de discrétiser l'espace des états de croyance afin de pouvoir calculer des politiques approximatives (seulement localement optimales) mais efficaces lorsqu'il s'agit d'être implémentées sur de vrais systèmes robotiques.

Comme nous allons le voir maintenant, la définition d'état de croyance telle que décrite dans cette section ne constitue plus une statistique suffisante à un agent pour la prise de décision optimale lorsqu'on considère une exécution *décentralisée* – c'est-à-dire, avec plusieurs agents. En effet, un agent ne doit alors plus uniquement considérer son incertitude sur l'état du système, mais également son incertitude sur le comportement (les actions) et les observations faites par les autres agents dans le système.

3.4 Contrôle décentralisé

Nous en venons à présent au cœur de notre étude sur les processus décisionnels. Jusqu'à présent, nous n'avons examiné que les cas où un seul et unique agent prend des décisions, influençant ainsi l'évolution du système. Que se passe-t-il à présent si plusieurs agents agissent de concert ? A priori, il n'y a rien de fondamentalement différent. Plusieurs agents qui exécutent des actions, c'est comme une machine industrielle ayant une unité de calcul unique mais de multiples bras mécaniques qui exécutent plusieurs actions en même temps pour usiner une pièce (perçage, rotation, refroidissement, etc., tout cela en même temps). C'est là qu'intervient le terme **décentralisé**. Contrairement au terme *distribué* (couramment rencontré dans le domaine de l'informatique parallèle par exemple), parler de contrôle décentralisé implique qu'un agent choisit son action individuellement, sans rendre de comptes aux autres agents, et sans savoir quelles actions eux-mêmes ont choisies – comme si chaque bras mécanique disposait de sa propre capacité de calcul et de prise de décision. Un DEC-POMDP est un modèle théorique de décision qui permet un tel contrôle décentralisé. Comme il s'agit d'un domaine riche, la section actuelle n'est pas exhaustive dans sa description et son explication du modèle. Le lecteur est invité à se rapporter à [Oliehoek and Amato, 2016] pour une introduction rapide au domaine des DEC-POMDPs et au livre en français [PDMIA, 2008] pour une revue complète des processus décisionnels de Markov utilisés en intelligence artificielle.

3.4.1 Définition du modèle

Un **processus décisionnel de Markov décentralisé et partiellement observable** [Bernstein et al., 2000] (*Decentralized Partially Observable Markov Decision Process*, **DEC-POMDP**, aussi appelé POMDP décentralisé) est une généralisation du modèle de POMDP au cas multi-agents.

Définition 3.4.1 - *Processus décisionnel de Markov décentralisé et partiellement observable* : Un DEC-POMDP est un tuple

$$(\mathcal{N},\mathcal{S},\mathcal{A},\mathcal{O},T,O,R,b^0,h),$$

où

— $\mathcal{N} = \{1,2,\ldots,n\}$ est l'ensemble fini des $n \in \mathbb{N}$ agents du système ;

— $\mathcal{S}$ est l'ensemble fini des états du système ;

— $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ est l'ensemble des actions jointes, et $\mathcal{A}_i$ est l'ensemble fini des actions individuelles de l'agent $i \in \mathcal{N}$;

— $\mathcal{O} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_n$ est l'ensemble des observations jointes, et $\mathcal{O}_i$ est l'ensemble fini des observations individuelles de l'agent $i \in \mathcal{N}$;

— $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ est la fonction de transition du système, telle que $T(s^t,a^t,s^{t+1}) = Pr(s^{t+1} \mid s^t,a^t)$ est la probabilité d'arriver dans l'état $s^{t+1} \in \mathcal{S}$ une fois que les agents ont effectué l'action jointe $a^t \in \mathcal{A}$ depuis l'état $s_t \in \mathcal{S}$;

— $O : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \times \mathcal{S} \to [0,1]$ est la fonction d'observation du système, telle que $O(s^t,a^t,o^{t+1},s^{t+1}) = Pr(o^{t+1} \mid s^t,a^t,s^{t+1})$ est la probabilité que les agents fassent l'observation jointe $o^{t+1} \in \mathcal{O}$ quand le nouvel état est s^{t+1} une fois que les agents ont effectué l'action jointe $a^t \in \mathcal{A}$ depuis l'état $s^t \in \mathcal{S}$;

— $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ est la fonction de récompense, telle que $R(s^t,a^t,s^{t+1})$ est la récompense obtenue après que les agents ont effectué l'action jointe $a^t \in \mathcal{A}$ depuis l'état $s^t \in \mathcal{S}$ et que le système a transité dans l'état $s^{t+1} \in \mathcal{S}$;

— b^0 est l'état de croyance initial, une distribution de probabilité sur les états de $\mathcal{S}$;

— h est l'horizon de planification.

Il arrive de rencontrer dans la littérature des définitions où les fonctions de transition T et d'observation O sont représentées dans une seule et même fonction, dite **fonction de dynamique**, notée $D : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \times \mathcal{S}$ telle que $D(s^t,a^t,o^{t+1},s^{t+1}) = Pr(s^{t+1},o^{t+1} \mid s^t,a^t) = T(s^t,a^t,s^{t+1})O(s^t,a^t,o^{t+1},s^{t+1})$ est la probabilité d'arriver dans l'état $s^{t+1} \in \mathcal{S}$ et que les agents fassent l'observation jointe $o^{t+1} \in \mathcal{O}$ une fois que les agents ont effectué l'action jointe $a^t \in \mathcal{A}$ depuis l'état $s_t \in \mathcal{S}$.

Un DEC-POMDP est une extension naturelle d'un POMDP. Si l'on fixe la taille de la population des agents à $|\mathcal{N}| = 1$, les deux définitions 3.3.1 et 3.4.1 coïncident parfaitement.

À un instant t, chaque agent $i \in \mathcal{N}$ choisit une action individuelle $a_i^t \in \mathcal{A}_i$, formant ainsi une **action jointe** $a^t = (a_1^t,\ldots,a_n^t)$. Chaque agent exécute son action simultanément, faisant ainsi transiter le système d'un état s^t à un état s^{t+1}. Une **observation jointe** $o^{t+1} = (o_1^{t+1},\ldots,o_n^{t+1})$ est émise, et chaque agent i reçoit sa propre composante individuelle $o_i^{t+1} \in \mathcal{O}_i$. De plus, l'action jointe a^t génère une récompense immédiate $R(s^t,a^t,s^{t+1})$. Cette récompense, bien qu'elle ne soit pas observée par les agents, est partagée entre eux. Observer la récompense obtenue après avoir effectué une action donnerait des informations supplémentaires sur l'état réel du système, informations non contenues dans l'observation que fait l'agent, celle émise par la fonction d'observation O.

Un DEC-POMDP peut disposer de plusieurs niveaux d'observabilité [Pynadath and Tambe, 2002] :

1. **Observabilité individuelle.** L'observation individuelle d'un agent lui suffit pour déterminer totalement l'état réel du système :

$$\forall i \in \mathcal{N},\ \forall o_i \in \mathcal{O}_i,\ \exists s \in \mathcal{S} \text{ tel que } O(o_i,s) = 1. \tag{3.28}$$

2. **Observabilité collective.** Une observation jointe, la réunion des observations individuelles de chaque agent, détermine totalement l'état réel du système :

$$\forall o \in \mathcal{O},\ \exists s \in \mathcal{S} \text{ tel que } O(o,s) = 1. \tag{3.29}$$

3. **Observabilité partielle.** Une observation jointe ne permet pas de déterminer totalement l'état réel du système. Aucune supposition n'est faite, c'est le cadre le plus général.

Dans le cas particulier où l'observabilité est collective, un DEC-POMDP se ramène à un **DEC-MDP** [Bernstein et al., 2000], modèle permettant de capturer les cas où l'observation jointe des agents identifie de façon unique l'état réel du système. Toutefois, sans hypothèse *a priori* sur le problème modélisé par le DEC-POMDP, on supposera dans la plupart des cas que l'observabilité est partielle. Le lecteur peut se référer à [Goldman and Zilberstein, 2004] pour de plus amples détails concernant les différents types d'observabilité dans DEC-POMDPs.

Une fois le processus exécuté à horizon h, un agent $i \in \mathcal{N}$ *ne reçoit pas* de part individuelle pour avoir effectué une série d'actions dans le système et qui serait une fraction de toutes les récompenses accumulées. Les agents sont coopératifs et agissent dans un but commun. Leur récompense personnelle – leur satisfaction, en quelque sorte – est grande si, et seulement si, la récompense de l'équipe est grande. Tout sportif professionnel le dira : un prix collectif (*Équipe championne*) a plus de valeur qu'un prix individuel (*Meilleur joueur*) [1]. C'est ce qui rend la coordination primordiale : un agent ne peut tout simplement pas maximiser ses propres récompenses à long terme, mais doit plutôt se coordonner avec ses coéquipiers pour maximiser une récompense commune. Cela s'avère difficile car cela implique que les agents doivent maximiser une récompense *globale* uniquement sur la base de perceptions et d'actions *locales*. En l'absence de communication durant l'exécution, cela requiert donc que l'étape de planification permettant de synchroniser les agents ait lieu de façon centralisée, en amont de l'exécution. Un peu à la manière d'un coach donnant des instructions à ses joueurs avant le début d'un match, avant de les laisser se débrouiller sur le terrain une fois le match commencé.

Mais pourquoi ne pas autoriser les communications entre agents ? Si toutes les informations concernant les observations perçues et les actions prises étaient partagées entre les agents, on se retrouverait dans le cas – relativement – simple d'un POMDP. Dans certaines situations, les communications directes sont coûteuses ou ne sont tout simplement pas possibles. C'est souvent le cas dans le domaine aérospatial. Établir un système de communication entre deux astromobiles (*rovers*) martiens (comme *Spirit* et *Opportunity*, lancés au début des années 2000) demanderait un important travail d'ingénierie puisque qu'un astromobile ne dispose que d'une puissance réduite, limitée par la surface et l'efficacité de ses panneaux solaires. Dans la pratique, les seuls systèmes de communication qui ont été implémentés l'ont été pour des communications vers la Terre, à un débit de quelques Kbit/s, pendant une durée limitée chaque jour. Dans d'autres situations, les communications peuvent être indésirables. Dans le domaine militaire, l'existence de communications implique le risque d'interception de ces communications, soit par des factions tierces soit par des civils, cela pouvant mener à de graves brèches de sécurité.

Un agent dans un DEC-POMDP ne peut donc pas compter sur ses coéquipiers pour obtenir des informations sur le système. Tout ce dont il dispose, ce sont les observations locales qu'il fait.

3.4.2 Historiques et politiques

Nous avons vu dans la section précédente que l'état de croyance (une distribution de probabilité sur les états) était une statistique suffisante pour la prise de décision optimale dans un POMDP. Ce n'est pas le cas pour un POMDP décentralisé car un agent ne connaît pas les observations reçues et les actions jointes sélectionnées par les autres agents.

Une statistique suffisante pour la prise de décision optimale dans un DEC-POMDP est l'historique des observations faites par un agent [Oliehoek et al., 2008].

1. Selon Luka Modric, milieu de terrain et capitaine de l'équipe de football de Croatie, finaliste perdant de la Coupe du monde de football 2018 et élu meilleur joueur de la compétition.

Définition 3.4.2 - *Historique d'observations individuel* : Soit M un DEC-POMDP. Un *historique d'observations individuel* $\vec{o}_i^t$ pour un agent $i \in \mathcal{N}$ est une séquence finie de t observations reçues par cet agent depuis le début du processus :

$$\vec{o}_i^t = (o_i^1, o_i^2, \ldots, o_i^t). \tag{3.30}$$

On note $\vec{\mathcal{O}}_i^t$ l'ensemble des historiques d'observations individuels de longueur t de l'agent i et $\vec{\mathcal{O}}_i$ l'ensemble de tous les historiques d'observations individuels de cet agent. Sauf mention contraire, on utilisera en général la notation $\vec{o}_i \in \vec{\mathcal{O}}_i$. Un *historique d'observations joint* $\vec{o} = (\vec{o}_1, \vec{o}_2, \ldots, \vec{o}_n)$ est défini comme la collection de tous les historiques d'observations individuels, un par agent. Comme pour les historiques d'observations individuels, on note $\vec{o}^t \in \vec{\mathcal{O}}^t$ et $\vec{o} \in \vec{\mathcal{O}}$. Finalement, on note $\vec{o}_{-i} = (\vec{o}_1, \vec{o}_2, \ldots, \vec{o}_{i-1}, \vec{o}_{i+1}, \ldots, \vec{o}_n)$ l'historique d'observations joint de tous les agents sauf de l'agent i.

Définition 3.4.3 - *Politique individuelle* : Soit M un DEC-POMDP. Une *politique déterministe individuelle* π_i pour un agent $i \in \mathcal{N}$ du DEC-POMDP M est une fonction

$$\pi_i : \vec{\mathcal{O}}_i \to \mathcal{A}_i.$$

Une politique pour un agent $i \in \mathcal{N}$ est donc une fonction des historiques d'observations vers l'ensemble de ses actions. Après avoir fait la séquence d'observations $\vec{o}_i^t = (o_i^1, o_i^2, \ldots, o_i^t) \in \vec{\mathcal{O}}_i$, l'agent i prend la décision d'agir avec l'action $\pi_i(\vec{o}_i^t)$. Une **politique jointe** $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ est une collection de politiques individuelles, une par agent. On note Π_i l'ensemble de toutes les politiques individuelles d'un agent $i \in \mathcal{N}$, et Π l'ensemble de toutes les politiques jointes. Comme pour les historiques d'observations joints, on adopte la notation $\pi_{-i} = (\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n)$ pour désigner la politique jointe de tous les agents sauf de l'agent i.

Une politique jointe peut également être vue comme une fonction des historiques d'observations joints vers l'ensemble des actions jointes

$$\pi : \vec{\mathcal{O}} \to \mathcal{A}.$$

Notons en revanche que toutes les applications de cette forme ne sont pas des politiques jointes séparables. Une politique jointe est dite **séparable**, ou **décentralisée**, si la même action individuelle a_i est prescrite à un agent i lorsqu'il fait l'historique d'observations $\vec{o}_i$, et ce quels que soient les historiques d'observations des autres agents (ce qui est équivalent à dire que l'on peut représenter une politique jointe sous la forme d'une collection de politiques individuelles). Cette notion de séparabilité est importante car elle garantit que la politique jointe pourra effectivement être exécutée de façon décentralisée par les agents une fois l'étape de planification terminée.

De par la nature séquentielle des historiques d'observations, il est plus pratique de voir la politique d'un agent sous la forme d'une structure arborescente plutôt que d'une simple application d'un ensemble à un autre.

La figure 3.7 présente un exemple d'arbre de politique individuelle dans le cadre d'un DEC-POMDP. Chaque nœud correspond à une action et chaque arc à une observation. La profondeur de l'arbre est égale à l'horizon de planification du DEC-POMDP. Chaque niveau de l'arbre est un pas de temps, une étape de décision où l'agent choisit l'action correspondante à son historique d'observations. Comme un arbre correspond à la politique individuelle d'un agent, une politique jointe séparable correspond à une collection de n arbres, un par agent.

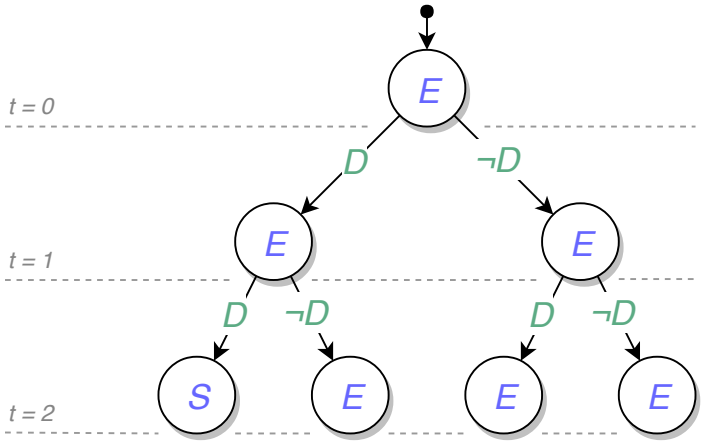


FIGURE 3.7 – Un exemple de politique individuelle à horizon 3 pour un POMDP décentralisé, représentée par un arbre, avec deux actions { E, S } et deux observations { D, $\neg D$ }.

Cette fois-ci, afin d'illustrer l'aspect décentralisé et partiellement observable du modèle, reprenons l'exemple de la figure 3.2 mais considérons une équipe d'aventuriers venus ensemble explorer les ruines et travaillant de concert pour amasser le plus de richesses. Les agents ont décidé, afin de couvrir le plus de terrain possible, de se séparer et d'explorer les ruines chacun de leur côté, puis de se rejoindre à l'extérieur au bout d'un certain temps (l'horizon de planification). Les ruines sont pleines de dangers inconnus des agents. Ces derniers ne peuvent que faire des observations partielles de leur environnement : un agent fait l'observation *Danger* (D) lorsqu'il sent qu'un danger est proche de lui (bruit inquiétant, tâches et ossements suspects, etc.). Sinon, l'agent n'observe pas de danger ($\neg D$). Notons bien que faire l'observation D ne signifie pas que l'agent se trouve assurément dans l'état *En danger* (comme c'est souvent le cas dans ce genre de situations, on ne se rend compte du danger que trop tard...) L'agent dispose de ses deux actions *Explorer* (E) et *Sortir* (S). La figure 3.7 donne un exemple de politique suivie par l'un des agents de l'équipe. Avant toute observation (c'est-à-dire, à son entrée dans les ruines), l'agent commence par prendre l'action E. Ensuite, s'il sent un danger (D), son côté baroudeur le fait continuer malgré tout et il exécute à nouveau l'action E. S'il observe à nouveau D, alors il cesse à présent de prendre des risques et se décide à prendre l'action S. Dans tous les autres cas, il continue avec l'action E. Notons qu'il s'agit de la politique d'un agent en particulier. Un autre agent pourra suivre une autre politique. Par exemple, un agent plus prudent pourra sortir dès qu'il sentira un danger pour la première fois. Une fois que l'horizon de planification (ici, $h = 3$) a été atteint (ou que tous les agents ont fait l'action S), on suppose que tous les agents sortent et mettent en commun les récompenses qu'ils ont accumulées.

Comme pour les deux modèles décisionnels présentés jusqu'ici, il est possible de déterminer la valeur d'une politique jointe comme étant l'espérance des récompenses accumulées par celle-ci. Formellement, la valeur d'une politique jointe π dans un état $s^t \in \mathcal{S}$ pour un historique d'observations joint $\vec{o}^t \in \vec{\mathcal{O}}$ peut être récursivement définie par

$$V^{\pi}(s^t,\vec{o}^t) = R(s^t,\pi(\vec{o}^t)) + \sum_{s^{t+1}\in\mathcal{S}} \sum_{o^{t+1}\in\mathcal{O}} Pr(s^{t+1},o^{t+1} \mid s^t,\pi(\vec{o}^t))V^{\pi}(s^{t+1},\vec{o}^{t+1}) \tag{3.31}$$

où $\vec{o}^{t+1} = \langle \vec{o}^t, o^{t+1} \rangle$ est le nouvel historique d'observations après avoir fait l'observation jointe o^{t+1}, et où

$$Pr(s^{t+1},o^{t+1} \mid s^t,a^t) = T(s^t,a^t,s^{t+1})O(s^t,a^t,o^{t+1}) \tag{3.32}$$

est une notation factorisée entièrement définie par les éléments du modèle du DEC-POMDP.

La valeur de la politique jointe est finalement obtenue en calculant une moyenne pondérée de la valeur en chaque état initial possible, lorsqu'aucune observation ($\varnothing$) n'a encore été faite :

$$V(\pi) = \sum_{s^0 \in \mathcal{S}} V^{\pi}(s^0, \varnothing) b^0(s^0). \tag{3.33}$$

L'équation 3.31 est fondamentalement très proche de l'équation 3.9 donnant la valeur de la politique d'un MDP. Les seules différences sont les suivantes :

— la fonction de valeur $V^{\pi} : \mathcal{S} \times \vec{\mathcal{O}} \to \mathbb{R}$ est définie sur les états mais également sur les historiques d'observations joints (car la politique jointe est elle-même définie sur ces historiques), et ;

Notons finalement que, afin d'avoir une définition récursive valide de la fonction de valeur, on a

$$V^{\pi}(s^h, \vec{o}^h) = 0. \tag{3.34}$$

Résoudre un DEC-POMDP consiste à trouver une politique jointe séparable π^* telle que $V(\pi^*)$ soit maximale :

$$\pi^* = \operatorname*{argmax}_{\pi} V(\pi). \tag{3.35}$$

3.4.3 Complexité

Bien que le modèle de DEC-POMDP soit particulièrement adapté à la planification multi-agents sous incertitude, il souffre d'une complexité algorithmique notoire qui empêche son utilisation pratique dans le domaine de l'industrie ou d'applications réelles et à grande échelle. Intuitivement, la complexité d'un DEC-POMDP vient du fait qu'un agent doit raisonner sur l'incertitude qu'il a du comportement des autres agents, en plus de son incertitude sur l'état du système et sur les effets de ses actions (comme dans un POMDP). Les recherches menées dans le domaine de la théorie des jeux ont montré qu'il n'existe pas de critère d'optimalité qu'un agent puisse maximiser sans avoir à tenir compte du comportement des autres agents. Autrement dit, dire qu'une politique individuelle est meilleure qu'une autre dépend des politiques suivies par les autres agents [Szer et al., 2005].

Plus formellement, trouver une politique optimale pour un DEC-POMDP à horizon fini avec deux agents ou plus est un problème NEXP-complet [Bernstein et al., 2000]. La classe de complexité NEXP regroupe les problèmes qui peuvent se résoudre sur une machine de Turing non déterministe en temps $O(2^{p(n)})$, où $p(n)$ est un polynôme, et n est la taille de l'entrée du problème étudié (dans le cas d'un DEC-POMDP, il s'agit du nombre d'agents, des tailles des espaces d'états, d'actions, d'observations, et de la longueur de l'horizon de planification). Une solution pour un DEC-POMDP peut donc être vérifiée en temps exponentiel. Notons également que c'est le nombre d'agents et la prise de décision décentralisée, plus que l'observabilité partielle, qui sont responsables de la complexité du problème. En effet, résoudre un DEC-MDP, cas particulier des DEC-POMDPs où l'état du système est conjointement observable, est également un problème de la classe de complexité NEXP lorsque l'on considère plus de deux agents.

⁎ **Jeux stochastiques partiellement observables.** Les jeux stochastiques partiellement observables [Hansen et al., 2004] (*Partially Observable Stochastic Games*, **POSG**) constituent la classe la plus générale des modèles utilisés dans le domaine de la théorie des jeux. La plupart des modèles de décision existants (MDP, POMDP, DEC-MDP, DEC-POMDPs, jeux répétés, jeux stochastiques, etc.) sont tous des spécialisations d'un POSG. Dans ce type de jeu, chaque

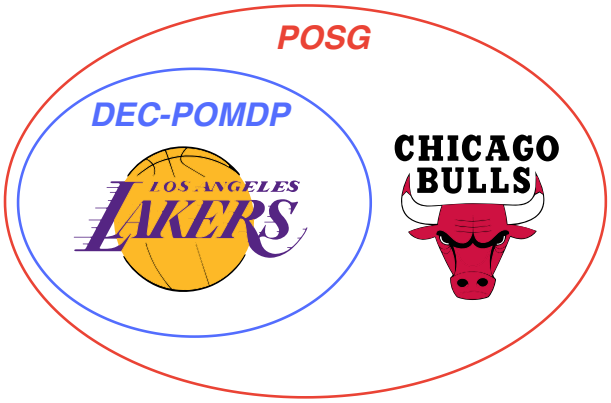


FIGURE 3.8 – Un DEC-POMDP est un cas particulier de POSG, où tous les agents sont coopératifs. On peut étudier le problème d'un match de basket-ball du point de vue des *Lakers* seulement. Dans ce cas-là, on se retrouve face à un DEC-POMDP, et les autres agents, ceux des *Bulls*, ne sont pris en compte qu'en tant qu'éléments contraignants de l'environnement.

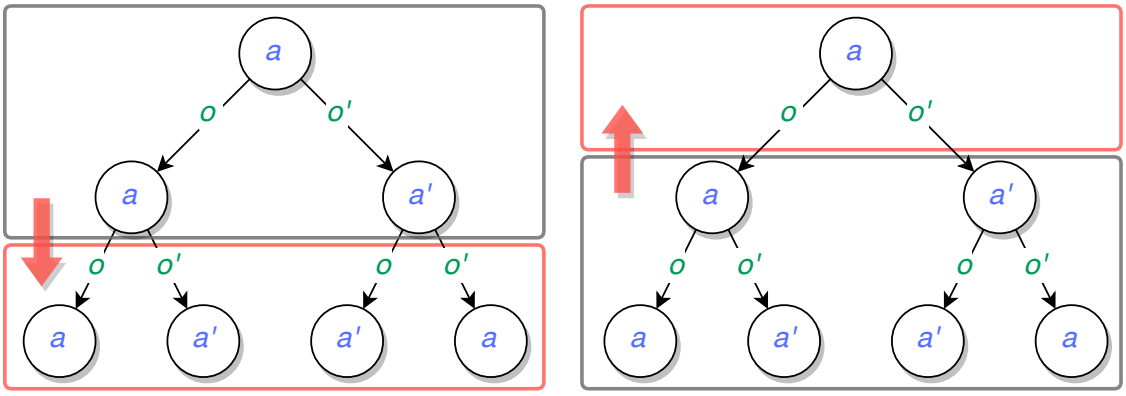
agent $i \in \mathcal{N}$ possède une fonction de récompense individuelle $R_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. Il est ainsi possible de représenter dans ce modèle des situations où les agents sont en compétition les uns avec les autres. Par exemple, on peut modéliser un match entre deux équipes en attribuant une fonction de récompense identique à une moitié des agents, et une fonction de récompense opposée à l'autre moitié des agents, comme illustré sur la figure 3.8, de sorte que les gains d'une équipe correspondent aux pertes de l'autre équipe. Les jeux stochastiques souffrent d'une complexité encore plus importante que les DEC-POMDPs (classe de complexité NEXP^{NP}).

Le sujet de cette thèse n'étant pas d'étendre ni d'améliorer les algorithmes de résolution de DEC-POMDPs existants, nous n'allons pas énumérer de façon exhaustive l'ensemble des approches et des dernières avancées algorithmiques du domaine. Nous nous contentons dans la section suivante de décrire brièvement les méthodes générales de résolution habituellement employées.

3.4.4 Méthodes de résolution

On distingue principalement trois approches lorsqu'il s'agit de résoudre un DEC-POMDP : les approches *descendantes*, les approches *ascendantes*, et les approches hybrides, qui utilisent un peu des deux autres approches. Les figures 3.9a et 3.9b illustrent le principe général de ces deux approches.

Méthodes descendantes (*top-down*). Ces algorithmes construisent les politiques des agents dans le sens séquentiel normal, c'est-à-dire, de l'instant $t = 0$ jusqu'à l'instant $t = h - 1$. Il s'agit principalement de méthodes heuristiques. L'algorithme *Multiagent A** (MAA*) [Szer et al., 2005] est une méthode de résolution optimale de DEC-POMDP qui se calque sur le célèbre algorithme A* pour explorer l'espace des politiques jointes séparables. L'algorithme construit un arbre de recherche où un nœud à la profondeur t correspond à une politique jointe séparable d'horizon t, notée δ^t. Chaque itération de l'algorithme évalue les feuilles de l'arbre et étend celle ayant la meilleure évaluation. Cette évaluation correspond à la somme de la valeur de la politique jointe δ^t, connue et donc calculable, et de la valeur de la politique jointe restante, notée Δ^{h-t}, qui est quant à elle inconnue. L'algorithme se base sur des heuristiques admissibles (c'est-à-dire des surestimations optimistes) pour estimer la valeur de Δ^{h-t}. Par exemple, il est possible



(a) Approche descendante. (b) Approche ascendante.

FIGURE 3.9 – Constructions descendantes et ascendantes d'une politique individuelle pour un DEC-POMDP avec deux actions { a, a' } et deux observations { o, o' }. Encadrés en rouge, les nœuds et arcs nouvellement créés. Encadrés en gris, les anciens nœuds et arcs.

d'utiliser la valeur du MDP sous-jacent (en considérant donc une version centralisée, totalement observable, du problème initial) pour obtenir une surestimation de la valeur réelle. L'algorithme élague ainsi l'arbre de recherche à chaque itération et construit petit à petit une politique jointe optimale. L'algorithme est très vite limité par la complexité du problème, c'est pourquoi plusieurs améliorations ont par la suite été proposées, parmi lesquelles *Generalized*-MAA* [Oliehoek et al., 2008] puis *Generalized*-MAA* *with incremental clustering and expansion* [Oliehoek et al., 2013].

Méthodes ascendantes (*bottom-up*). Ces algorithmes partent du dernier instant $t = h-1$, lorsqu'il ne reste plus qu'une seule action à effectuer, et construisent les politiques des agents en remontant jusqu'à $t = 0$. Les algorithmes de ce genre sont basés sur le principe de programmation dynamique tel que présenté dans la section 3.2.4. Le premier algorithme de cette catégorie, DP [Hansen et al., 2004], est une application directe de ce principe. Afin de contrer l'énumération d'un nombre exponentiel de politiques individuelles à générer pour chaque agent à chaque itération de l'algorithme, celui-ci se sert de la notion de *dominance* pour identifier et éliminer (par programmation linéaire) les politiques individuelles qui, quel que soit l'état de croyance, ne pourront pas permettre d'accumuler plus de récompenses que d'autres politiques individuelles. Malgré cela, la programmation dynamique ne permet pas de véritablement passer à l'échelle (c'est-à-dire de traiter des problèmes de grande taille, avec beaucoup d'agents). L'algorithme MBDP (*Memory-Bounded Dynamic Programming*) [Seuken and Zilberstein, 2007b, Seuken and Zilberstein, 2007a] est une méthode hybride qui utilise le principe de programmation dynamique pour construire des politiques jointes, mais fixe une limite au nombre de politiques générées à chaque itération. De plus, il utilise également des méthodes heuristiques pour élaguer en amont certaines politiques.

Récemment, d'autres auteurs ont proposé de reformuler un DEC-POMDP sous la forme d'un MDP à états continus [Dibangoye et al., 2016]. Cette méthode, bien qu'elle ne pourra jamais éluder totalement la complexité intrinsèque liée à la résolution d'un DEC-POMDP, a toutefois permis des avancées significatives dans le domaine.

3.5 Conclusion

Ce chapitre a présenté les briques de bases nécessaires à la compréhension et l'étude de la planification pour différents modèles de décision. Nous avons introduit les définitions des processus décisionnels de Markov, partiellement observables et décentralisés, expliqué les notions essentielles à ces modèles et mis en lumière les techniques généralement employées pour les résoudre. Les DEC-POMDPs en particulier sont au cœur du travail de cette thèse, puisqu'ils permettent le contrôle décentralisé d'agents évoluant dans un environnement incertain. Cela répond donc à une partie du but que l'on se fixe dans cette thèse, à savoir la planification dans les systèmes multi-agents coopératifs dynamiques et stochastiques. La partie qu'il nous reste à explorer est donc l'aspect ouvert des systèmes multi-agents – autrement dit, il nous faut définir et résoudre un *DEC-POMDP ouvert*.

Comme il n'existe pas à l'heure actuelle de tel modèle, nous passons en revue dans le chapitre suivant quelques-unes des approches existantes qui prennent en compte l'aspect ouvert des systèmes multi-agents.

Chapitre 4

Équipes flexibles

Sommaire

4.1 Introduction 48
4.2 Allocation de rôles 49
4.2.1 Problème d'affectation 50
4.2.2 Modèle de décision pour l'allocation de rôles 52
4.2.3 Stratégies de ré-allocation de rôles 54
4.3 Formation de coalitions 55
4.3.1 Définitions 56
4.3.2 Propriétés 58
4.4 Contrôle partiel de systèmes multi-agents ouverts 59
4.4.1 Travail d'équipe ad-hoc 60
4.4.2 Agents non dévoués 61
4.5 Conclusion 62

« Il se trouve que je suis célèbre. Sur ça, je n'ai aucun contrôle.»

Kevin Mitnick

4.1 Introduction

Nous l'avons vu, les DEC-POMDPs constituent un cadre de travail pratique pour la planification multi-agents sous incertitude. Ce qu'il manque au modèle de DEC-POMDP, et ce qui nous intéresse dans ce travail, c'est une évolution de la population d'agents au cours de l'exécution du processus – en d'autres termes, c'est d'ajouter la propriété d'*ouverture* au modèle de DEC-POMDP. En effet, le nombre $\mathcal{N}$ d'agents est un paramètre fixe. C'est une donnée constante. En fait, la planification est conditionnée à ce nombre d'agents. La question implicitement posée est : *Quel est le meilleur comportement que peut avoir cette équipe d'agents vis-à-vis de cette tâche ?* ou bien *Comment résoudre cette tâche le plus efficacement étant donnée cette équipe d'agents ?*

Pourtant, nombre d'applications réelles suggèrent que le nombre d'agents mobilisés est amené à être modifié durant la résolution d'une tâche. Imaginons l'application suivante (relativement semblable à celle que nous avons abordée dans l'introduction de cette thèse). Une équipe de drones autonomes est dépêchée sur une zone touchée par une catastrophe naturelle. Les drones sont chargés de surveiller la zone, trouver des survivants et les guider en sécurité le cas échéant. On distingue deux modes relatifs à l'entrée ou la sortie d'un agent dans un système :

1. **Non contrôlé** : l'entrée et la sortie des agents ne sont pas contrôlées, il s'agit d'évènements imprévus, qui peuvent avoir été anticipés, mais qui n'ont pas été demandés ou requis à l'agent. Dans notre exemple :
   - suite à une collision avec un oiseau, un des drones tombe en panne. Le drone hors-service sort du système en ce sens qu'il ne peut plus assurer sa tâche de surveillance. Les drones restants, s'ils apprennent la défection inopinée du drone hors-service, vont devoir se réorganiser pour gérer la surveillance sans celui-ci ;
   - un nouveau drone, développé et déployé par un producteur différent, rejoint l'équipe déjà en place. Il entre dans le système en rejoignant le reste des drones qui vont devoir se réorganiser pour se coordonner avec lui. Son arrivée n'aurait pas pu être anticipée avec certitude par les autres drones.
2. **Contrôlé** : l'entrée et la sortie des agents sont contrôlées par le processus décisionnel. Ce sont les agents qui décident d'eux-mêmes d'entrer ou sortir – ou du moins, c'est l'ordinateur ou la personne qui a conçu la politique de cet agent qui s'attend à ce que celui-ci entre ou sorte dans certaines situations, sous certaines conditions. Quoi qu'il en soit, l'entrée ou la sortie est *planifiée* à l'avance et se fait pour optimiser la résolution de la tâche. Dans notre exemple :
   - aucun drone n'a détecté de présence humaine depuis déjà un long moment. Afin d'alléger la charge système (liaison GPS, utilisation de sa batterie, etc.) et de réduire les coûts d'utilisation, un drone décide de se retirer. Il sort du système car il décide de ne plus assurer sa tâche de surveillance. Il pourra être réaffecté à une autre mission ou simplement rester en attente ;
   - l'équipe de drone a signalé la présence d'une personne ayant besoin d'assistance immédiate. Un drone de secours spécialisé est alors immédiatement envoyé et va pouvoir prendre en charge la personne en lui fournissant du matériel de première nécessité (eau, vivres, couverture, bandages, etc.). Le drone de secours entre dans le système afin de soutenir l'équipe de drones déjà en place en apportant son aide spécialisée (qui n'est pas nécessaire lorsqu'aucun survivant n'a encore été trouvé).

La **flexibilité** d'une équipe est sa capacité à s'adapter au changement lorsque cela est nécessaire. Par exemple, les agents d'une équipe flexible vont pouvoir modifier leurs rôles, plans,

politiques ou comportements pour tenir compte du fait qu'un des membres de l'équipe est hors-service. Ou bien, une équipe flexible va se départir d'un de ses membres si l'évolution de la tâche en cours le requiert. La flexibilité d'une équipe a donc trait à la fois aux modes contrôlé et non-contrôlé des entrées et sorties.

Ce chapitre présente quelques axes de recherche relatifs aux processus de décision multi-agents qui exploitent le caractère flexible de l'équipe d'agents. Ces axes nous semblent être des points pertinents à aborder afin d'avoir une vue généralisée des recherches similaires (parfois parallèles, parfois connexes) au sujet qui nous intéresse plus particulièrement. Nous allons le voir, il n'existe pas de méthode qui traite pleinement le problème de planification pour la formation et la re-formation décentralisées d'équipes sous l'hypothèse d'observabilité partielle. Toutes les approches présentées ci-après ne se concentrent que sur un, ou quelques aspects du problème, ou alors font des hypothèses qui restreignent l'applicabilité et la généralisation de la solution proposée, par exemple en considérant des environnements localement observables ou en utilisant des méthodes centralisées. Dans ce chapitre, on examine trois axes que nous avons identifiés comme pertinents par rapport à la problématique d'équipes flexibles :

1. l'**allocation et la ré-allocation de rôles** au sein d'une équipe d'agents, où les agents se répartissent les tâches selon leur rôles et peuvent se réorganiser en cas d'imprévu,
2. la **formation de coalitions**, où l'objectif est pour un ensemble d'agents de se regrouper en coalitions (ou sous-équipes) afin d'accomplir des objectifs communs,
3. le **contrôle dans les systèmes multi-agents ouverts**, où nous traiterons les agents **ad-hoc** (agents rejoignant une équipe d'agents inconnus) et les agents **non dévoués** (agents pouvant quitter l'équipe à tout moment).

4.2 Allocation de rôles

Rendre possible les entrées et sorties d'agents dans un système multi-agents permet à l'équipe actuellement sur le terrain de s'adapter à l'évolution de la tâche. Cela est d'autant plus vrai que souvent, tous les agents ne sont pas homogènes. Ils peuvent être pourvus de capacités, de modes d'action ou de perceptions différents. Un agent peut tenir un certain *rôle* dans la résolution d'une tâche [Campbell and Wu, 2011]. Deux agents avec deux rôles différents vont aider à résoudre la même tâche, tout en agissant sur des composantes distinctes de leur environnement. Ainsi, si la tâche évolue d'une façon qui encourage l'emploi d'un certain type d'agents, ceux-ci seront prioritairement utilisés pour agir dans le système, tandis que les agents moins utiles pourront être laissés hors du système pendant une certaine durée.

La définition de rôle varie selon les sources et le domaine dans lequel le concept s'applique, tel que les sciences humaines, la sociologie, la psychologie, ou encore l'économie. Ici, nous adoptons le point de vue généralement admis dans le domaine de la robotique multi-agents que la notion de rôle est distincte de celle de politique suivie par un agent [Wooldridge et al., 1999]. Deux agents ayant des rôles similaires peuvent suivre des politiques différentes. Le rôle d'un agent le *spécialise*. Il se réfère ainsi à sa *fonction* (capacités, ressources, mode de mobilité, équipement, etc.) et non à son comportement (plan, stratégie, politique, actions choisies), bien que ce dernier soit par la suite directement influencé par le rôle. De plus, le concept de rôle tend à se confondre avec le concept de *tâche* (ou de *sous-tâche*) [Gerkey and Matarić, 2003b]. Le rôle peut être défini comme la *tâche* assignée à un agent parmi un groupe d'agents. Enfin, le rôle peut être *rigide*, c'est-à-dire inné à l'agent, et dans ce cas-là il ne peut pas changer ; ou bien le rôle peut être *flexible*, et alors il s'agit d'une propriété variable qui caractérise l'agent à un instant donné mais est susceptible d'être changé [Stone and Veloso, 1999].

Illustrons ce concept de rôle avec notre exemple fil rouge. Considérons une équipe de quatre agents : Gordie, Chris, Teddy et Vern. Gordie est l'*éclaireur* : ses connaissances de la civilisation disparue l'aident à déceler et désamorcer les pièges. Chris est le *protecteur* : sa maîtrise des armes à feu et des techniques de survie en milieux hostiles le rend plus résistant aux épreuves qu'il rencontre. Teddy est le *scientifique* : il récolte les données de la mission et prend note des découvertes qu'il fait. Enfin, Vern est le *transporteur* : il garde sur lui les trésors trouvés ainsi que le matériel nécessaire à l'expédition. Chacun des quatre agents possède un rôle qui lui est propre et qui le spécialise. Si Vern, le transporteur, tombe dans un piège, l'équipe se retrouvera fortement pénalisée, il est donc important que Gordie éclaire correctement la voie. Les données récoltées par Teddy, le scientifique, sont précieuses mais rares, difficiles à collecter. C'est donc à Chris, le protecteur, qu'il revient d'appuyer sur les dalles saillantes, d'enclencher les leviers mystérieux, de prendre les risques.

Dans cet exemple, les rôles (éclaireur, transporteur, protecteur et scientifique) sont implicitement induits par les caractéristiques, physiques ou intellectuelles, de chacun des aventuriers (érudition, endurance, survivabilité, esprit d'analyse), et les agents peuvent donc être vus comme des instances de ces rôles (de la même façon qu'en programmation orientée objet, un objet est une instance d'une classe). Cela signifie que les rôles ne sont pas nécessairement des propriétés intrinsèques des agents. Souvent, un agent peut tenir plusieurs rôles, et parfois, certains rôles ne sont accessibles qu'à certains agents. De la même façon, plusieurs agents peuvent occuper le même rôle. L'*allocation de rôles* est la tâche initiale qui consiste à donner un rôle à chaque agent avant le début du processus. Selon le contexte, il peut également s'agir de déterminer le nombre d'agents qui tiendront chaque rôle. Par exemple, quel est le nombre initial d'éclaireurs qu'il faut envisager pour la mission d'exploration ? Teddy serait-il un meilleur éclaireur que Gordie ? Lorsque le rôle des agents change durant le processus, on parle de *ré-allocation de rôles*. Gordie ayant des difficultés à nettoyer la voie de tous les pièges, peut-être qu'il faut affecter le rôle d'éclaireur à Chris afin qu'il aille aider Gordie à éclairer le chemin.

Dans la suite de cette section, on présente deux modèles étudiés dans le domaine de la recherche multi-agents pour faire de l'allocation de rôles : d'abord le problème d'affectation optimale, ensuite un problème de décision multi-agents basé sur les rôles, plus général et qui peut intégrer des concepts tels que l'observabilité partielle, le contrôle décentralisé ou les récompenses jointes. Nous décrivons les principaux avantages et inconvénients de ces deux modèles lorsqu'il s'agit de faire de l'allocation et ré-allocation de rôles.

4.2.1 Problème d'affectation

Le *problème d'affectation optimale* (*Optimal Assignment Problem*, OAP) [Gale, 1989] est un modèle utilisé dans le domaine de la recherche opérationnelle. Ce problème consiste à affecter des travailleurs à des tâches, sachant que chaque tâche n'a pas la même priorité et que chaque travailleur ne sait pas résoudre chaque tâche avec la même efficacité. Il s'énonce comme suit.

Définition 4.2.1 - *Problème d'affectation optimale* : Un *problème d'affectation optimale* est un tuple $(\mathcal{N},\mathcal{M},w,u)$ tel que

- $\mathcal{N}$ est un ensemble fini de n travailleurs,
- $\mathcal{M}$ est un ensemble fini de m tâches à réaliser par les travailleurs,
- $w : \mathcal{M} \to \mathbb{R}^*$ est une fonction de poids telle que $w_j \equiv w(j)$ est la relative importance de la tâche $j \in \mathcal{M}$,
- $u : \mathcal{N} \times \mathcal{M} \to \mathbb{R}^*$ est une fonction d'utilité telle que $u_{ij} \equiv u(i,j)$ représente l'utilité du travailleur $i \in \mathcal{N}$ à être affecté à la tâche $j \in \mathcal{M}$.

Résoudre un OAP consiste à trouver une fonction appelée *affectation*

$$a : \mathcal{N} \to \mathcal{M}$$

telle que a soit bijective :

$$\forall j \in \mathcal{M}\ \exists! i \in \mathcal{N}\ a(i) = j,$$

et maximise l'utilité espérée, qui est la somme des utilités u des travailleurs affectés à leurs tâches, pondérées par l'importance w desdites tâches :

$$U = \sum_{i \in \mathcal{N}} u_{ia(i)} w_{a(i)}. \tag{4.1}$$

En pratique, il est généralement admis que le nombre de travailleurs est égal au nombre de tâches à réaliser : $|\mathcal{N}| = n = m = |\mathcal{M}|$. Nous allons donc simplement noter n le nombre de travailleurs et de tâches. De plus, on pose un certain nombre d'hypothèses, relatives au caractère bijectif de la fonction a :

1. une tâche ne peut être exécutée que par un seul travailleur à la fois,
2. chaque tâche doit être assignée à un travailleur,
3. un travailleur ne peut exécuter qu'une seule tâche à la fois.

Le modèle OAP et ces hypothèses se révèlent être pratiques et simples pour faire de l'allocation de rôles dans les systèmes multi-agents [Gerkey and Matarić, 2003b]. Les auteurs de [Gerkey and Matarić, 2003b] résolvent le problème d'allocation de rôles en ramenant ce problème à un programme linéaire visant à résoudre l'OAP. Formellement, il s'agit de trouver n^2 entiers naturels α_{ij} qui sont solution du programme linéaire :

$$\text{maximiser} \quad \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} \alpha_{ij} u_{ij} w_j$$

tels que

$$\begin{cases} \alpha_{ij} \in \{0,1\},\ \forall i \in \mathcal{N},\ \forall j \in \mathcal{M}, \\ \sum\limits_{i \in \mathcal{N}} \alpha_{ij} = 1,\ \forall j \in \mathcal{M}, \\ \sum\limits_{j \in \mathcal{M}} \alpha_{ij} = 1,\ \forall i \in \mathcal{N}. \end{cases} \tag{4.2}$$

L'équation 4.2 assure que chaque tâche sera affectée à un travailleur et qu'à chaque travailleur sera assignée une tâche – autrement dit, dans le cas de l'allocation de rôles, que chaque rôle sera affecté à un agent, et que tous les agents auront un unique rôle. Comme les α_{ij} sont des entiers naturels, cela signifie qu'ils valent soit 0 soit 1. La fonction d'affectation a est construite à partir des α_{ij} de la façon suivante :

$$a(i) = j \iff \alpha_{ij} = 1. \tag{4.3}$$

Il est ainsi possible de résoudre un OAP en résolvant le programme linéaire ci-dessus, mais un OAP peut également être résolu via d'autres méthodes telles que l'algorithme Hongrois [Kuhn, 1955] (résolution optimale) ou bien via des méthodes gloutonnes, par exemple en assignant séquentiellement le rôle le plus critique à l'agent le plus capable, le second rôle le plus critique au second agent le plus capable, etc... (résolution localement optimale). Qu'elles soient centralisées ou distribuées, toutes les méthodes existantes pour résoudre un OAP ont au pire des complexités polynomiales (complexité en $O(n^3)$ dans le cas de l'algorithme Hongrois lorsque $n = m$).

Une fois la fonction d'affectation a obtenue, les agents sont ainsi assignés à leur rôle et ne s'en départent pas. La version itérative du modèle OAP permet de représenter les problèmes d'affectation répétés, où des changements dans l'environnement peuvent modifier l'utilité d'un agent à exercer son rôle, et donc l'enjoindre à être affecté à un rôle autre que celui qui lui était initialement assigné. À chaque unité de temps (ou toutes les x unités de temps, ou lorsque les utilités u_{ij} d'un ou plusieurs agents changent), les agents re-calculent l'allocation de rôles via l'un des algorithmes évoqués. En pratique, il a été montré empiriquement que la complexité algorithmique de la méthode Hongroise est suffisamment faible pour que l'algorithme puisse être facilement appliqué en temps réel sur un système multi-robots avec $n \leqslant 11$ robots [Gerkey and Matarić, 2003a].

Ramener un problème d'allocation de rôles à un OAP présente donc l'avantage d'être facile à résoudre, même sur des systèmes multi-robots à (relativement) large échelle. Toutefois, le cadre applicatif d'un OAP reste limité. Tout d'abord, il est supposé que l'utilité globale est linéairement calculée selon l'utilité des agents à endosser leur rôle. Peu importe la façon dont l'utilité u_{ij} d'un agent i pour un rôle j est définie, il faut que l'utilité globale U soit une combinaison linéaire des utilités des agents, or cela ne peut pas être le cas dans certaines situations. De plus, l'utilité des agents à assumer leur rôle est une donnée difficile à obtenir dans bon nombre de domaines d'applications. Un basketteur peut jouer à de nombreux postes, mais peut-être que ses capacités physiques et son style de jeu sont mieux adaptés au rôle de *pivot* que de *meneur*. Mais comment calculer son utilité réelle à ce poste-là ? Peut-être qu'il peut aussi bien jouer le rôle d'*ailier fort*, un poste similaire au pivot mais néanmoins différent [2]. Une troisième limite à l'approche OAP pour l'allocation de rôle est le fait que chacune des utilités individuelles doit être connue de tous les agents. Un agent doit en effet connaître le rôle des autres agents, et si un agent change de rôle au milieu de l'exécution, les autres agents doivent en être informés. L'utilité d'un agent pour un rôle peut varier si finalement l'un des agents échoue à son rôle, ou est victime d'une panne. Les utilités doivent donc être communiquées entre les agents pour que ceux-ci (dans le cas de la version itérative du modèle OAP) puissent re-calculer l'allocation adéquate. Néanmoins, cette hypothèse de communication n'est pas toujours adaptée, et dans le cas totalement décentralisé qui nous intéresse, cette hypothèse n'est même pas envisageable.

Finalement, terminons en disant que ce modèle ne permet pas non plus de modéliser les situations où un rôle peut être endossé par plusieurs agents, ni celles où un agent peut endosser plusieurs rôles. Des variantes existent, telles que le GAP (*Generalized Allocation Problem*, où les agents peuvent avoir plusieurs rôles en même temps) ou l'E-GAP (*Extended GAP*, où les agents peuvent en plus changer de rôle en cours d'exécution) [Scerri et al., 2005] mais là encore, de tels modèles ne permettent pas de prendre en compte l'incertitude inhérente (observabilité partielle, actions aux effets stochastiques) des problèmes multi-agents que l'on étudie. Un autre modèle, plus général, est donc requis pour parvenir à représenter des situations réalistes.

4.2.2 Modèle de décision pour l'allocation de rôles

Les travaux de [Nair et al., 2002] (voir également les autres versions [Nair et al., 2003a] et [Nair et al., 2003b]) proposent une analyse pour l'allocation et la ré-allocation de rôles pour le travail d'équipe. Ils décrivent pour cela un modèle, le *Role-based Multiagent Team Decision Problem* (RMTDP, et sa version avec communications, R-COM-MTDP).

2. Dans de tels modèles utilitaristes, les fonctions d'utilité peuvent être fournies par un expert du domaine ou apprises à partir des données.

Définition 4.2.2 - *Role-based Multiagent Team Decision Problem* : Un RMTDP est un tuple $\langle \mathcal{N},\mathcal{S},\mathcal{A},\mathcal{O},T,O,R,\mathcal{L} \rangle$, où

— $\mathcal{N} = \{1,\ldots,n\}$ est l'ensemble fini des agents ;

— $\mathcal{S}$ est l'ensemble des états ;

— $\mathcal{A}$ est l'ensemble des actions jointes ;

— $\mathcal{O}$ est l'ensemble des observations jointes ;

— $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ est la fonction de transition ;

— $O : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow [0,1]$ est la fonction d'observation ;

— $R : \mathcal{S} \times \mathcal{A}$ est la fonction de récompense ;

— $\mathcal{L} = \{l_1,\ldots,l_{|\mathcal{L}|}\}$ est l'ensemble des rôles que peuvent prendre les agents.

Chaque rôle $l \in \mathcal{L}$ peut être endossé par un ou plusieurs agents. L'ensemble $\mathcal{A}_i$ des actions d'un agent i se distingue en deux sous-ensembles :

$$\mathcal{A}_i = \Upsilon_i \cup \Phi_i, \tag{4.4}$$

où

— Υ_i est l'ensemble des *actions de prise de rôle*. Une action $v_{i,l} \in \Upsilon_i$ désigne le fait que l'agent i *choisit* le rôle l.

— $\Phi_i = \cup_{l \in \mathcal{L}} \Phi_{i,l}$ est l'ensemble des *actions d'exécution de rôle*, et $\Phi_{i,l}$ est l'ensemble des actions du rôle l pour l'agent i.

Dans un RMTDP, les agents alternent entre sélection de rôles et exécution d'actions liées à ces rôles. À une étape de décision t divisible par 2, un agent i choisit un rôle $l \in \mathcal{L}$ en prenant l'action $v_{i,l}^t \in \Upsilon_i$. À $t+1$, il exécute une action $\phi_{i,l}^{t+1} \in \Phi_{i,l}$ qui correspond au rôle choisi. La sélection d'un rôle limite le choix des actions d'exécution de rôle disponibles pour un agent i mais cette étape est nécessaire si, pour deux rôles $l_1,l_2 \in \mathcal{L}$, les ensembles d'actions d'exécution de rôle Φ_{i,l_1} et Φ_{i,l_2} sont disjoints.

Cette modélisation permet de représenter les coûts liés à un changement de rôle et les récompenses attachées aux actions d'exécution de rôle (qui sont les "vraies" actions à proprement parler). La fonction de récompense se distingue ainsi elle-même selon l'instant de décision. Aux instants pairs, $R_\Upsilon(s,v_{1,l}^t,\ldots,v_{n,l}^t)$ est le coût, strictement négatif, lié à la prise des rôles par chaque agent, et aux instants impairs, $R_\Phi(s,\phi_{1,l}^t,\ldots,\phi_{n,l}^t)$ est la récompense liée aux effets des actions sur le système. Les coûts liés aux changements de rôles peuvent par exemple représenter l'investissement nécessaire à l'entraînement d'un agent, ou bien à un temps de trajet et une consommation de carburant, ou encore à un changement d'équipement. Ainsi, même si changer les rôles au sein de l'équipe d'agent peut engendrer un coût immédiat non négligeable, celui-ci pourra être amorti sur le long terme par les récompenses des actions qui seront effectuées.

Le modèle est assez riche pour représenter de nombreuses situations. En particulier, il est possible de comparer deux agents $i,j \in \mathcal{N}$ qui endossent le même rôle $l \in \mathcal{L}$ mais avec des ensembles d'actions d'exécution de rôle $\Phi_{i,l}$ et $\Phi_{j,l}$ différents. Il est de même possible de comparer l'efficacité d'un agent $i \in \mathcal{N}$ dans deux rôles $l,k \in \mathcal{L}$ différents en analysant les récompenses générées avec les ensembles d'actions $\Phi_{i,l}$ et $\Phi_{i,k}$. Enfin, il est possible d'analyser l'inactivité d'un agent i lorsqu'il choisit un rôle $l \in \mathcal{L}$ tel que son ensemble d'actions $\Phi_{i,l} = \varnothing$ est vide.

Résoudre un RMTDP consiste à trouver des politiques jointes π_Υ (pour les actions de prise de rôle) et π_Φ (pour les actions d'exécution de rôle) qui maximisent la somme des récompenses espérées sur un certain horizon de planification h. Les auteurs ont montré dans [Nair et al., 2003a]

que trouver des politiques jointes optimales pour un RMTDP était un problème NEXP-complet, la même classe de complexité que les DEC-POMDPs. Lorsqu'on autorise les communications, comme dans un R-COM-MTDP, ils ont montré dans [Nair et al., 2003b] que le problème était PSPACE-complet, la même classe de complexité que les POMDPs. Ces résultats suggèrent qu'il est possible de représenter ces deux modèles directement avec des (DEC-)POMDPs tels que présentés dans le chapitre 3.

Comparé à l'utilisation d'un OAP (ou d'un E-GAP) pour faire de l'allocation de rôles, un RMTDP est un modèle plus général qui permet de représenter une plus grande classe de problèmes. En premier lieu, le modèle tient compte de l'observabilité partielle des agents sur leur environnement, et permet de faire de la résolution de tâche en plus de l'allocation de rôle, là où un OAP ne se contente que d'attribuer un rôle aux agents. Ensuite, contrairement à un OAP qui se base sur une fonction d'*utilité*, un RMTPD se base sur une fonction de *récompense* pour évaluer la pertinence d'un agent dans un rôle. La différence conceptuelle est importante, car la fonction d'utilité, généralement difficile à obtenir lors de la modélisation d'un problème, est censée être une mesure globale et parfaite de l'information dont on dispose sur la performance d'un agent dans un rôle. C'est comme faire jouer un joueur de basket-ball mille fois à deux postes différents et établir une moyenne des matchs gagnés à chacun des postes. On obtient ainsi l'utilité de ce joueur à ces deux postes. La fonction d'utilité est donc comparable à une fonction de valeur, semblable à ce que l'on retrouve dans les (PO)MDPs. La fonction de récompense, quant à elle, est une mesure immédiate de l'action d'un agent ayant un certain rôle dans un certain état, mais ne donne pas d'information sur les performances à long terme de cet agent dans ce rôle. Il s'agit en général d'une donnée plus simple à renseigner lorsque l'on modélise un problème. Au basket-ball par exemple, on peut donner une large récompense positive à un arrière qui marque un panier (2 ou 3 points, selon la distance au panier), ou une petite récompense positive à un ailier qui réussit un rebond défensif, etc. Malheureusement, l'utilisation d'une fonction de récompense, l'ajout de l'observabilité partielle et le contrôle séquentiel et décentralisé font tomber les RMTDPs dans le domaine des problèmes NEXP-complets.

À l'époque de la publication de ces travaux, au début des années 2000, il n'existait que peu, voire pas, de résultats algorithmiques pour résoudre efficacement un DEC-POMDP. L'étude des DEC-POMDPs n'en était elle-même qu'à ses balbutiements. Les méthodes de re-formation d'équipe étudiées pour les RMTDPs se concentraient donc principalement sur l'étude et la comparaison de politiques d'allocation et de ré-allocation de rôles, ce dont nous traitons dans la section suivante.

4.2.3 Stratégies de ré-allocation de rôles

Lorsque l'environnement change, qu'un évènement imprévu se présente, qu'un agent tombe hors-service ou se retrouve incapable de continuer à tenir son rôle, il est nécessaire que les autres membres de l'équipe se réorganisent en réaction à un tel évènement.

STEAM. La stratégie STEAM (*Shell for TEAMwork*) [Tambe, 1997] est une stratégie de changement de rôle suite à la défection d'un agent. Un agent a qui suit la stratégie STEAM et ayant un rôle $l_a \in \mathcal{L}$ change de rôle pour un rôle $l_b \in \mathcal{L}$ si un agent b de rôle l_b venait à être défectueux et si

$$C(l_b) - C(l_a) > 0, \tag{4.5}$$

où $C : \mathcal{L} \rightarrow \{0,1\}$ mesure l'importance des rôles, tel que $C(l) = 1$ si $l \in \mathcal{L}$ est critique ; 0 sinon. Autrement dit, un agent change de rôle pour un rôle tenu par un agent défectueux si son propre rôle est moins critique à la résolution de la tâche que celui de l'agent défectueux.

Cette définition de rôle *critique* est faite avant le début de l'exécution de la tâche. Reprenons l'exemple de nos explorateurs. Le seul rôle critique est celui de *transporteur*, tenu par Vern. En effet, sans *transporteur*, l'équipe ne va trivialement pas accumuler de récompenses. De même, il s'agit du seul rôle nécessaire : les autres agents ne sont là que pour protéger Vern, mais dans le meilleur des mondes possibles (celui où il ne rencontre aucun piège), Vern pourrait se suffire à lui-même pour accomplir la tâche. Si toutefois Vern tombe dans un piège et qu'il ne peut s'en sortir, l'un des autres aventuriers va devoir endosser le rôle de *transporteur*. On peut imaginer que c'est Teddy, le *scientifique*, qui va devoir abandonner sa tâche pour se concentrer sur l'objectif principal, et ainsi devenir le nouveau *transporteur* de l'équipe.

DPRE. La stratégie DPRE (*Dynamic Positioning and Role Exchange*) [Reis et al., 2000], permet à deux agents d'échanger leurs rôles s'ils se mettent d'accord sur l'utilité que ce changement pourrait apporter à l'équipe. Comme son nom l'indique, cette méthode, appliquée au domaine de la *RoboCup Soccer* (une compétition de robots joueurs de football) [Kitano et al., 1997] permet aussi de gérer le positionnement (géographique) des agents, mais nous ignorons cette composante pour notre cas. Formellement, deux agents $a,b \in \mathcal{N}$ ayant des rôles respectifs $l_a,l_b \in \mathcal{L}$ échangent leurs rôles si

$$U(l_a,b) + U(l_b,a) > U(l_a,a) + U(l_b,b), \tag{4.6}$$

où $U(l_i,j)$ est une fonction d'utilité (ou fonction de valeur) qui estime la valeur d'un agent $j \in \mathcal{N}$ prenant le rôle $l_i \in \mathcal{L}$ d'un agent $i \in \mathcal{N}$. En d'autres termes, deux agents échangent leurs rôles si cet échange est favorable.

Par exemple, Gordie, l'*éclaireur*, peut décider d'échanger son rôle avec Chris, le *protecteur*, si Chris est mieux armé pour désamorcer les pièges en avant de son équipe et que Gordie peut décemment protéger Teddy et Vern, pas forcément aussi bien que Chris le ferait, mais suffisamment pour que l'échange de rôles ne soit pas contre-productif.

La stratégie DPRE peut également être appliquée au cas particulier de ré-allocation de rôles suite à la défection d'un agent : un agent a de rôle l_a remplace un agent b défectueux de rôle l_b si

$$U(l_b,a) > U(l_a,a). \tag{4.7}$$

Les deux stratégies STEAM et DPRE permettent à une équipe d'être flexible durant la résolution d'une tâche, que ce soit dans le cas de la défection d'un agent, ou dans le cas où un agent sous-performe à son rôle. L'allocation et la ré-allocation de rôles sert une partie du processus de formation et de re-formation d'équipe.

Maintenant que l'on a examiné l'allocation et la ré-allocation de rôles, nous nous attardons dans la section suivante sur le second axe relatif aux équipes flexibles : la formation de coalitions et la recherche de structures de coalitions optimales.

4.3 Formation de coalitions

Le sujet de la formation de coalitions trouve son origine dans la théorie des jeux coopératifs, une branche de la théorie des jeux. De façon générale, la formation de coalitions est un moyen incitant des agents (plus communément appelés *joueurs* dans le cadre de la théorie des jeux) à se regrouper temporairement afin de résoudre une tâche commune, ou d'atteindre un but commun. On ne considère pas que les agents sont égoïstes et cherchent à se regrouper pour maximiser leurs gains personnels. On considère plutôt les situations où les agents doivent se séparer en

sous-groupes (les coalitions) pour parvenir à résoudre plus efficacement une tâche globale ou plusieurs sous-tâches individuelles.

Dans un **jeu de coalitions** [Morgenstern and Von Neumann, 1953], chaque coalition possède une valeur. Cette valeur décrit l'utilité de cette coalition d'agents. Le but, pour l'ensemble des agents, est de se grouper en coalitions pour faire en sorte de maximiser la somme des valeurs de chacune d'entre elles. Dans cette section, notre objectif est de décrire le cadre général de la formation de coalitions et de faire le lien qui existe entre ce domaine et notre problématique de planification dans les systèmes multi-agents ouverts.

4.3.1 Définitions

Soit $\mathcal{N} = \{1,2,\ldots,n\}$ un ensemble de n agents coopératifs.

Définition 4.3.1 - *Coalitions* : Une *coalition* $\mathcal{C} \subseteq \mathcal{N}$ est un sous-ensemble d'agents.
On note $2^{\mathcal{N}}$ l'ensemble des coalitions. Il s'agit des parties non vides de $\mathcal{N}$. Il y en a $2^n - 1$.
On appelle *grande coalition* la coalition $\mathcal{N}$ qui contient tous les agents.
On appelle *coalition singleton* une coalition composée d'un unique agent.

Définition 4.3.2 - *Structures de coalitions* : Une *structure de coalitions* $\mathcal{SC}$ est une partition de $\mathcal{N}$, c'est-à-dire un ensemble de coalitions $\mathcal{SC} = \{\mathcal{C}_1,\mathcal{C}_2,\ldots,\mathcal{C}_m\}$ respectant les propriétés suivantes :

1. Toutes les coalitions sont *deux à deux disjointes* : $\forall \mathcal{C}_i,\mathcal{C}_j \in \mathcal{SC}$, $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing$,
2. Toute coalition est *non vide* : $\forall \mathcal{C}_i \in \mathcal{SC},\mathcal{C}_i \neq \varnothing$,
3. L'union des coalitions *recouvre* $\mathcal{N}$: $\bigcup_{i=1}^{m} \mathcal{C}_i = \mathcal{N}$.

Illustrons ces deux définitions. Considérons une expédition des ruines anciennes effectuée par trois de nos explorateurs : Gordie (G, l'éclaireur), Chris (C, le protecteur) et Vern (V, le transporteur). La grande coalition est l'ensemble des trois agents :

$$\mathcal{N} = \{G,C,V\}.$$

Il y a en tout $2^{|\mathcal{N}|} - 1 = 7$ coalitions non vides :

$$\{\{G\},\{C\},\{V\},\{G,C\},\{G,V\},\{C,V\},\{G,C,V\}\},$$

et 5 structures de coalitions différentes :

— $\mathcal{SC}_1 = \{\{G\},\{C\},\{V\}\}$,
— $\mathcal{SC}_2 = \{\{G,C\},\{V\}\}$,
— $\mathcal{SC}_3 = \{\{G\},\{C,V\}\}$,
— $\mathcal{SC}_4 = \{\{G,V\},\{C\}\}$,
— $\mathcal{SC}_5 = \{\{G,C,V\}\}$.

Par exemple, la structure de coalitions $\mathcal{SC}_3$ signifie que Gordie est seul de son côté pendant que Chris et Vern restent ensemble. En se rappelant que Gordie est l'éclaireur du groupe et que Vern est le transporteur de la cargaison et qu'il a besoin de la protection de Chris, alors cette structure de coalitions fait sens. Comparée à la structure de coalitions $\mathcal{SC}_1$ où chaque aventurier se retrouve séparé des autres et explore les ruines de son côté, $\mathcal{SC}_3$ est intuitivement meilleure.

La **fonction caractéristique** est le moyen utilisé afin de comparer les utilités de deux structures de coalitions différentes.

Définition 4.3.3 - *Fonction caractéristique* : Une *fonction caractéristique*, ou *fonction de valuation*,

$$v : 2^{\mathcal{N}} \rightarrow \mathbb{R}$$

associe à chaque coalition $\mathcal{C} \in 2^{\mathcal{N}}$ une valeur $v(\mathcal{C})$, telle que $v(\varnothing) = 0$. La valeur d'une structure de coalitions $\mathcal{SC}$ est la somme des valeurs des coalitions de $\mathcal{SC}$:

$$v(\mathcal{SC}) = \sum_{\mathcal{C} \in \mathcal{SC}} v(\mathcal{C}). \tag{4.8}$$

Le tableau 4.1 de gauche donne un exemple de fonction caractéristique appliquée au domaine de notre illustration fil rouge.

$\mathcal{C}$	$v(\mathcal{C})$
$\{G\}$	1
$\{C\}$	0
$\{V\}$	-1
$\{G,C\}$	2
$\{G,V\}$	1
$\{C,V\}$	3
$\{G,C,V\}$	2

$\mathcal{SC}$	$v(\mathcal{SC})$
$\{\{G\},\{C\},\{V\}\}$	0
$\{\{G,C\},\{V\}\}$	1
$\{\{G\},\{C,V\}\}$	4
$\{\{G,V\},\{C\}\}$	1
$\{\{G,C,V\}\}$	2

Tableau 4.1 – Exemple de fonction caractéristique pour un jeu de coalitions. Le tableau de gauche donne les utilités marginales des différentes coalitions. Par exemple, Vern ne doit pas être laissé seul, mais c'est mieux s'il est avec Chris que s'il est avec Gordie. Le tableau de droite donne les utilités des structures de coalitions, calculées à partir de l'équation 4.8.

On dispose à présent de tous les ingrédients afin d'introduire le modèle formel de jeu de coalitions.

Définition 4.3.4 - *Jeu de coalitions* : Un *jeu de coalitions* est une paire ordonnée $(\mathcal{N},v)$, où

— $\mathcal{N} = \{1,2,\ldots,n\}$ est l'ensemble fini des $n \in \mathbb{N}$ joueurs ;

— $v : 2^{\mathcal{N}} \rightarrow \mathbb{R}$ est la fonction caractéristique, telle que $v(\varnothing) = 0$.

Résoudre un jeu de coalitions $(\mathcal{N},v)$ consiste à trouver une structure de coalitions $\mathcal{SC}^*$ optimale, c'est-à-dire une structure de coalitions telle que

$$\mathcal{SC}^* = \underset{\mathcal{SC}}{\operatorname{argmax}}\, v(\mathcal{SC}). \tag{4.9}$$

En quoi la recherche d'une structure de coalitions optimale aide-t-elle au problème de planification dans les systèmes multi-agents ouverts ? Nous en reparlerons lorsque nous aborderons les contributions de notre travail, mais l'idée est que lorsque l'on considère un système multi-agents ouvert, on divise en deux sous-groupes l'ensemble des agents : d'une part les agents *à l'intérieur* du système, ceux qui travaillent activement à la résolution de la tâche, et d'autre part les agents *à l'extérieur* du système, ceux qui sont en attente ou hors-service. Cela revient à considérer des structures de coalitions de taille deux – c'est-à-dire, composées uniquement de deux coalitions.

Comme nous allons le voir maintenant, certaines propriétés des jeux de coalitions font qu'il est parfois simple de trouver la structure de coalitions optimale. Il s'agit en général d'un problème compliqué, puisque le nombre de coalitions augmente exponentiellement avec le nombre

n d'agents, et le nombre de structures de coalitions plus rapidement encore [Sandholm et al., 1999].

4.3.2 Propriétés

On examine dans cette section certaines propriétés observées par certaines classes de jeux de coalitions. Ces définitions vont nous être utiles par la suite pour identifier des classes de problèmes de planification multi-agents ouverts qui présentent le même genre de caractéristiques.

Définition 4.3.5 - *Monotonie* : Un jeu de coalitions $(\mathcal{N},v)$ est *monotone* si, pour toutes coalitions $\mathcal{C}_1 \subseteq \mathcal{C}_2 \subseteq \mathcal{N}$, on a :

$$v(\mathcal{C}_1) \leqslant v(\mathcal{C}_2). \tag{4.10}$$

La propriété de monotonie décrit le cas où les plus grosses coalitions ont une plus grande utilité. Dans un jeu de rôle en ligne massivement multi-joueurs, plusieurs dizaines, voire centaines, de joueurs se regroupent parfois pour arriver à vaincre certains ennemis particulièrement redoutables. Les récompenses obtenues par les joueurs lorsqu'ils éliminent l'ennemi ne sont pas partagées entre eux. Chaque joueur reçoit (généralement) le même gain, qui ne dépend que du niveau de l'ennemi, de sa puissance et de ses points de vie, et pas du nombre de joueurs qui se sont mobilisés pour résoudre cette quête. Ainsi, il ne peut être que bénéfique aux joueurs que d'autres joueurs se joignent à eux pour battre l'ennemi plus facilement.

Toutes les situations ne sont bien évidemment pas monotones. Un exemple simple est celui du tableau 4.1 : la grande coalition $\{G,C,V\}$ qui contient tous les explorateurs est moins efficace que la coalition $\{C,V\}$, car Gordie se trouve ralenti et inefficace dans son rôle d'éclaireur s'il doit rester auprès de ses compagnons.

Définition 4.3.6 - *Sur-additivité* : Un jeu de coalitions $(\mathcal{N},v)$ est *sur-additif* si, pour toutes coalitions disjointes $\mathcal{C}_1,\mathcal{C}_2 \subseteq \mathcal{N}$ telles que $\mathcal{C}_1 \cap \mathcal{C}_2 = \varnothing$, on a :

$$v(\mathcal{C}_1) + v(\mathcal{C}_2) \leqslant v(\mathcal{C}_1 \cup \mathcal{C}_2). \tag{4.11}$$

Cette définition spécifie que deux coalitions disjointes peuvent être meilleures en se regroupant en une seule coalition. On peut dire d'une certaine manière que le tout est supérieur à la somme des parties.

Une illustration immédiate montrant l'omniprésence de la notion de sur-additivité dans beaucoup de situations réelles est encore une fois l'exemple du feu de forêt. On pourrait envoyer une équipe de pompiers pour empêcher le feu de ravager la zone habitée ; ou bien, on pourrait envoyer une brigade de policiers ériger un périmètre de sécurité et procéder à l'évacuation des civils. Ces deux équipes, envoyées séparément (l'une après l'autre), pourraient efficacement réussir la mission qui leur incombe, mais elles auraient tout intérêt à coopérer et agir en parallèle pour obtenir de meilleurs résultats.

Comme en ce qui concerne la monotonie, l'exemple du tableau 4.1 n'est pas sur-additif : la coalition $\{G,C,V\}$ a une utilité de 2, alors que, prises séparément, les coalitions $\{G\}$ et $\{C,V\}$ ont une utilité cumulée de 4.

Notons que tout jeu de coalitions sur-additif est également monotone (si $v(\mathcal{C}) > 0,\forall\mathcal{C}$). De la définition de sur-additivité ci-dessus peut dériver le concept plus fort encore de convexité.

Définition 4.3.7 - *Convexité* : Un jeu de coalitions $(\mathcal{N},v)$ est *convexe* si, pour tout agent $i \in \mathcal{N}$ et pour toutes coalitions $\mathcal{C}_1 \subset \mathcal{C}_2 \subset \mathcal{N}\backslash\{i\}$ ne contenant pas i, on a :

$$v(\mathcal{C}_1 \cup \{i\}) - v(\mathcal{C}_1) \leqslant v(\mathcal{C}_2 \cup \{i\}) - v(\mathcal{C}_2). \tag{4.12}$$

La propriété de convexité d'un jeu de coalitions spécifie que la volonté d'un agent quelconque à rejoindre une coalition augmente proportionnellement avec la taille de cette coalition – Lloyd Shapley, fameux mathématicien et économiste américain spécialisé dans le domaine de la théorie des jeux, récipiendaire du prix Nobel de science économique en 2012, parle d'*effet boule de neige* [Shapley, 1971].

Nous n'allons pas plus entrer dans les détails de la formation de coalitions dans cette thèse. En particulier, on ne traitera pas des algorithmes de recherche de structures de coalitions optimales, ni du calcul des valeurs des coalitions ou bien de la formation de coalitions d'agents égoïstes. Tout cela est hors de la portée de notre travail. Les définitions et propriétés structurelles décrites dans cette partie sont néanmoins des observations qu'il sera bon de garder en mémoire, puisque nous y reviendrons dans la partie relative à nos contributions. Le lecteur peut se référer à [Ray and Vohra, 2015] pour un état de l'art sur la formation de coalitions, à [Wooldridge, 2011] pour un point du vue concernant l'aspect théorique et algorithmique de la théorie des jeux coopératifs, et enfin à [Curiel, 2013] pour plus d'informations concernant les apports de la théorie des jeux coopératifs aux problèmes d'optimisation combinatoires.

Nous abordons à présent le troisième et dernier axe concernant les équipes flexibles. Dans la section suivante, nous traitons du travail d'équipe *ad-hoc* et des agents *non dévoués*.

4.4 Contrôle partiel de systèmes multi-agents ouverts

Cette section présente les autres aspects du contrôle de systèmes multi-agents ouverts qui n'entrent ni dans le cadre de la formation dynamique d'équipes, ni celui des stratégies de remplacement ou d'échange de rôles, ni celui de la théorie des jeux coopératifs.

Nous l'avons déjà abordé, notamment dans la section 2.1.1 de notre chapitre concernant les définitions générales, les systèmes multi-agents ouverts désignent des environnements où les agents sont développés par différentes parties (entreprises ou pays concurrents), servent des intérêts différents et peuvent librement quitter ou rejoindre le système à n'importe quel moment [Shehory, 2001,Calmet et al., 2004,Huynh et al., 2006]. En particulier, il est généralement supposé que les agents ayant des objectifs distincts ne sont pas coopératifs les uns avec les autres, voire qu'ils sont en directe confrontation. De plus, tous les agents ne sont pas parfaitement conscients de leur environnement. Ils peuvent ne pas savoir quels sont les autres agents présents avec eux dans le système et ils n'ont qu'une vision locale de tout leur environnement. Les exemples de systèmes multi-agents ouverts sont nombreux :

— *Nuées d'oiseaux.* Certaines espèces d'oiseaux, telles que les passeriformes (moineaux, étourneau, etc.), se regroupent par instinct grégaire en immenses nuées dans un but coopératif, par exemple pour effectuer une migration.

— *Places de marché en ligne.* Les acheteurs et vendeurs se connectent sur une plate-forme de commerce électronique pour acheter et vendre des biens et services. La collaboration comme la compétition sont de mise, comme souvent lorsqu'il s'agit d'agents humains.

— *Blockchain.* Les mineurs doivent s'assurer d'être assez nombreux pour résoudre les problèmes cryptographiques permettant d'assurer la sécurité de la blockchain, mais aucun mineur ne veut fournir la plus grande puissance de calcul.

— *Jeux de rôle en ligne massivement multi-joueurs.* Certains ennemis particulièrement puissants requièrent que des groupes de joueurs se forment pour parvenir à les vaincre. N'importe quel joueur peut rejoindre ou quitter le groupe à n'importe quel moment, mais il faut que les joueurs soient assez nombreux et assez forts s'ils veulent obtenir le butin de récompense.

— etc.

Dans tous ces exemples, le nombre d'agents est susceptible de varier à tout moment. En outre, et c'est le point qui nous importe le plus, à cause de la différence de provenance des agents, il n'est généralement pas possible d'exercer un contrôle sur eux. En fait, le seul cas de contrôle dans ce genre de systèmes se manifeste lorsque l'on est nous-même un utilisateur du système. Mais du point de vue du concepteur du système multi-agents, le but n'est pas tant de contrôler les agents que, soit de leur offrir une plate-forme réglementée et sûre pour leur permettre d'interagir en toute confiance (ou au moins, en connaissance de cause) et avec efficacité [Huynh et al., 2006], soit d'étudier les propriétés théoriques du-dit système [Hendrickx and Martin, 2017].

Les systèmes multi-agents non-contrôlés sont hors de la portée de cette thèse. Toutefois, nous allons examiner deux cas particuliers de systèmes multi-agents où une part de contrôle est consacrée à un (ou plusieurs) agent(s). On identifie deux cas :

1. un agent, *sur lequel on exerce un contrôle*, **entre** dans un système d'agents coopératifs sur lesquels on n'a aucun contrôle ;
2. un agent, sur lequel on n'a aucun contrôle, **quitte** un système d'agents coopératifs, *sur lesquels on exerce un contrôle.*

Cette partie va être l'occasion d'aborder brièvement quelques travaux de recherche récents qui s'attaquent aux deux cas ci-dessus. Le premier cas correspond au travail d'équipe ad-hoc.

4.4.1 Travail d'équipe ad-hoc

Le **travail d'équipe ad-hoc** [Stone et al., 2010, Genter et al., 2011, Barrett and Stone, 2012] se réfère au cas où des agents doivent coopérer dans l'optique de résoudre une tâche, mais sans qu'ils aient de coordination préalable sur comment travailler ensemble. Ces situations peuvent apparaître lorsque des agents, qui n'ont pas été conçus ensemble et/ou qui n'étaient pas programmés pour travailler de concert sur une même tâche, doivent se mettre à coopérer car leurs objectifs sont parfaitement alignés. Par exemple, suite à une catastrophe naturelle de grande ampleur, plusieurs pays peuvent envoyer des robots de sauvetage sur place afin d'aider les populations. Ces robots ont probablement des origines différentes et peut-être également des capacités différentes, mais leur objectif commun est le même. Il leur faut donc se coordonner s'ils veulent parvenir à sauver le maximum de vies sans interférer les uns avec les autres.

L'objectif de la recherche dans le domaine du travail d'équipe ad-hoc consiste à concevoir des agents, dits **agents ad-hoc**, qui soient autonomes et capables de collaborer efficacement avec des coéquipiers préalablement inconnus sur des tâches pour lesquelles ils peuvent tous contribuer individuellement. L'aspect *ouvert* dans le travail d'équipe ad-hoc intervient au niveau de l'agent ad-hoc. Celui-ci rejoint une équipe déjà présente sur place et doit s'adapter à elle. Il s'agit donc du seul agent sur lequel on exerce un contrôle, mais l'ensemble correspond tout de même à un système multi-agents coopératif.

Dans [Genter and Stone, 2014, Genter and Stone, 2016], les auteurs présentent des méthodes pour influencer un essaim d'agents robotiques simples à adopter un comportement souhaité dans le cadre d'un travail d'équipe ad-hoc. En intégrant un ou plusieurs agents ad-hoc à l'intérieur d'un essaim composé de plusieurs centaines d'agents, les auteurs montrent qu'il est possible d'orienter

la trajectoire de l'essaim vers la direction souhaitée, par exemple afin d'éviter des zones à risques telles que les aéroports. Dans ces travaux, l'agent ad-hoc possède une connaissance de l'équipe, une connaissance de l'environnement et travaille avec des agents réactifs à ses actions.

La plupart des recherches existantes sur le travail d'équipe ad-hoc sont axées sur les cas dans lesquels l'agent ad-hoc a accès à un modèle correct de ses coéquipiers. Lorsque le comportement ou la stratégie suivis par les agents dans le système sont inconnus de l'agent ad-hoc, celui-ci doit observer ses coéquipiers et passer par une phase d'apprentissage s'il veut coopérer efficacement avec eux [Barrett et al., 2012, Barrett et al., 2016]. Il s'agit du cas le plus complexe de problème ad-hoc : l'agent ne connaît pas le comportement de ses coéquipiers et ne sait pas à l'avance si ses propres actions auront une quelconque influence sur eux. L'agent ad-hoc peut soit disposer d'une base préalable de modèles potentiels qui peuvent être suivis par les autres agents (il peut s'agir d'une politique ou d'un comportement plus simple), et auquel cas il doit sélectionner le bon modèle et s'en servir pour planifier son propre comportement ; soit il ne dispose d'aucune information préalable, et dans ce cas-là il va devoir commencer par observer, hors-ligne, les actions prises par ses coéquipiers pour se construire une représentation interne de leur vrai modèle.

4.4.2 Agents non dévoués

Les **agents non dévoués** [Cohen and Levesque, 1991, Tambe, 1997] constituent, en un sens, le cas "inverse" des agents ad-hoc. Plutôt que de contrôler un unique agent qui *entre* dans une équipe avec laquelle il va devoir coopérer, on contrôle ici une équipe d'agents coopératifs qui doit réagir lorsqu'un agent *sort* de l'équipe de façon inattendue. Ce genre de situations peut se présenter lorsque les agents ont une chance de quitter l'équipe à tout instant, soit à cause d'une panne technique (par exemple, un drone de surveillance peut être la cible d'une attaque malveillante et tomber hors-service), soit parce qu'il leur faut gérer une tâche prioritaire (un pompier qui éteint un feu peut être rappelé de toute urgence par son chef de service afin de se rendre sur le lieu d'un autre accident).

Dans [Agrawal and Varakantham, 2017, Agrawal et al., 2018], les auteurs présentent un modèle d'allocation de tâches pour des équipes d'agents non dévoués. Chaque agent i possède un paramètre δ_i^t de *non dévotion*, une mesure de la probabilité que l'agent quitte l'équipe à l'instant t. Un agent est modélisé par un MDP et possède donc notamment un ensemble d'actions, des états internes et une fonction de récompense individuelle. Les agents doivent se répartir les différentes tâches, certains agents pouvant éventuellement être affectés à plusieurs tâches, d'une façon assez similaire à ce que nous avons vu dans la section 4.2.2 concernant l'allocation de rôles. Le but est de maximiser la somme des récompenses que chaque agent obtient en travaillant sur les tâches qui lui ont été allouées. Les auteurs décrivent également différentes approches réactives et pro-actives pour faciliter la coordination parmi les agents restants lorsqu'ils doivent se répartir les tâches laissées inaccomplies par les agents non dévoués qui ont quitté l'équipe. D'une part, les approches réactives requièrent que les agents recalculent l'allocation de tâches optimale chaque fois qu'un agent quitte l'équipe. D'autre part, les approches pro-actives permettent quant à elles de calculer hors-ligne une unique allocation de tâches qui prend en compte les probabilités de défection δ_i^t des agents. La combinaison de ces deux approches fournit les meilleurs résultats.

Nous pourrons également citer [Chandrasekaran et al., 2016], où les auteurs focalisent leur attention sur les systèmes multi-agents où des agents peuvent quitter momentanément l'équipe en place avant de revenir, mais où de nouveaux agents ne peuvent pas entrer dans le système. Par exemple, un pompier qui combat un feu de forêt peut se retrouver à court de ressources et va devoir momentanément quitter le système, faire le plein, puis revenir. Le travail de ces

auteurs s'intéresse à la planification individuelle d'un agent qui évolue dans un tel système. Celui-ci doit pouvoir prévoir lorsque l'un de ses voisins quitte le système ou le rejoint après l'avoir préalablement quitté. Il s'agit donc d'un problème à mi-chemin entre le travail d'équipe ad-hoc et celui sur les agents non dévoués.

La recherche dans le domaine des équipes d'agents non dévoués est encore récente et il ne semble pas s'agir d'un domaine en plein essor, mais savoir que cela existe est important pour estimer la position de nos travaux par rapport à l'existant.

4.5 Conclusion

Ce chapitre aura été l'occasion d'étudier les travaux existants qui abordent, d'une façon ou d'une autre, le problème de formation dynamique d'équipes, ainsi que le caractère flexible de ces équipes. Nous avons examiné trois approches distinctes.

L'allocation et la ré-allocation de rôles permettent d'attribuer des rôles à des agents (afin de les spécialiser dans un certain domaine), et de leur permettre de modifier dynamiquement leur rôle (afin de s'adapter au travail d'équipe et aux changements de situations qui peuvent survenir). Dans les différents modèles étudiés, il n'est pas tant question de formation dynamique d'équipe que d'adaptation de l'unique équipe d'agents. Ainsi, bien que le caractère *flexible* de l'équipe soit exploité, il reste que cette méthode est insuffisante pour aborder les cas plus complexes des systèmes ouverts.

Ensuite, la formation de coalitions dans le domaine de la théorie des jeux coopératifs vise à former des sous-groupes d'agents de façon à maximiser le bien-être ou les récompenses que ces agents peuvent recevoir. Certains agents préfèrent être avec d'autres en particulier, ou bien un agent a tout simplement plus à gagner en se ralliant à une coalition d'agents puissante ou influente. Tous les concepts utilisés dans le domaine de la théorie des jeux coopératifs ne sont pas applicables à nos travaux. En effet, on cherche avant tout à déterminer le meilleur groupe d'agents dans chaque situation possible, et non à trouver la meilleure structure de coalitions. Malgré cette différence (certes fondamentalement importante), certains des concepts utilisés dans la théorie des jeux coopératifs vont se révéler être utiles par la suite.

Finalement, le travail d'équipe ad-hoc et l'étude des agents non dévoués constituent deux domaines qui permettent d'exercer un contrôle partiel sur un système multi-agents. En ce sens, ces travaux représentent ce qui se rapproche le plus des recherches présentées dans cette thèse. Toutefois, malgré les efforts existants dans le domaine, il n'existe toujours pas de modèle de décision que permette de faire de la planification dans les systèmes multi-agents décentralisés, partiellement observables et ouverts, où les agents peuvent entrer et sortir à tout moment.

Troisième partie

Modèles

Chapitre 5

POMDPs décentralisés & ouverts

Sommaire

5.1 Introduction 66
5.2 Vers un DEC-POMDP ouvert 67
5.2.1 Le problème TARS 67
5.2.2 Analyse de benchmarks 68
5.3 DEC-POMDPs ouverts 69
5.3.1 Définition du modèle 70
5.3.2 Niveaux de contrôle du processus d'entrées et sorties 71
5.3.3 Coûts des transitions d'équipes 73
5.3.4 Historiques, politiques et résolution 73
5.3.5 Avantages et inconvénients 74
5.4 L'algorithme BRD 74
5.4.1 Équilibre de Nash 74
5.4.2 Best Response Dynamics 75
5.4.3 Sélection des agents 76
5.5 L'algorithme POS-UCT 77
5.5.1 Présentation 78
5.5.2 Recherche arborescente Monte-Carlo dans les domaines partiellement observables 78
5.5.3 Historiques rares 84
5.6 Conclusion 85

« Si on marche assez loin, dit Dorothée, nous arriverons quelque part à un moment donné. »

Lyman Frank Baum
Le Magicien d'Oz

5.1 Introduction

Effectuons un bref récapitulatif de ce que nous avons vu jusqu'à présent. Les deux chapitres précédents se sont attachés à effectuer une revue de l'état de l'art. D'un côté, le chapitre 3 a introduit les processus décisionnels de Markov et, en particulier, les DEC-POMDPs, des modèles décisionnels multi-agents adaptés à la planification sous incertitude de politiques décentralisées. Les principaux problèmes d'un tel modèle sont sa complexité algorithmique et le fait que la configuration de l'équipe d'agents doit rester statique lors de l'exécution. D'un autre côté, le chapitre 4 a présenté quelques modèles et méthodes qui permettent de prendre en compte la flexibilité des équipes d'agents dans un système, sans pour autant qu'il n'existe de cadre généralisé ou de modèle standard universellement reconnu. En particulier, la question de la planification de la formation dynamique dynamique d'agents dans un environnement partiellement observable n'a été que peu, voire pas, étudiée jusqu'à présent dans la littérature. Notons que l'on ne cherche pas uniquement à faire de la planification pour former et reformer des équipes d'agents au gré de l'évolution de la situation. Il faut également que l'on puisse calculer une politique pour chacune des équipes possibles qui sera déployée.

Dans ce chapitre, nous présentons notre première tentative pour étendre le modèle des DEC-POMDPs au cas ouvert où les agents peuvent entrer et sortir durant l'exécution. Ce chapitre est divisé en trois grandes sections qui correspondent aux trois contributions principales :

1. **DEC-POMDP ouvert** : dans la section 5.3, nous introduisons un modèle de planification multi-agents sous incertitude en milieux ouverts : le DEC-POMDP ouvert. Il s'agit de notre première tentative de modèle pour couvrir l'ensemble des cas de systèmes multi-agents. En particulier, notre modèle permet de modéliser à la fois les situations où les entrées et sorties sont déterministes et les situations où elles sont non-déterministes. Nous donnons également quelques définitions relatives à ce modèle, notamment en ce qui concerne les processus d'entrée et sortie des agents. Enfin, nous décrivons comment faire de la prise de décision avec ce modèle.

2. **Meilleures réponses** : dans la section 5.4, nous présentons notre algorithme de résolution pour DEC-POMDPs ouverts. Étant donné qu'un DEC-POMDP ouvert souffre d'une complexité algorithmique importante, nous privilégions les méthodes de résolution approximatives. Plus précisément, nous nous basons sur l'algorithme BRD (*Best Response Dynamics*) pour calculer des politiques jointes séparables. Malgré son manque de garanties théoriques sur l'optimalité de la politique jointe calculée, l'algorithme BRD est toutefois assuré de converger vers un équilibre. Nous en profiterons pour rappeler la notion d'équilibre dans le cadre des jeux coopératifs.

3. **Recherche arborescente de politiques** : dans la section 5.5, on applique un algorithme de recherche heuristique afin de calculer les politiques individuelles des agents lors d'une itération de l'algorithme BRD. Notre contribution consiste à adapter un populaire algorithme de recherche arborescente Monte-Carlo au domaine partiellement observable et stochastique qui nous concerne.

Le contenu de ce chapitre a donné lieu à la publication d'un article lors de la Conférence Internationale sur les Outils en Intelligence Artificielle (*IEEE International Conference on Tools with Artificial Intelligence*, ICTAI), présenté en 2017 à Boston, États-Unis [Cohen et al., 2017].

5.2 Vers un DEC-POMDP ouvert

Comme nous l'avons vu au début du chapitre 4 sur les équipes flexibles, le modèle de DEC-POMDP est adapté à la planification multi-agents sous incertitude, mais ne permet pas de prendre en compte une potentielle évolution de la structure même du groupe d'agents qui agit sur le système. Étant donné un ensemble *fixe* d'agents, les algorithmes de résolution de DEC-POMDPs existants se contentent de calculer une politique jointe séparable, de préférence optimale. Si l'on souhaite décrire un modèle formel et concevoir des algorithmes qui permettent de prendre en compte l'aspect ouvert d'une équipe flexible, il nous faut donc partir de la définition d'un DEC-POMDP et l'augmenter ou la modifier de façon adéquate.

5.2.1 Le problème TARS

Dans cette section, nous introduisons formellement le problème de la planification sur les actions des agents et sur la formation dynamique d'équipes : le **problème TARS** (*Team Action and Reformation Scheduling*). S'attacher à définir précisément le problème que l'on se pose est une étape primordiale si l'on souhaite proposer un modèle qui réponde correctement à notre problématique.

> **Team Action and Reformation Scheduling** (**TARS**). Dans un environnement dynamique, stochastique et partiellement observable, permettre à un ensemble d'agents coopératifs décentralisés de se reformer, de changer leurs rôles ou de modifier la configuration de l'équipe afin que celle-ci puisse s'adapter à la tâche en cours.

Cette définition peut se décomposer en trois parties.

1. *Dans un environnement dynamique, stochastique et partiellement observable ...* : cela signifie que l'on se place dans le contexte décrit dans le chapitre 2 introductif de cette thèse. On considère des environnements complexes où les actions des agents ont des issues incertaines et où ils n'observent pas parfaitement leur environnement.
2. *... permettre à un ensemble d'agents coopératifs décentralisés de se reformer, de changer leurs rôles ou de modifier la configuration de l'équipe ...* : cela a trait à la modification de la structure de l'équipe opérationnelle durant l'exécution de la tâche. Des agents peuvent joindre l'équipe, d'autres peuvent la quitter, certains agents peuvent échanger leurs rôles, un agent peut se spécialiser dans un domaine, etc., le tout sachant que les communications entre agents sont limitées ou impossibles.
3. *... afin que celle-ci puisse s'adapter à la tâche en cours.* : cela concerne la politique, le plan ou la stratégie effectivement suivis par les agents faisant partie de l'équipe opérationnelle. Un agent qui a changé de rôle va devoir modifier son comportement en conséquence, un agent qui vient de rejoindre l'équipe va suivre une nouvelle politique pour soutenir ses équipiers, un agent tombé en panne va devoir être remplacé, etc.

Le problème TARS vise donc à couvrir les cas où l'évolution d'une tâche en cours de résolution favorise le changement de l'équipe d'agents actuellement mobilisée. En particulier, le problème TARS permet de représenter les entrées et sorties des agents, qu'elles soient contrôlées ou non contrôlées, comme présenté dans la section 4.1.

Nous allons le voir, tous les problèmes de planification multi-agents sous incertitude ne sont pas sujets à la formation dynamique de l'équipe d'agents. Dans certains cas, on sait à l'avance que l'équipe ne changera pas. Dans la section suivante, on compare certains benchmarks de DEC-POMDPs pour illustrer la pertinence du problème TARS dans certaines situations.

Benchmark	Description	Nombre d'agents
Broadcast Channel	Deux personnes communiquent sur le même canal.	2
Fire Fighting	Des pompiers éteignent un feu qui se propage.	2 ou 3
Mars Rover	Des rovers récoltent des échantillons sur Mars.	2
Box pushing	Des agents poussent des boîtes pour les ranger dans une certaine zone.	2
Dec-Tiger	Des agents doivent ouvrir l'une des deux portes en face d'eux. Derrière l'une se cache un trésor ; derrière l'autre, un tigre affamé.	2

Tableau 5.1 – Exemple de benchmarks de DEC-POMDP avec leur nombre d'agents impliqués.

5.2.2 Analyse de benchmarks

Bien souvent, c'est le concepteur du problème ou de l'application qui décide, avant de procéder à l'étape de planification, combien d'agents vont être utilisés pour résoudre la tâche en question. Du moins dans le domaine de la recherche dans les DEC-POMDPs, il n'y a pas, ou presque pas, de phase de formation initiale, d'allocation de rôles ou de formation dynamique. La figure 5.1 donne quelques exemples de benchmarks régulièrement utilisés dans l'évaluation des algorithmes de résolution de DEC-POMDPs, ainsi qu'une brève description et le nombre d'agents employés dans le benchmark.

Analysons en détail deux des benchmarks illustrés sur la figure 5.1, et voyons en quoi ceux-ci sont adaptés, ou non, à la formation et la formation dynamique d'équipes.

Broadcast channel. Dans ce benchmark, Alice et Bob essaient de communiquer l'un avec l'autre en s'envoyant des messages sur un même canal partagé. Si Alice et Bob émettent un message en même temps, le signal se trouve bruité et aucune information ne parvient à aucun des deux. Si en revanche, l'un seulement émet pendant que l'autre écoute le canal, alors le message arrive à destination. Seulement, Alice et Bob ne peuvent pas se mettre d'accord "en direct" sur qui émet et qui écoute. Ce problème se modélise bien sous la forme d'un DEC-POMDP. Il est possible de concevoir une politique pour coordonner les agents en amont de l'exécution (pour que l'un écoute pendant un certain moment avant de commencer à émettre, par exemple). Ici, le nombre d'agents fait parfaitement sens compte tenu du contexte. C'est dans la nature même du problème d'avoir un ensemble d'agents qui reste fixe durant l'exécution.

Mars Rover. Dans ce benchmark, deux astromobiles autonomes prélèvent des échantillons sur la surface d'une planète inhabitée. Les échantillons à prélever (des carottes du sol extraterrestre) sont répartis sur la surface selon une certaine distribution. Tous ne sont pas équivalents en terme de composition chimique, aussi les astromobiles doivent se séparer pour prélever autant d'échantillons distincts que possible, car tous présentent un intérêt scientifique notable. Dans ce contexte, la formation initiale d'une équipe est une étape qui semble bien plus importante que

(a) Alice et Bob partagent le même canal.

(b) Deux astromobiles prélèvent des échantillons sur la même planète.

FIGURE 5.1 – Deux benchmarks utilisés pour évaluer l'efficacité des algorithmes de résolution de DEC-POMDPs.

pour le benchmark précédent. En effet, on peut vouloir envoyer un certain type d'astromobile, portant un certain équipement, prélever des échantillons en particulier. De plus, au fur et à mesure que certains échantillons sont récoltés, un astromobile peut décider de rentrer à la station afin d'économiser sa batterie. D'où le besoin de reformer l'équipe, en réassignant un astromobile à une autre zone par exemple, pour conserver un comportement optimal.

Ces deux benchmarks sont assez similaires en ce sens qu'ils mettent tous deux en scène des agents ayant à se partager une ressource commune, soit spatialement (les astromobiles explorent des zones différentes d'une même planète), soit temporellement (Alice et Bob communiquent chacun leur tour sur le même canal). Sans coordination préalable, les agents risquent de se gêner mutuellement et donc de faire baisser l'utilité globale de leur association. Dans un cas en revanche, celui du *Mars Rover*, la question de la formation et de la formation dynamique de l'équipe fait sens. Plutôt que de se demander : *Comment utiliser les deux astromobiles pour optimiser leur rendement ?* on se demande : *Comment optimiser le rendement de la mission ?* Évidemment, certaines contraintes doivent toujours être prises en compte. On ne peut pas simplement envoyer un million d'astromobiles. Ce serait peut-être optimal d'un point de vue scientifique, mais probablement pas d'un point de vue logistique et économique.

Tous les problèmes et toutes les situations ne se prêtent donc pas à la (re)formation d'équipes. Parfois, l'ensemble des agents est fixe et ne peut (ou ne doit) pas être modifié durant l'exécution. De plus, il faut remettre les choses dans leur contexte. Les deux benchmarks décrits ci-dessus servent à évaluer l'efficacité des algorithmes de résolution de DEC-POMDP, pas à modéliser une situation réaliste où le nombre d'agents ne seraient pas une constante fixée dès le départ. Néanmoins, cette brève analyse a montré qu'il est possible de trouver des situations où la formation et la formation dynamique d'équipes sont des caractéristiques importantes pour permettre la résolution optimale d'une tâche. C'est à ce genre de situations que le problème TARS s'adresse et auxquelles nous voulons proposer une solution dans cette thèse.

5.3 DEC-POMDPs ouverts

La principale motivation pour l'introduction d'un nouveau modèle de décision est que les modèles actuels, tels que les DEC-POMDPs, ne semblent pas avoir été conçus pour les problèmes ouverts. Cela ne signifie pas que nous ne pouvons pas trouver un moyen d'appliquer ces modèles aux problèmes qui nous intéressent. Alors, est-il possible d'utiliser un DEC-POMDP pour faire de la planification dans les systèmes ouverts, où les agents entrent et sortent pour optimiser la résolution de la tâche ? La réponse est oui. Dans cette section, on propose donc de passer par

l'utilisation d'un DEC-POMDP pour représenter les situations où les agents peuvent entrer et sortir à leur gré du système. Nous allons le voir, cette approche possède à la fois des avantages et des inconvénients.

Le modèle que l'on introduit, l'OPEN-DEC-POMDP, est une sur-classe des DEC-POMDPs et permet de représenter, en plus des cas généralement traités par les DEC-POMDPs, de nombreuses situations où la population des agents est variable. Notre modèle, en accord avec le problème TARS, peut prendre en compte les cas où un agent tombe en panne, où un agent rejoint le système de façon inopinée, etc. C'est-à-dire à la fois les cas sur lesquels on, en tant que décideurs, garde un contrôle, et les cas où les entrées et sorties sont hors de contrôle. Toutefois, nous allons principalement nous focaliser sur une sous-classe de notre modèle où l'on peut garder un contrôle total sur l'équipe d'agents.

5.3.1 Définition du modèle

Un **processus décisionnel de Markov décentralisé, partiellement observable et ouvert** [Cohen et al., 2017] (*Open Decentralized Partially Observable Markov Decision Process*, **OPEN-DEC-POMDP**, aussi appelé **POMDP décentralisé ouvert** ou **DEC-POMDP ouvert**) est une généralisation du modèle de DEC-POMDP au cas où les agents peuvent entrer et sortir du système durant l'exécution.

Définition 5.3.1 - ***Processus décisionnel de Markov décentralisé, partiellement observable et ouvert* :** Un OPEN-DEC-POMDP pour une population de n agents est un tuple

$$(\mathcal{N},\mathcal{C},\mathcal{S},\mathcal{A},\mathcal{O},D,R,b^0,h),$$

où

- $\mathcal{N} = \{1,2,\ldots,n\}$ est l'ensemble fini des $n \in \mathbb{N}$ agents du système ;
- $\mathcal{C} \subseteq 2^{\mathcal{N}}$ est l'ensemble fini des équipes possibles basées sur la population $\mathcal{N}$; $2^{\mathcal{N}}$ désigne l'ensemble des parties de $\mathcal{N}$;
- $\mathcal{S}$ est l'ensemble fini des états du système ;
- $\mathcal{A} = \{\mathcal{A}_i \mid i \in \mathcal{N}\}$ est l'ensemble des actions ; $\mathcal{A}_i$ est l'ensemble fini des actions individuelles de l'agent $i \in \mathcal{N}$; $\mathcal{A}_C = \times_{i \in C}\mathcal{A}_i$ est l'ensemble des actions jointes a_C de l'équipe $C \in \mathcal{C}$;
- $\mathcal{O} = \{\mathcal{O}_i \mid i \in \mathcal{N}\}$ est l'ensemble des observations ; $\mathcal{O}_i$ est l'ensemble fini des observations individuelles de l'agent $i \in \mathcal{N}$; $\mathcal{O}_C = \times_{i \in C}\mathcal{O}_i$ est l'ensemble des observations jointes o_C de l'équipe $C \in \mathcal{C}$;
- $D : \mathcal{C} \times \mathcal{S} \times \mathcal{O}_{\mathcal{C}} \times \mathcal{C} \times \mathcal{S} \times \mathcal{A}_{\mathcal{C}} \to [0,1]$ est la fonction de dynamique du modèle, où $D(C',s',o_{C'},C,s,a_C) = Pr(C',s',o_{C'} \mid C,s,a_C)$ est la probabilité de transiter d'une équipe $C \in \mathcal{C}$ à une équipe $C' \in \mathcal{C}$, que les agents de l'équipe C' fassent l'observation jointe $o_{C'} \in \mathcal{O}_{C'}$ et que le système passe dans l'état $s' \in \mathcal{S}$ après que les agents de C ont pris l'action jointe $a_C \in \mathcal{A}_C$ dans l'état $s \in \mathcal{S}$;
- $R : \mathcal{S} \times \mathcal{A}_{\mathcal{C}} \times \mathcal{S} \to \mathbb{R}$ est la fonction de récompense, telle que $R(s,a_C,s')$ est la récompense obtenue quand le système passe dans l'état $s' \in \mathcal{S}$ après que les agents de $C \in \mathcal{C}$ ont pris l'action jointe $a_C \in \mathcal{A}_C$ dans l'état $s \in \mathcal{S}$;
- b^0 est l'état de croyance initial, une distribution de probabilité sur les états de $\mathcal{S}$;
- h est l'horizon de planification.

Dans un OPEN-DEC-POMDP, à chaque instant t, une seule équipe d'agents $C^t \in \mathcal{C}$ est présente dans le système. Cette équipe est appelée **équipe opérationnelle**, et un agent $i \in \mathcal{N}$ est un **agent actif** à l'instant t s'il appartient à l'équipe opérationnelle : $i \in C^t$. Pour le reste, un OPEN-DEC-POMDP se déroule sensiblement comme un DEC-POMDP normal. L'état initial s^0 est échantillonné aléatoirement selon la distribution de probabilité initiale b^0. Les agents de l'équipe opérationnelle effectuent une action jointe et font transiter le système d'un état à un autre. L'équipe opérationnelle change alors (ou peut également rester la même), et la nouvelle équipe opérationnelle fait une observation. Ce processus se répète jusqu'à ce que l'horizon de planification h soit atteint.

La différence primordiale entre un DEC-POMDP et un OPEN-DEC-POMDP vient de la fonction de dynamique D qui contrôle :

1. la transition d'un état du système vers un autre état sous l'effet de l'action jointe de l'équipe opérationnelle ;
2. le changement d'équipe opérationnelle intervenant entre deux étapes de décision ;
3. l'émission de l'observation que la nouvelle équipe opérationnelle va faire.

Contrairement à un DEC-POMDP donc, un agent peut quitter le système (et par là même, l'équipe opérationnelle) d'une étape de décision à l'autre suivant l'état actuel du système ou suivant son propre état interne. De façon similaire à un jeu de coalitions tel que présenté dans la définition 4.3.4, les agents de la population $\mathcal{N}$ se divisent, à chaque instant t, en deux sous-groupes, ou coalitions : l'équipe opérationnelle C^t et la **réserve** $\mathcal{N} \backslash C^t$. Le **processus d'entrées et sorties**, ou **processus de transition d'équipes**, est la fonction qui gouverne le changement d'équipe opérationnelle, qui décrit à chaque étape de décision avec quels agents la tâche à résoudre sera traitée. Nous allons le voir, il existe différents types de processus d'entrées et sorties.

5.3.2 Niveaux de contrôle du processus d'entrées et sorties

Définition 5.3.2 - *Processus de transition d'équipes non contrôlé* : Soit M un OPEN-DEC-POMDP. Le processus de transition d'équipes sous-jacent à M décrivant la variation de la population d'agents dans le système est dit **non contrôlé** s'il existe deux fonctions $\psi : \mathcal{C} \times \mathcal{C} \rightarrow [0,1]$ et $T : \mathcal{S} \times \mathcal{O} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ telles que

$$D(C',s',o_{C'},C,s,a_C) = \psi(C,C')T(s',o_{C'},a_C,s), \tag{5.1}$$

où $\psi(C',C) = Pr(C' \mid C)$ est la probabilité que l'équipe opérationnelle passe de C' à C, et $T(s',o_{C'},a_C,s) = Pr(s',o_{C'} \mid a_C,s)$ est la probabilité que la nouvelle équipe opérationnelle C' reçoive l'observation jointe $o_{C'}$ et que le système transite dans l'état s' après que l'ancienne équipe opérationnelle C a effectué l'action a_C dans l'état s.

La définition 5.3.2 dit que le processus de transition d'une équipe à une autre est **non contrôlé** si le processus par lequel les agents entrent et sortent du système est hors de leur portée. De façon plus générale, et comme on l'a déjà partiellement décrit en introduction du chapitre 4, on peut distinguer différents niveaux de contrôle concernant le processus de transition d'équipes :

1. **Aucun contrôle.** C'est le type de processus tel que décrit dans l'équation 5.1. Les agents ne possèdent pas d'actions explicites qui leur permettent de rejoindre l'équipe opérationnelle ou de la quitter. De plus, ils ne peuvent pas agir sur l'état du système via leurs actions pour, d'une façon ou d'une autre, influer sur l'évolution de l'équipe opérationnelle.

Le destin des agents est déterminé. Par exemple, les processus de naissance et de mort (*birth-death processes*) forment une famille de chaînes de Markov à temps continu modélisant le nombre de particules dans un système, où chaque particule peut mourir ou donner vie à une autre particule. Il s'agit de modèles utilisés en biologie des populations [Novozhilov et al., 2006] ou en théorie des files d'attente [Kendall, 1953].

2. **Contrôle total.** Les agents peuvent entrer et sortir à leur guise du système. Formellement, tout agent $i \in \mathcal{N}$ est (au moins) pourvu d'actions $\{\textit{Rejoindre}, \textit{Quitter}\} \subseteq \mathcal{A}_i$ et d'une observation $\textit{Rien} \in \mathcal{O}_i$. Ces actions et observations spéciales vérifient les ensembles d'équations suivants :

$$\begin{cases} a_i^t = \textit{Rejoindre} & \Rightarrow i \in C^{t+1} \\ a_i^t = \textit{Quitter} & \Rightarrow i \notin C^{t+1} \\ i \notin C^t, a_i^t \neq \textit{Rejoindre} & \Rightarrow i \notin C^{t+1} \\ i \in C^t, a_i^t \neq \textit{Quitter} & \Rightarrow i \in C^{t+1} \end{cases} \tag{5.2}$$

et

$$\begin{cases} i \in C^t \Rightarrow o_i^t \neq \textit{Rien} \\ i \notin C^t \Rightarrow o_i^t = \textit{Rien} \end{cases} \tag{5.3}$$

où $C^t, C^{t+1} \in \mathcal{C}$ sont les équipes opérationnelles aux instants t et $t+1$, respectivement. Ainsi, lorsque le contrôle sur le processus de transition d'équipes est total, les agents ne peuvent pas entrer et sortir de l'équipe opérationnelle autrement que par eux-mêmes. Dans ce cas-là, la fonction de dynamique D ne peut jamais être écrite sous la forme factorisée présentée dans l'équation 5.1.

3. **Contrôle indirect.** Ici, les agents ne disposent pas d'actions explicites comme dans le cas du contrôle total. Néanmoins, les agents peuvent indirectement influer sur l'évolution de la composition de l'équipe opérationnelle en prenant les actions adéquates et en faisant transiter le système vers des états où l'équipe va devoir être reformée. Par exemple, un drone de surveillance peut épuiser sa batterie et finir par tomber en panne, certaines de ses actions lui demandant différentes quantités d'énergie. Le choix de tomber en panne n'est en revanche pas explicitement encodé dans son ensemble d'actions.

4. **Contrôle mixte.** Il s'agit du cas où le processus de transition d'équipes n'est ni totalement hors de contrôle, ni totalement contrôlé, ni totalement indirectement contrôlé. Certains agents peuvent être pourvus des actions *Rejoindre* et *Quitter*, et d'autres non. De plus, une influence extérieure aux agents peut affecter la composition de l'équipe opérationnelle en faisant entrer et/ou sortir des agents au cours de l'exécution du processus. Enfin, les agents, par leurs actions, ont une influence indirecte et peuvent entrer ou sortir de l'équipe opérationnelle suivant leur état interne ou l'état du système.

Ces quatre niveaux de contrôle sont directement définis via la fonction de dynamique D. Dans le reste de ce chapitre, nous n'allons considérer que la sous-classe des **OPEN-DEC-POMDPs à contrôle total**. Il s'agit du cas le plus simple où le comportement des agents est parfaitement sous le contrôle du planificateur (ou du décideur humain). Il s'agit également du seul cas pris en compte dans la définition du problème TARS, visant à permettre l'adaptation d'une équipe d'agents à l'évolution de la situation. Un OPEN-DEC-POMDP à contrôle total est donc exactement un DEC-POMDP. Sauf mention contraire, nous utiliserons simplement le terme d'OPEN-DEC-POMDP pour désigner un OPEN-DEC-POMDP à contrôle total.

Notons qu'un OPEN-DEC-POMDP est un modèle permettant de représenter des processus discrets. À ce titre, on ne suppose pas de *temps de transition* d'une équipe opérationnelle à

l'autre. Par exemple, on néglige le temps de trajet nécessaire à un agent pour sortir de l'équipe ou pour la rejoindre. On suppose que le changement d'équipe opérationnelle se fait instantanément entre deux étapes de décision. Il s'agit là d'une hypothèse pour laquelle on peut trouver à la fois des exemples et des contre-exemples dans le cas de certaines applications réelles. Ainsi, lorsque l'entraîneur d'une équipe de football décide de procéder à un changement de joueur, le match est temporairement arrêté. Bien que le chronomètre tourne toujours, l'action est stoppée par l'arbitre pour laisser le temps aux joueurs concernés de procéder au changement. Pendant ce temps, les autres joueurs n'effectuent plus d'actions, et le jeu ne reprend que lorsque le changement a été effectué. À l'inverse, il arrive que le passage d'une équipe opérationnelle à une autre ne soit pas instantané, par exemple lorsqu'un drone de surveillance doit quitter son hangar pour aller rejoindre le périmètre à surveiller à l'extérieur de la base. Nous ignorons cet aspect dans le reste de nos recherches.

5.3.3 Coûts des transitions d'équipes

Outre le niveau de contrôle sur le processus d'entrées et sorties, la transition d'une équipe à une autre dans un OPEN-DEC-POMDP peut également être caractérisée par un éventuel **coût**. Dans beaucoup d'applications réelles, il y a un coût non trivial pour un agent à rejoindre une équipe, par exemple lorsqu'il a besoin de se positionner au sein de l'équipe ou de s'adapter à la vitesse de déplacement de ses coéquipiers. De même, il peut y avoir un coût à quitter une équipe (qui risque d'incomber principalement aux membres restants de l'équipe), par exemple si un agent hors-service provoque un accident ou abîme un produit dont il devait s'occuper. Ces coûts peuvent aussi traduire une pénalité "virtuelle" pour représenter le temps de transition entre deux équipes opérationnelles : certes on considère que la transition d'une équipe à l'autre est immédiate, mais on peut quand même pénaliser les changements d'équipes conséquents.

Le modèle de OPEN-DEC-POMDP permet de représenter ces situations où le changement d'équipe induit un coût, coût qui n'est pas lié directement aux actions des agents (et donc à l'efficacité d'une équipe à résoudre sa tâche) mais qui est lié à la transition entre deux équipes opérationnelles. La définition de tels coûts, spécifiée dans la fonction de récompense R, est liée au domaine étudié et varie donc d'un cas à l'autre. Lorsque l'on considère qu'il n'y a pas de coût à passer d'une équipe opérationnelle à une autre, on dit que le processus de transition d'équipes de l'OPEN-DEC-POMDP décrivant la variation de la population d'agents dans le système est **à transitions gratuites**.

5.3.4 Historiques, politiques et résolution

Comme pour un DEC-POMDP, résoudre un OPEN-DEC-POMDP consiste à trouver, pour la population entière d'agents N, une politique jointe séparable

$$\pi = (\pi_1,\pi_2,\ldots,\pi_n) \tag{5.4}$$

où $\pi_i : \vec{\mathcal{O}}_i \to \mathcal{A}_i$ est la politique individuelle de l'agent $i \in \mathcal{N}$, une application associant, à chaque étape de décision t, une action individuelle $a_i^t = \pi_i(\vec{o}_i^t) \in \mathcal{A}_i$ à chaque historique individuel d'observations $\vec{o}_i^t \in \vec{\mathcal{O}}_i$. Remarquons que l'action jointe $\pi(\varnothing)$ induit l'équipe opérationnelle $C^0 = \{i \in \mathcal{N} \mid \pi_i(\varnothing) \neq \textit{Rien}\}$ qui est initialement utilisée à l'instant $t = 0$.

La politique jointe recherchée π est la politique qui maximise la somme espérée des récompenses cumulées sur l'horizon de planification h, en commençant à l'instant $t = 0$ depuis l'état de croyance initial b^0 :

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{h-1} R^t \mid \pi,b^0\right] \tag{5.5}$$

où $\mathbb{E}[\cdot]$ désigne l'opérateur de l'espérance mathématique et où les $(R^t)_{t\in\{0,1,\ldots,h-1\}}$ correspondent aux récompenses obtenues en suivant la politique jointe π depuis l'état de croyance initial b^0. Cette équation est fondamentalement identique à l'équation 3.35 concernant l'optimalité d'une politique jointe dans un DEC-POMDP. On peut donc également se servir de l'équation 3.31 pour calculer la valeur d'une politique d'OPEN-DEC-POMDP.

5.3.5 Avantages et inconvénients

Comme nous l'avons dit, un OPEN-DEC-POMDP est un DEC-POMDP où les agents disposent d'actions et d'observations supplémentaires pour "simuler" leur sortie de l'équipe opérationnelle. Un avantage de cette approche est qu'il est alors possible d'appliquer l'ensemble des algorithmes de résolution de DEC-POMDPs pour résoudre un OPEN-DEC-POMDP, comme ceux présentés dans la section 3.4.4.

Néanmoins, il reste que la résolution optimale d'un DEC-POMDP est une étape difficile, demandant des temps de calcul parfois exceptionnellement longs. Cela est d'autant plus vrai que, dans le cas de notre modèle, la population d'agents $\mathcal{N}$ doit souvent être grande si l'on veut pouvoir modéliser de nombreuses équipes, chacune adaptée aux différentes situations d'une même tâche à résoudre. Les problèmes à deux ou trois agents seulement, tels que ceux présentés dans le tableau de benchmarks 5.1, ne sont pas vraiment réalistes comparés aux situations que l'on souhaite pouvoir étudier dans le cadre du problème TARS. Un autre inconvénient, toujours lié au problème de complexité, est qu'un OPEN-DEC-POMDP est un DEC-POMDP "augmenté", avec des actions et observations supplémentaires pour permettre la formation dynamique de l'équipe opérationnelle. Autrement dit, il est possible de prendre un DEC-POMDP simple (comme l'un des benchmarks du tableau 5.1) et de le rendre *ouvert* en ajoutant l'équipe opérationnelle dans l'espace d'états et en augmentant les espaces d'actions et d'observations de chaque agent avec des actions *Rejoindre* et *Quitter* et l'observation *Rien* (en plus de devoir modifier les modèles de transition, d'observation et de récompense en conséquence). En d'autres termes, un OPEN-DEC-POMDP (à contrôle total, rappelons-le) *est* un DEC-POMDP, seulement encore plus riche, et donc plus complexe à résoudre, du point de vue du temps de calcul et de l'espace en mémoire requis. Les méthodes de résolution optimales existantes pour les DEC-POMDPs n'ont donc que peu de chances de pouvoir être appliquées telles quelles avec succès. En conclusion, le modèle d'OPEN-DEC-POMDP doit posséder des caractéristiques et propriétés structurelles particulières qui le différencient réellement du modèle de DEC-POMDP. Ces propriétés doivent pouvoir être utilisées afin de calculer des politiques de façon efficace dans notre modèle.

Afin de pouvoir calculer rapidement des politiques jointes pour un OPEN-DEC-POMDP, nous privilégions les approches approximatives.

5.4 L'algorithme BRD

Dans cette section, nous présentons l'algorithme de planification hors-ligne que nous avons choisi pour calculer des politiques jointes séparables.

5.4.1 Équilibre de Nash

Un algorithme de planification utilisé pour résoudre des DEC-POMDPs classiques est l'algorithme JESP (*Joint Equilibrium Search of Policies*) [Nair et al., 2003c]. Cet algorithme est basé sur l'algorithme plus général, utilisé en théorie des jeux, de recherche de meilleures réponses : l'**algorithme BRD** (*Best-Response Dynamics*). Comme l'indique son nom, le principe de cet

algorithme vise à calculer des stratégies (ou politiques) individuelles à chaque joueur (ou agent), telles qu'elles soient toutes de meilleures réponses les unes aux autres.

Avant d'introduire en détail cet algorithme dans la section suivante, nous commençons par donner les définitions de meilleures réponses et d'équilibre de Nash.

Définition 5.4.1 - *Politique de meilleure réponse* : Une politique de meilleure réponse d'un agent $i \in \mathcal{N}$ à une politique jointe π_{-i} des autres agents est une politique individuelle $\bar{\pi}_i$ telle que, pour toute autre politique individuelle π_i de cet agent, on a :

$$\mathbb{E}\left[\sum_{t=0}^{h-1} R^t \mid b_0,\langle\bar{\pi}_i,\pi_{-i}\rangle\right] \geqslant \mathbb{E}\left[\sum_{t=0}^{h-1} R^t \mid b_0,\langle\pi_i,\pi_{-i}\rangle\right] \tag{5.6}$$

ou, de façon équivalente :

$$V(\langle\bar{\pi}_i,\pi_{-i}\rangle) \geqslant V(\langle\pi_i,\pi_{-i}\rangle). \tag{5.7}$$

Une politique de meilleure réponse d'un agent (ou, plus simplement, la meilleure réponse d'un agent), est la meilleure politique que cet agent puisse suivre étant donnée la politique actuelle suivie par les autres agents. Comme tous les agents sont coopératifs, la meilleure réponse $\bar{\pi}_i$ d'un agent $i \in \mathcal{N}$ doit donc être celle qui maximise la fonction de valeur $V(\pi)$ de la politique jointe $\pi = \langle\bar{\pi}_i\pi_{-i}\rangle$ *sachant fixe* la politique jointe π_{-i} des autres agents. Il ne doit pas exister d'autres politiques que l'agent i pourrait suivre et qui permettraient à l'équipe d'accumuler de meilleures récompenses. De cette notion de meilleure réponse découle le concept d'**équilibre de Nash**.

Définition 5.4.2 - *Équilibre de Nash (cas coopératif)* : Soit M un OPEN-DEC-POMDP. Une politique jointe $\pi^* = (\pi_1^*,\pi_2^*,\ldots,\pi_n^*)$ pour les n agents de M est un **équilibre de Nash** si aucun agent $i \in \mathcal{N}$ ne peut unilatéralement changer de politique individuelle π_i^* sans faire baisser la valeur de la politique jointe π^* :

$$\forall i \in \mathcal{N},\ \forall \pi_i \in \Pi_i,\ V(\langle\pi_i^*,\pi_{-i}^*\rangle) \geqslant V(\langle\pi_i,\pi_{-i}^*\rangle), \tag{5.8}$$

où Π_i est l'ensemble des politiques individuelles de l'agent i.

Bien que le concept d'équilibre de Nash soit historiquement lié au domaine de la théorie des jeux *compétitifs*, son utilisation dans le cas coopératif se fait intuitivement. On note NE (*Nash Equilibrium*) l'ensemble des politiques jointes qui sont des équilibres de Nash.

5.4.2 Best Response Dynamics

L'algorithme 1 donne les grandes lignes de notre méthode de résolution.

L'algorithme BRD est une approche directe pour trouver des meilleures réponses. Il commence en initialisant une politique jointe arbitrairement (ligne 1). Généralement, on choisit une politique jointe π aléatoire (où chaque politique individuelle est elle-même initialisée aléatoirement). L'algorithme itère ensuite jusqu'à trouver un équilibre de Nash (ligne 2). À chaque itération, il sélectionne un agent $i \in \mathcal{N}$ (ligne 3) et calcule sa meilleure réponse $\bar{\pi}_i$ en supposant fixe la politique jointe π_{-i} des autres agents (ligne 4). Si l'agent sélectionné possédait déjà une politique individuelle, celle-ci est préalablement supprimée et une nouvelle meilleure réponse est calculée. La nouvelle politique jointe $\pi = (\bar{\pi}_i,\pi_{-i})$ est la composition de la meilleure réponse et des politiques fixes des autres agents (ligne 5). Une fois que l'algorithme a trouvé un équilibre de Nash, la boucle principale s'arrête et la politique est retournée (ligne 6).

Algorithme 1 : L'algorithme BRD.

Données : Un OPEN-DEC-POMDP
Résultat : Une politique jointe séparable π

```
Initialiser une politique jointe π de façon arbitraire.
tant que π ∉ NE faire
    Sélectionner un agent i ∈ 𝒩.
    Calculer la meilleure réponse π̄_i de l'agent i au reste des politiques π_{-i} des autres
     agents.
    π ← (π̄_i, π_{-i}).
retourner π
```

L'algorithme BRD n'est en général pas assuré de converger vers un équilibre quelconque (par exemple dans le cas d'un jeu non-coopératif). Lorsqu'il est utilisé pour calculer des politiques jointes pour un DEC-POMDP ou un OPEN-DEC-POMDP à contrôle total, l'algorithme BRD est, en revanche, garanti de converger vers un équilibre de Nash [Shoham and Leyton-Brown, 2008, Nisan et al., 2007], ce qui assure que l'algorithme va bien s'arrêter après un nombre fini d'itérations. Cet algorithme est utilisé hors-ligne, et calcule donc une politique jointe de manière centralisée. La politique jointe résultante est localement maximale (puisque la valeur de la politique à chaque itération ne peut qu'augmenter) et séparable, ce qui assure également qu'elle pourra être exécutée de façon décentralisée par les agents.

Avant de rentrer dans les détails de l'algorithme, remarquons que l'algorithme BRD présente un parallèle évident avec le travail d'équipe ad-hoc tel que présenté en section 4.4.1. Rappelons-le, l'objectif d'un agent ad-hoc est d'apprendre ou d'observer le comportement des agents travaillant avec lui, et de s'adapter afin de pouvoir être le meilleur coéquipier possible. En particulier, si l'équipe avec laquelle l'agent ad-hoc doit se synchroniser suit une certaine politique jointe (possiblement stochastique, mais néanmoins stationnaire, fixe), alors l'agent ad-hoc doit apprendre quelle est cette politique jointe, et se comporter en suivant une politique de meilleure réponse. L'avantage, dans le cas de l'algorithme BRD, c'est que l'agent que l'on choisit d'optimiser n'a pas à apprendre la politique jointe des autres agents. L'algorithme étant centralisé, et tous les agents étant sous notre contrôle, leur politique jointe est connue : nous l'avons nous-même calculée lors d'une itération précédente de l'algorithme BRD.

La boucle principale de l'algorithme consiste en deux étapes.

1. Sélectionner un agent,
2. Calculer sa meilleure réponse.

Nous allons maintenant examiner comment réaliser la sélection des agents. Dans la section suivante, nous étudierons comment faire le calcul des politiques de meilleure réponse dans le cadre d'un OPEN-DEC-POMDP. Nous utiliserons pour cela un algorithme de recherche arborescente basé sur des simulations.

5.4.3 Sélection des agents

Il existe des façons simples pour sélectionner un agent dans une itération de l'algorithme BRD. On peut :

1. soit choisir le prochain agent à optimiser **de façon aléatoire**,
2. soit sélectionner les agents **séquentiellement**, les uns après les autres, toujours dans le même ordre.

Ces deux techniques ont autant de chances l'une que l'autre de produire des résultats similaires, puisqu'elles sont toutes deux arbitraires et ne sont pas guidées par le calcul des meilleures réponses. Cependant, si l'on pouvait se rendre compte que calculer la meilleure réponse d'un certain agent n'augmente que très peu, voire pas du tout, la valeur de la politique jointe, alors on serait moins enclin à sélectionner cet agent lors des prochaines itérations de l'algorithme. De plus, choisir un agent aléatoirement ou séquentiellement n'exploite pas les avantages structurels que le modèle d'OPEN-DEC-POMDP a à offrir. Nous identifions ici un critère lié aux OPEN-DEC-POMDPs pour guider la sélection d'un agent lors d'une itération de l'algorithme BRD : l'état de présence des agents.

Il peut être utile de quantifier le *temps de présence* d'un agent i dans le système – autrement dit, le nombre moyen de pas de temps que celui-ci passe dans l'équipe opérationnelle lorsqu'il suit une certaine politique.

Définition 5.4.3 - *État de présence* : Soit M un OPEN-DEC-POMDP. L'**état de présence** $\tau_i \in \mathbb{R}$ d'un agent $i \in \mathcal{N}$ de M suivant une politique individuelle π_i est défini par :

$$\tau_i = \mathbb{E}\left[\sum_{t=0}^{h-1} Z_i^t \mid b_0, \pi_i\right], \tag{5.9}$$

où $\mathbb{E}[\cdot]$ est l'opérateur de l'espérance mathématique et Z_i^t est une variable aléatoire binaire valant 1 si l'agent i est dans l'équipe opérationnelle C^t à l'instant t :

$$Z_i^t = \begin{cases} 1, \text{ si } i \in C^t \\ 0, \text{ sinon} \end{cases}. \tag{5.10}$$

On peut à présent se servir de l'état de présence d'un agent pour guider la sélection des agents lors de l'étape de sélection de l'algorithme BRD. Par exemple, on peut sélectionner le prochain agent $i \in \mathcal{N}$ dont on souhaite calculer la meilleure réponse selon une loi de probabilité exponentielle normalisée (ou *softmax*)

$$\frac{e^{\tau_i}}{\sum_{k \in \mathcal{N}} e^{\tau_k}}.$$

Cette heuristique de sélection d'agent permet de focaliser la procédure d'optimisation sur les agents qui ont plus de chances d'avoir un impact sur la résolution de la tâche du fait de leur plus grande présence dans le système. Un agent dont la meilleure réponse lui préconise d'être hors de l'équipe opérationnelle la majeure partie du temps verra ainsi rarement sa politique être optimisée à nouveau. Peut-être que cet agent ne présente qu'un intérêt épisodique pour la résolution de la tâche ou qu'il ne possède pas un ensemble d'actions qui lui permette d'être plus efficace que les autres agents.

La section suivante va aborder la seconde étape de la boucle principale de l'algorithme BRD : le calcul de la politique de meilleure réponse.

5.5 L'algorithme POS-UCT

Une fois que l'agent à optimiser est choisi, il faut calculer sa politique de meilleure réponse. Une recherche exhaustive ferait que chaque itération de l'algorithme BRD requerrait une énumération et une évaluation gourmande en temps et en mémoire d'un nombre exponentiel de

politiques individuelles pour trouver la meilleure réponse de l'agent sélectionné. Suivant la taille des espaces d'actions et d'observations et la longueur de l'horizon de planification, une telle recherche exhaustive ne présente que peu d'intérêt.

Il est possible de restreindre le nombre de politiques énumérées en utilisant la programmation dynamique [Hansen et al., 2004], de façon à écarter les politiques individuelles qui sont *dominées* (c'est-à-dire, qui se révéleront être toujours moins bonnes que d'autres politiques). Alternativement, il est possible de voir le problème visant à calculer une meilleure réponse comme un POMDP [Nair et al., 2005]. En effet, lorsque les politiques de tous les agents sauf un sont fixes, alors trouver la politique du-dit agent consiste à résoudre un POMDP simple, avec ce seul agent. Les autres agents ne sont en quelque sorte que des composantes stationnaires de l'environnement. Tous les algorithmes de résolution de POMDP – tels que ceux mentionnés dans la section 3.3.4 – peuvent alors être appliqués pour trouver la politique individuelle optimale de l'agent. Toutefois, lorsque les espaces d'états et d'observations deviennent particulièrement grands, il peut être judicieux de se tourner vers des techniques approximatives où les valeurs des actions sont estimées à base d'échantillonage. Ici, nous adoptons cette approche, basée sur une méthode récente et efficace de résolution des POMDPs, la **recherche arborescente Monte-Carlo** (MCTS, *Monte-Carlo Tree Search*).

5.5.1 Présentation

MCTS est une méthode de recherche heuristique par échantillonnage particulièrement adaptée à la prise de décision dans les domaines qui peuvent être représentés par des séquences d'actions ou des structures arborescentes. Récemment, l'algorithme a été appliqué avec un succès retentissant pour programmer des ordinateurs devenus maîtres au jeu de Go [Silver et al., 2016], mais il a aussi été utilisé pour résoudre efficacement des problèmes de planification, notamment des POMDPs [Silver and Veness, 2010, Auger, 2011, Katt et al., 2017]. Il s'agit donc d'un algorithme approprié pour calculer des politiques individuelles dans le contexte de l'algorithme BRD. Bien que MCTS soit généralement employé pour calculer des politiques de POMDP en-ligne (c'est-à-dire, *durant* l'exécution du processus), nous nous en servons ici en tant que planificateur hors-ligne (en amont de l'exécution) pour calculer la politique individuelle de meilleure réponse d'un agent à chaque itération de l'algorithme BRD. Le lecteur est invité à se référer à [Browne et al., 2012] pour un état de l'art complet assez récent à propos de l'algorithme, de ses différentes variantes, applications et méthodes relatives.

Le principe de base de MCTS consiste à construire un arbre, où les nœuds correspondent à des états du jeu (par exemple, la position des pièces sur un échiquier), et où les arcs correspondent à des actions (par exemple, l'ensemble des coups légaux de chaque pièce). L'arbre est étendu incrémentalement en essayant une multitude d'actions possibles dans chaque nœud prometteur et en simulant des trajectoires de jeu aléatoires afin d'obtenir une évaluation de l'impact à long terme de ces actions à ces nœuds. Une illustration est donnée dans la figure 5.2.

5.5.2 Recherche arborescente Monte-Carlo dans les domaines partiellement observables

L'algorithme 2 donne les grandes lignes de notre algorithme de **recherche arborescente Monte-Carlo partiellement observable et stochastique** (*Partially Observable and Stochastic Monte-Carlo Tree Search*, POS-MCTS), tandis que 3 en donne une version détaillée. La figure 5.3 quant à elle illustre de façon schématique chacune des principales étapes de l'algorithme. L'algorithme POS-MCTS est une variation de l'algorithme *Partially Observable Monte-Carlo*

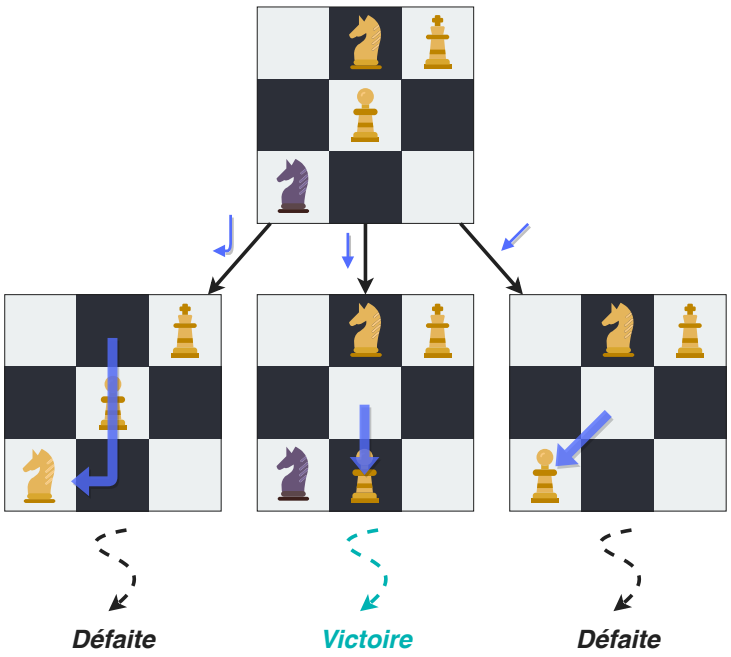


FIGURE 5.2 – Illustration du principe de base de l'algorithme MCTS aux échecs. Par souci de simplicité, seule une partie de l'échiquier est représentée. De plus, seuls certains coups des blancs sont examinés. La valeur d'un coup aux échecs ne se résume pas au gain immédiat obtenu après que le coup a été joué : l'impact à long terme doit être pris en compte. Une estimation de la valeur d'un des coups blancs { ↵ , ↓ , ↙ } dans l'état actuel du jeu peut être obtenue en simulant une trajectoire aléatoire de coups après celui-ci. Si cette trajectoire aboutit sur une victoire pour les blancs, alors le coup n'était pas mauvais. Ici, prendre le cavalier noir (avec ↵ ou ↙) semble être un choix gagnant à première vue, mais, suivant l'état de l'échiquier, agir ainsi pourrait affaiblir la défense du roi blanc.

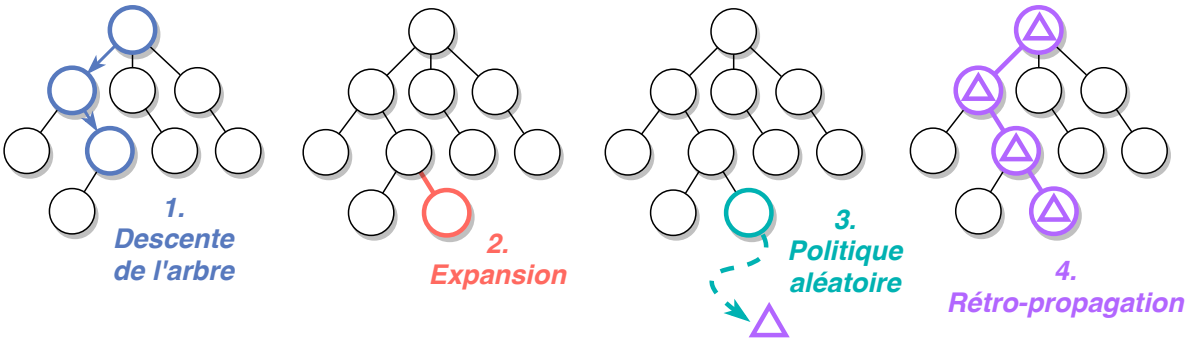


FIGURE 5.3 – Illustration des quatre étapes principales de l'algorithme MCTS. Figure inspirée de [Browne et al., 2012].

Planning (POMCP) [Silver and Veness, 2010]. POS-MCTS est un algorithme de planification hors-ligne, là où POMCP est un algorithme de planification en-ligne.

Dans le contexte d'un POMDP, l'objectif de l'algorithme POS-MCTS est de construire un arbre, appelé *arbre de recherche* (*search tree*) qui va au final correspondre à la politique individuelle stochastique de l'agent dont on cherche la meilleure réponse. Pour plus de clarté, et pour bien se rappeler que l'on cherche toujours à calculer la meilleure réponse d'un agent à un ensemble de politiques fixes, nous allons noter $i \in \mathcal{N}$ l'agent dont nous voulons calculer la politique avec l'algorithme POS-MCTS (même si, en toute rigueur, cela est superflu compte tenu du fait que POS-MCTS est un algorithme de résolution de POMDP).

Algorithme 2 : L'algorithme POS-MCTS.

Données : Un POMDP et $maxIter \in \mathbb{N}$
Résultat : Une politique individuelle π_i

```
1 iter ← 0
2 Initialiser le nœud racine o_i^0
3 tant que iter < maxIter faire
4     o_i^x ← Descente(o_i^0)
5     f(o_i^x) ←Expansion(o_i^x)
6     Δ ← Trajectoire(f(o_i^x))
7     Rétropropagation(Δ, o_i^x)
8     iter ← iter + 1
9 retourner π_i
```

Au début de l'algorithme, on commence avec un arbre enraciné, orienté et composé d'un seul nœud, la *racine*. Un nœud à la profondeur $0 \leqslant t < h$ de l'arbre de recherche correspond à un historique d'observations $\vec{o}_i^t$ de l'agent i et stocke trois informations essentielles :

1. une observation individuelle $o_i \in \mathcal{O}_i$,
2. un vecteur N de **nombre de visites** de taille $|\mathcal{A}_i|$, tel que $N(\vec{o}_i^t,a_i)$ est le nombre de fois que l'action $a_i \in \mathcal{A}_i$ a été essayée à l'historique $\vec{o}_i^t$,
3. un vecteur Q de **valeurs estimées** de taille $|\mathcal{A}_i|$, tel que $Q(\vec{o}_i^t,a_i)$ correspond à la somme des récompenses cumulées estimées pour agir à l'historique $\vec{o}_i^t$ avec l'action a_i.

La racine de l'arbre de recherche contient toujours l'historique d'observations vide $\vec{o}_i^0 = o_i^0 = \varnothing$. Comme l'arbre de recherche est enraciné et dirigé, une chaîne de la racine jusqu'à un nœud de profondeur t dans l'arbre peut être vue comme une séquence d'observations – c'est-à-dire, comme un historique d'observations. Sans perte de généralité, un nœud fera donc référence à un historique d'observations, et *vice versa*.

L'arbre de recherche est ensuite construit de façon itérative en répétant quatre étapes, correspondant aux quatre étapes illustrées sur la figure 5.3.

1. Descente de l'arbre. En partant de la racine $\vec{o}_i^0$, l'agent exécute une certaine séquence d'actions, déterminée par une *politique de descente* (*tree policy*, la fonction `Descente`, ligne 4). À chaque fois que l'agent exécute une action, il se trouve à un certain nœud (ou historique) $\vec{o}_i^t$ et reçoit une observation o_i^{t+1} du système, ce qui lui indique vers quel nœud fils se diriger : $(\vec{o}_i^t,o_i^{t+1})$. S'il n'existe pas de nœud fils dont l'observation corresponde à celle reçue, alors un

Algorithme 3 : L'algorithme POS-MCTS (version étendue).

Données : Un POMDP et $maxIter \in \mathbb{N}$
Résultat : Une politique individuelle π_i

Fonction `Résoudre` **:**
  $iter \leftarrow 0$
  Initialiser le nœud racine $\vec{o}_i^{\,0}$
  $\Psi \leftarrow [\varnothing]$
  tant que $iter < maxIter$ **faire**
    $\vec{o}_i^{\,x} \leftarrow$ `Descente`$(\vec{o}_i^{\,0}, \Psi)$
    $f(\vec{o}_i^{\,x}) \leftarrow$`Expansion`$(\vec{o}_i^{\,x}, \Psi)$
    $\Delta \leftarrow$ `Trajectoire`$(f(\vec{o}_i^{\,x}), \Psi)$
    `Rétropropagation`$(\Delta, \vec{o}_i^{\,x}, \Psi)$
    $iter \leftarrow iter + 1$
  pour tous $\vec{o}_i^{\,t}$ **faire**
    $\pi_i(\vec{o}_i^{\,t}) \leftarrow \underset{a_i \in \mathcal{A}_i}{\operatorname{argmax}} \frac{Q(\vec{o}_i^{\,t},a_i)}{N(\vec{o}_i^{\,t},a_i)}$
  retourner π_i

Fonction `Descente`$(\vec{o}_i^{\,0}, \Psi)$ **:**
  $\vec{o}_i^{\,t} \leftarrow \vec{o}_i^{\,0}$
  tant que $\vec{o}_i^{\,t}$ *n'est pas extensible* **faire**
    $\bar{a}_i^t \leftarrow \underset{a_i \in \mathcal{A}_i}{\operatorname{argmax}} \frac{Q(\vec{o}_i^{\,t},a_i)}{N(\vec{o}_i^{\,t},a_i)} + \chi \cdot \sqrt{\frac{\ln N(\vec{o}_i^{\,t})}{N(\vec{o}_i^{\,t},a_i)}}$
    Effectuer $\bar{a}_i^t$, recevoir o_i^{t+1} et R^{t+1}
    $\vec{o}_i^{\,t} \leftarrow (\vec{o}_i^{\,t},o_i^{t+1})$
    $\Psi \leftarrow \Psi \cup [\bar{a}_i^t,o_i^{t+1},R^{t+1}]$
  retourner $\vec{o}_i^{\,t}$

Fonction `Expansion`$(\vec{o}_i^{\,x}, \Psi)$ **:**
  Choisir $a_i^x \in \mathcal{A}_i$ telle que $N(\vec{o}_i^{\,x},a_i^x) = 0$
  Effectuer a_i^x, recevoir o_i^{x+1} et R^{x+1}
  $f(\vec{o}_i^{\,x}) \leftarrow (\vec{o}_i^{\,x},o_i^{x+1})$
  Concaténer le nœud $f(\vec{o}_i^{\,x})$ à $\vec{o}_i^{\,x}$
  $\Psi \leftarrow \Psi \cup [a_i^x,o_i^{x+1},R^{x+1}]$
  retourner $f(\vec{o}_i^{\,x})$

Fonction `Trajectoire`$(\vec{o}_i^{\,x}, \Psi)$ **:**
  $t \leftarrow x$
  tant que $t < h$ **faire**
    Choisir $a_i^t \in \mathcal{A}_i$ aléatoirement
    Effectuer a_i^t, recevoir o_i^t et R^{t+1}
    $\Psi \leftarrow \Psi \cup [\bar{a}_i^t,o_i^{t+1},R^{t+1}]$
    $t \leftarrow t + 1$
  $\Delta \leftarrow 0$
  pour tous $R^t \in \Psi$ **faire**
    $\Delta \leftarrow \Delta + R^t$
  retourner Δ

Fonction `Rétropropagation`$(\Delta, \vec{o}_i^{\,x}, \Psi)$ **:**
  $\vec{o}_i^{\,t} \leftarrow \vec{o}_i^{\,x}$
  tant que $t \geqslant 0$ **faire**
    Extraire a_i^t de Ψ
    $Q(\vec{o}_i^{\,t},a_i^t) \leftarrow Q(\vec{o}_i^{\,t},a_i^t) + \Delta$
    $N(\vec{o}_i^{\,t},a_i^t) \leftarrow N(\vec{o}_i^{\,t},a_i^t) + 1$
    $t \leftarrow t - 1$

nœud fils est concaténé au nœud actuel avec les informations suivantes :

$$\begin{cases} o_i = o_i^{t+1}, \\ \forall a_i \in \mathcal{A}_i, N(\vec{o}_i^{t+1}, a_i) = 0, \\ \forall a_i \in \mathcal{A}_i, Q(\vec{o}_i^{t+1}, a_i) = 0. \end{cases} \tag{5.11}$$

L'agent descend l'arbre jusqu'à rencontrer un nœud à *étendre*, noté $\vec{o}_i^x$, à une profondeur $0 < x < h$. Un nœud est dit *extensible* si toutes les actions n'ont pas été essayées au moins une fois à l'historique d'observations correspondant – c'est-à-dire s'il existe une action $a_i \in \mathcal{A}_i$ telle que $N(\vec{o}_i^x, a_i) = 0$.

2. Expansion. Depuis ce nœud extensible $\vec{o}_i^x$, l'agent essaie l'une des actions qui n'ont pas encore été essayées et obtient une observation o_i^{x+1} (la fonction `Expansion`, ligne 5). On crée alors un nouveau nœud fils, noté $f(\vec{o}_i^x)$, tel que $f(\vec{o}_i^x) = (\vec{o}_i^x, o_i^{x+1})$, et on connecte ce nouveau nœud à son nœud parent $\vec{o}_i^x$. Il est initialisé via l'équation 5.11.

3. Politique aléatoire. Depuis le nœud nouvellement créé $f(\vec{o}_i^x)$, on suit une *politique aléatoire* (*random* ou *rollout policy*, la fonction `Trajectoire`, ligne 6), en choisissant aléatoirement une action à ce nœud, puis en choisissant de même des actions aléatoires jusqu'à atteindre l'horizon de planification h. Cette séquence d'actions est appelée *trajectoire*. Notons que durant cette étape, aucun nouveau nœud n'est créé ou ajouté à l'arbre. Ces trois premières étapes constituent ce que l'on appelle une *simulation*. La séquence d'actions sélectionnées, d'observations perçues et de récompenses émises par le système peut être enregistrée dans un vecteur spécial, appelé le *chemin*, noté Ψ :

$$\Psi = [\varnothing, a_i^0, o_i^1, R^t, a_i^1, \ldots, a_i^{h-2}, o_i^{h-1}, R^{h-1}],$$

où a_i^t est l'action sélectionnée, o_i^t l'observation perçue et R^t la récompense émise à l'instant t. On note

$$\Delta = \sum_{t=0}^{h-1} R^t$$

la *récompense cumulée* durant la simulation.

4. Rétro-propagation de la récompense. La récompense cumulée Δ est finalement rétro-propagée (fonction `Rétropropagation`, ligne 7) dans tous les nœuds visités durant les étapes 1 et 2, c'est-à-dire entre les instants $t = 0$ et $t = x$. Ces nœuds peuvent être retrouvés en suivant les informations contenues dans la variable chemin Ψ. Formellement, l'équation de mise à jour est :

$$Q(\vec{o}_i^t, a_i) \leftarrow Q(\vec{o}_i^t, a_i) + \Delta. \tag{5.12}$$

Le nombre de visites du nœud est quant à lui incrémenté de un :

$$N(\vec{o}_i^t, a_i) \leftarrow N(\vec{o}_i^t, a_i) + 1. \tag{5.13}$$

Ces quatre étapes constituent une itération de l'algorithme POS-MCTS. À chaque itération, le système démarre dans un état échantillonné aléatoirement depuis l'état de croyance initial b^0. Il est important de noter que, durant l'intégralité du processus, les autres agents de l'OPEN-DEC-POMDP agissent également dans le système, en suivant chacun leur politique individuelle calculée par l'algorithme POS-MCTS dans une itération précédente de l'algorithme BRD (et, s'ils n'en ont pas encore, en suivant une politique aléatoire). Quand l'agent i prend une action

individuelle, le reste des agents prend également une action jointe. De même, lorsqu'il reçoit une observation individuelle, ses coéquipiers reçoivent également une observation jointe. La transition d'un état à l'autre du système se fait selon le modèle normal de l'OPEN-DEC-POMDP. La récompense Δ générée à la fin de la politique aléatoire correspond à une récompense après avoir exécuté l'OPEN-DEC-POMDP une fois, de $t = 0$ à $t = h - 1$, en supposant que tous les agents ont suivi leur propre politique individuelle (qui sont toutes construites avec POS-MCTS dans des itérations différentes de BRD).

La première ligne de la boucle principale de l'algorithme POS-MCTS consiste à utiliser une *politique de descente*. Il s'agit d'une politique guidée qui indique à l'agent l'action à effectuer à chaque étape de décision durant la descente de l'arbre, jusqu'à ce qu'un nœud extensible soit atteint. Pour cette politique, nous utilisons l'*Upper Confidence Bounds* (UCB) [Auer et al., 2002]. Cette stratégie permet de considérer le choix de l'action à un historique comme un problème de bandit manchot à bras multiples [Katehakis and Veinott, 1987]. À chaque étape de décision $t < x$ (x étant l'étape à laquelle la politique de descente se termine), l'action $\bar{a}_i^t$ choisie à l'historique $\vec{o}_i^t$ est :

$$\bar{a}_i^t = \underset{a_i \in \mathcal{A}_i}{\operatorname{argmax}} \frac{Q(\vec{o}_i^t, a_i)}{N(\vec{o}_i^t, a_i)} + \chi \cdot \sqrt{\frac{\ln N(\vec{o}_i^t)}{N(\vec{o}_i^t, a_i)}} \tag{5.14}$$

où

$$N(\vec{o}_i^t) = \sum_{a_i \in \mathcal{A}_i} N(\vec{o}_i^t, a_i) \tag{5.15}$$

est le nombre total de visites de l'historique $\vec{o}_i^t$, et $\chi \in \mathbb{R}^+$ est le *paramètre d'exploration*, utilisé pour faire le compromis entre l'exploration d'actions rarement utilisées et l'exploitation d'actions reconnues comme étant efficaces. Remarquons que la valeur de χ doit être soigneusement choisie selon la forme de la fonction de récompense de l'OPEN-DEC-POMDP. Plus χ a une valeur grande, plus il y a d'actions d'exploration. Quand $\chi \to +\infty$, le terme de gauche

$$\frac{Q(\vec{o}_i^t, a_i)}{N(\vec{o}_i^t, a_i)}$$

de l'équation 5.14, qui mesure la valeur moyenne d'une action, devient négligeable, et UCB choisit alors l'action la moins utilisée. Inversement, quand $\chi = 0$, le terme de droite

$$\sqrt{\frac{\ln N(\vec{o}_i^t)}{N(\vec{o}_i^t, a_i)}},$$

qui est grand pour les actions peu utilisées et petit pour les actions qui ont déjà été souvent utilisées à l'historique $\vec{o}_i^t$, est annulé, et UCB choisit donc de façon gloutonne l'action ayant la meilleure valeur moyenne.

⁎ **Bandit manchot à bras multiples.** La figure 5.4 illustre le concept de bandit manchot à bras multiples dont nous avons fait mention ci-dessus.

Imaginons que l'on se rende au casino afin d'y dépenser des sommes folles dans l'espoir de faire fortune. Toutes les machines à sous du casino (aussi appelées bandits manchots) sont alignées les unes à côté des autres. Toutes possèdent des espérances de gains qui sont inconnues. Peut-être que certaines sont plus souvent gagnantes que d'autres, mais nous ne disposons pas de cette information initialement. On souhaite maximiser nos gains en jouant aux machines ayant les meilleures espérances de gains. Pour trouver ces machines, il n'y a pas d'autres choix que de les essayer toutes, au moins une fois, puis chacune suffisamment de fois pour obtenir une

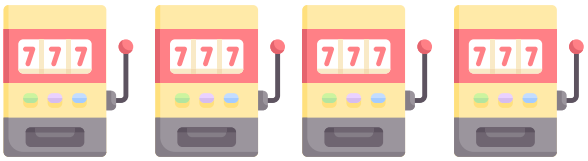


FIGURE 5.4 – Illustration d'un bandit manchot à bras multiples.

bonne estimation des gains de chaque machine. C'est ce que l'on appelle le **compromis exploration/exploitation** (*exploration/exploitation trade-off*). On souhaite essayer plus souvent les machines qui rapportent le plus (exploitation) mais également essayer de temps en temps celles qui jusqu'ici ont moins rapporté, dans l'espoir d'obtenir de meilleurs gains (exploration).

Lorsque la formule UCB est utilisée dans l'algorithme MCTS pour effectuer la sélection des actions lors de la descente de l'arbre, alors l'algorithme est appelé UCT (*Upper Confidence Bound for Trees*) [Kocsis and Szepesvári, 2006]. De la même façon, nous allons à présent, et pour le reste de cette thèse, nous référer à l'algorithme POS-MCTS sous le terme de **POS-UCT** (*Partially Observable and Stochastic UCT*) lorsque l'équation UCB sera utilisée lors de la descente de l'arbre (et, sauf mention contraire, elle le sera toujours).

Si on effectue assez d'itérations dans l'algorithme POS-UCT, nous arriverons à obtenir, dans chaque nœud visité suffisamment de fois, une bonne estimation de la valeur de chaque action effectuée à l'historique d'observations correspondant, et en particulier de meilleures estimations pour les actions prometteuses. Dans la pratique, on se fixe un budget de calcul (paramètre $maxIter$, ligne 3) correspondant au nombre maximum d'itérations que l'on souhaite effectuer (mais il pourrait également s'agir d'une limite de temps), et l'algorithme s'arrête quand le budget alloué est épuisé. L'algorithme retourne une politique individuelle π_i (ligne 9). Cette politique peut être extraite de l'arbre en *déterminisant* ce dernier, par exemple en appliquant une règle de sélection d'actions gloutonne à chaque historique $\vec{o}_i^t$:

$$\pi_i(\vec{o}_i^t) \leftarrow \underset{a_i \in \mathcal{A}_i}{\operatorname{argmax}} \frac{Q(\vec{o}_i^t, a_i)}{N(\vec{o}_i^t a_i)} \tag{5.16}$$

La politique individuelle $\pi_i : \vec{\mathcal{O}}_i \to \mathcal{A}_i$ est une politique de meilleure réponse approximative de l'agent i à la politique jointe π_{-i} des autres agents. Lorsque l'algorithme BRD sélectionne un autre agent $j \neq i \in \mathcal{N}$ à optimiser, sa politique individuelle π_j est supprimée et une nouvelle est calculée avec BRD. Nous appelons **BRD-POS-UCT** l'algorithme de planification visant à construire une politique jointe séparable en calculant des meilleures réponses successives aux agents via l'algorithme UCT appliqué au domaine partiellement observable et stochastique.

Nous terminons cette section en abordant un point de réflexion concernant l'algorithme MCTS appliqué au domaine partiellement observable : que faire si tous les historiques individuels ne sont pas visités durant les différentes itérations de l'algorithme ?

5.5.3 Historiques rares

Dans la première étape de l'algorithme POS-UCT, la descente de l'arbre s'effectue jusqu'à la rencontre de ce que l'on appelle un nœud extensible. Dans l'algorithme MCTS originel, les effets des actions sur le jeu sont supposés déterministes (car l'algorithme MCTS est généralement appliqué à des jeux à information parfaite et complète, tels que le jeu de Go). Un nœud extensible est un nœud dans lequel toutes les actions non pas été essayées au moins une fois ou, de façon équivalente, un nœud dont tous les nœuds fils (correspondant à des états du jeu) n'ont

pas été crées. Dans notre algorithme POS-UCT (c'est-à-dire, dans un cas partiellement observable et stochastique) essayer une action individuelle dans un nœud va générer une observation individuelle qui dépend, d'une part, de la nature stochastique de la fonction de dynamique du modèle, et, d'autre part, du fait que les autres agents sélectionnent de leur côté une action jointe, suivant leur politique actuelle. En d'autres termes, effectuer une action à un nœud ne va pas toujours nous amener vers le même nœud fils, et tous les nœuds fils ne seront pas nécessairement atteignables, peu importe l'action choisie.

Certains historiques d'observations peuvent donc ne jamais être atteints car la séquence correspondante d'observations peut ne jamais apparaître durant les différentes itérations de l'algorithme POS-UCT. Il n'est alors pas possible de leur assigner directement une action via la règle de l'équation 5.16. Ce n'est cependant pas un problème pour deux raisons.

1. Tout d'abord, l'algorithme (POS-)MCTS est conçu pour ne pas explorer et construire l'arbre de recherche dans son intégralité. Il s'agit de l'un des points forts de cet algorithme : l'arbre construit est asymétrique, et tend à ne pas explorer trop en profondeur les chemins non prometteurs, où les récompenses cumulées seront faibles. Cela nous permet de concevoir des politiques individuelles moins lourdes en mémoire et plus rapides à calculer.
2. Ensuite, même s'il se trouve qu'un ou plusieurs des historiques d'observations n'ont pas été visités durant l'algorithme, mais que, durant l'exécution réelle du processus (une fois l'étape de planification terminée) l'un des agents rencontre cet historique, alors on peut décider d'associer une action par défaut, aléatoirement définie, à cet historique. En pratique, les historiques rarement rencontrés durant plusieurs milliers d'itérations de l'algorithme ont tout aussi peu de chances d'être rencontrés durant l'exécution en-ligne. Leur influence sur le comportement global du système multi-agents est négligeable. Cela dépend évidement du problème étudié. Dans un problème où tous les historiques ont une (quasi-)équiprobabilité très faible d'apparaître, un algorithme tel que POS-UCT peut être en défaut.

5.6 Conclusion

Ce chapitre a introduit formellement le problème TARS de formation et de formation dynamique d'équipes sous incertitude. Il s'agit d'un problème qui n'a été – au mieux – qu'à peine traité par la communauté de la recherche en théorie de la décision. Afin de traiter les problèmes de planification multi-agents ouverts, nous avons proposé en première approche le modèle d'OPEN-DEC-POMDP, qui permet de modéliser les situations où les agents peuvent entrer et sortir de l'équipe opérationnelle durant le processus de résolution de la tâche confiée aux agents. Ce concept de flexibilité de l'équipe opérationnelle est une propriété qui se retrouve souvent dans les applications réelles, où robots militaires, drones de surveillance et autres types d'agents sont sujets à des conditions qui ne leur permettent pas, ou ne les encouragent pas, à rester actifs dans le système indéfiniment. Nous nous sommes concentrés sur une classe particulière de notre modèle, l'OPEN-DEC-POMDP à contrôle total, où les agents ne peuvent entrer et sortir du système que s'ils effectuent explicitement les actions associées. Dans cette sous-classe, un agent ne peut pas, par exemple, être mis hors-service par accident ou se retrouver à court de ressources et ainsi modifier contre son gré la composition de l'équipe opérationnelle.

Nous avons utilisé l'algorithme de recherche de meilleures réponses BRD pour calculer des politiques jointes séparables approximatives (non-optimales) en gardant fixe l'ensemble de toutes les politiques des agents sauf une, ce qui nous permet de considérer notre problème initial de calcul d'une politique jointe comme une séquence de résolutions de POMDPs simples. À chaque itération de l'algorithme BRD-POS-UCT, la meilleure réponse d'un agent est calculée via

l'algorithme MCTS, ici adapté pour pouvoir calculer hors-ligne des politiques dans un domaine partiellement observable. Nous avons opté pour MCTS car il s'agit d'un algorithme dont les performances récentes se sont révélées être particulièrement prometteuses, et qui jouit d'une popularité importante au sein de la communauté scientifique.

De nombreuses améliorations peuvent être apportées pour résoudre un OPEN-DEC-POMDP de façon plus efficace et trouver des politiques jointes optimales. Le fait qu'un OPEN-DEC-POMDP à contrôle total soit une instance de DEC-POMDP peut nous permettre d'appliquer directement le large éventail d'algorithmes de résolution de DEC-POMDPs existants. Plutôt que de s'orienter dans cette voie, le chapitre suivant se voudra plus orienté vers la théorie des jeux coopératifs. Nous y étudierons notamment ce qu'elle peut apporter aux problèmes de formation d'équipes sous incertitude, puis nous présenterons une nouvelle formulation du modèle d'OPEN-DEC-POMDP ainsi que plusieurs modifications de l'algorithme POS-UCT qui nous permettrons d'exploiter les spécificités de ce nouveau modèle.

Chapitre 6

Éléments de théorie des jeux pour le problème de formation dynamique d'équipes sous incertitude

Sommaire

6.1 Introduction 88
6.2 POMDP d'équipes 89
6.2.1 Définition du modèle 89
6.2.2 Historiques et politiques partiels 90
6.3 Propriétés structurelles 92
6.3.1 Concepts de base 92
6.3.2 Monotonie, suradditivité et convexité 95
6.3.3 Illustrations 97
6.3.4 Complexité algorithmique 98
6.4 Indices de pouvoir 99
6.4.1 Utilité différentielle 100
6.4.2 Le système de classement Elo 102
6.4.3 Matchs 103
6.4.4 Volatilité et stabilité 104
6.5 L'algorithme MELO 104
6.5.1 Recherche arborescente Monte-Carlo avec classement Elo 105
6.5.2 Matchs des meilleures équipes 109
6.5.3 Justification des classements Elo 109
6.5.4 Séparabilité 110
6.6 Conclusion 112

« Concentrez-vous sur le gain matériel. Qu'importe ce que votre adversaire vous donne, prenez, sauf si vous voyez une bonne raison de ne pas le faire. »

Bobby Fisher

6.1 Introduction

Le chapitre 5 a été notre première tentative de modélisation pour faire de la planification sous incertitude dans un système multi-agents ouvert. Le modèle que nous avons introduit, l'OPEN-DEC-POMDP, est un modèle générique qui permet de représenter des situations où les agents peuvent entrer et sortir du système à tout moment. Ce qui nous intéresse dans cette thèse, c'est le contrôle total que nous pouvons exercer sur les systèmes multi-agents ouverts. On veut pouvoir dire aux agents quoi faire en quelle circonstance. En particulier, on veut pouvoir permettre aux agents de rejoindre ou quitter l'équipe en place suivant l'évolution de la situation, et on ne veut pas que ces entrées et sorties soient dues à des causes externes au processus de décision. C'est pourquoi nous reformulons le modèle d'OPEN-DEC-POMDP à contrôle total et introduisons un nouveau modèle : le POMDP d'équipes (ou TEAM-POMDP).

Nous décidons également de sacrifier la notion de séparabilité des politiques jointes calculées afin de pouvoir mettre au point un algorithme de résolution qui puisse exploiter la structure des systèmes multi-agents ouverts. Cela signifie que l'on va considérer une exécution *centralisée* une fois l'étape de planification effectuée. En d'autres termes, dans ce chapitre, on conçoit un coach de football qui, après avoir préparé sa stratégie d'avant-match, donne les consignes à ses joueurs depuis le bord du terrain et décide de l'équipe alignée.

Ce chapitre étend les travaux présentés au chapitre 5. Il se découpe suivant quatre sections principales :

1. **POMDPs d'équipes** : dans la section 6.2, nous présentons le TEAM-POMDP, correspondant à une nouvelle formulation du modèle d'OPEN-DEC-POMDP à contrôle total. Nous y donnons également les outils pour faire de la planification optimale et pointons les différences existantes entre ce nouveau modèle et l'ancien.
2. **Propriétés structurelles** : dans la section 6.3 sont données quelques définitions et propriétés inspirées du domaine de la théorie des jeux coopératifs et suivies par certaines classes du modèle de TEAM-POMDP. Parmi ces définitions, les notions de monotonie, de sur-additivité et de convexité seront abordées. Enfin, nous donnons la complexité théorique de la résolution de notre modèle.
3. **Indices de pouvoir** : dans la section 6.4, nous introduisons le concept d'indice de pouvoir. Ces indices, utilisés dans le domaine de la théorie des jeux et des systèmes multi-agents, permettent d'évaluer l'importance des agents au sein de leur groupe. Nous montrons comment nous servir du système de classement Elo afin d'évaluer la valeur, l'utilité, d'une équipe d'agents dans le cadre du modèle de TEAM-POMDP.
4. **Nouvelle méthode de recherche arborescente de politiques** : dans la section 6.5, on se base sur l'algorithme de résolution POS-UCT étudié au chapitre précédent pour proposer une nouvelle méthode de planification pour les TEAM-POMDPs. Notre nouvel algorithme intègre et met à jour le classement Elo des équipes directement dans l'arbre de recherche et permet de calculer des politiques jointes, non séparables, en exploitant les propriétés structurelles des problèmes traités.

Le contenu de ce chapitre a donné lieu à plusieurs publications :

— Article publié lors de la Conférence Internationale sur les Outils en Intelligence Artificielle (*International Conference on Tools with Artificial Intelligence*, ICTAI), présenté en 2018 à Volos, Grèce [Cohen and Mouaddib, 2018a].

— Article publié lors des Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA), présenté en 2018 à Nancy, France [Cohen and Mouaddib, 2018b].

— Résumé étendu publié lors à la Conférence Internationale sur les Agents Autonomes et les Systèmes Multi-agents (*International Conference on Autonomous Agents and Multiagent Systems*, AAMAS), présenté en 2019 à Montréal, Canada [Cohen and Mouaddib, 2019].

6.2 POMDP d'équipes

Dans cette section, nous proposons une nouvelle formulation du modèle d'OPEN-DEC-POMDP à contrôle total.

6.2.1 Définition du modèle

Un **processus décisionnel de Markov partiellement observable d'équipes** (*Team-oriented Partially Observable Markov Decision Process*, **TEAM-POMDP**, aussi appelé **POMDP d'équipes**) est une généralisation du modèle de DEC-POMDP au cas où la population d'agents est amenée à être modifiée lors de l'exécution.

Définition 6.2.1 - *Processus décisionnel de Markov partiellement observable d'équipes* : Un TEAM-POMDP est un tuple

$$(\mathcal{N},\mathcal{C},\mathcal{S},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{O}},\boldsymbol{D},\boldsymbol{R},b^0,h),$$

où

— $\mathcal{N} = \{1,2,\ldots,n\}$ est l'ensemble fini des $n \in \mathbb{N}$ agents du système ;

— $\mathcal{C} \subseteq 2^{\mathcal{N}}$ est l'ensemble fini des équipes possibles basées sur la population $\mathcal{N}$; $2^{\mathcal{N}}$ désigne l'ensemble des parties de $\mathcal{N}$;

— $\mathcal{S}$ est l'ensemble fini des états du système ;

— $\boldsymbol{\mathcal{A}} = \{\mathcal{A}_C \mid C \in \mathcal{C}\}$, où $\mathcal{A}_C = \{\mathcal{A}_i \mid i \in C\}$ est l'ensemble des actions jointes de l'équipe $C \in \mathcal{C}$, et $\mathcal{A}_i$ est l'ensemble des actions individuelles de l'agent $i \in \mathcal{N}$;

— $\boldsymbol{\mathcal{O}} = \{\mathcal{O}_C \mid C \in \mathcal{C}\}$, où $\mathcal{O}_C = \{\mathcal{O}_i \mid i \in C\}$ est l'ensemble des observations jointes de l'équipe $C \in \mathcal{C}$, et $\mathcal{O}_i$ est l'ensemble des observations individuelles de l'agent $i \in \mathcal{N}$;

— $\boldsymbol{D} = \{D_C \mid C \in \mathcal{C}\}$, où $D_C : \mathcal{S} \times \mathcal{A}_C \times \mathcal{O}_C \times \mathcal{S} \to [0,1]$ est la fonction de dynamique de l'équipe C, telle que $D_C(s,a_C,o_C,s') = Pr(o_C,s' \mid a_C,s)$ est la probabilité que le système passe dans l'état $s' \in \mathcal{S}$ et que les agents de $C \in \mathcal{C}$ fassent l'observation jointe $o_C \in \mathcal{O}_C$ après avoir pris l'action jointe $a_C \in \mathcal{A}_C$ dans l'état $s \in \mathcal{S}$;

— $\boldsymbol{R} = \{R_C \mid C \in \mathcal{C}\}$, où $R_C : \mathcal{S} \times \mathcal{A}_C \times \mathcal{S} \to \mathbb{R}$ est la fonction de récompense de l'équipe $C \in \mathcal{C}$, telle que $R_C(s,a_C,s')$ est la récompense obtenue quand le système passe dans l'état s' après que les agents de C ont effectué l'action jointe $a_C \in \mathcal{A}$ dans l'état $s \in \mathcal{S}$;

— b^0 est l'état de croyance initial, une distribution de probabilité sur les états de $\mathcal{S}$;

— h est l'horizon de planification.

Un TEAM-POMDP se déroule principalement de la même façon qu'un DEC-POMDP. La figure 6.1 illustre une étape de décision dans un TEAM-POMDP. L'état initial du système à $t = 0$ est s^0, déterminé selon la distribution initiale b^0. À chaque étape de décision t, une (et une seule) équipe d'agents $C^t \in \mathcal{C}$ est dite **opérationnelle** dans le système. Pendant ce temps, les agents qui ne font pas partie de l'équipe opérationnelle sont dans la **réserve**, réserve constituée des agents $\mathcal{N} \backslash C^t$. L'équipe opérationnelle joue une action jointe $a^t_{C^t}$, fait transiter

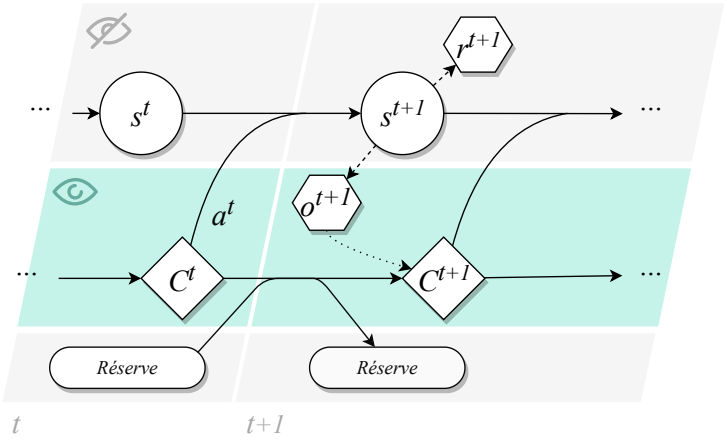


FIGURE 6.1 – Représentation de l'exécution du modèle TEAM-POMDP. Une transition d'une étape de décision t à l'étape de décision $t+1$ est représentée. Les agents de l'équipe C^t prennent l'action jointe a^t, générant une récompense r^{t+1} et percevant l'observation o^{t+1}. Le système passe de l'état s^t à l'état s^{t+1}, et la nouvelle équipe devient C^{t+1} lorsque certains agent quittent C^t pour rejoindre la *Réserve* et d'autres agents quittent la *Réserve* pour rejoindre la nouvelle équipe C^{t+1}. Les pictogrammes en forme d'œil indiquent ce qui est observable par les agents et ce qui ne l'est pas.

le système d'un état s^t à un état s^{t+1} et reçoit l'observation jointe $o^{t+1}_{C^t}$. À l'étape de décision $t+1$, une nouvelle équipe $C^{t+1} \in \mathcal{C}$ (potentiellement la même qu'à l'instant t), devient l'équipe opérationnelle. Cette nouvelle équipe va à son tour effectuer une action jointe et recevoir une observation jointe. Ce processus se déroule ainsi jusqu'à ce que l'horizon de planification $t = h-1$ soit atteint. Dans ce travail, on ne considère que des horizons finis.

Le modèle de TEAM-POMDP possède quelques caractéristiques communes évidentes avec le modèle de DEC-POMDP. Les principales différences proviennent de la nature multiple des différents éléments du tuple : plusieurs fonctions de dynamique, de récompense, et plusieurs ensembles d'actions et d'observations, un pour chaque équipe possible. Le modèle peut être vu comme une reformulation du modèle standard de DEC-POMDP, adapté à la formation dynamique d'équipe. Un TEAM-POMDP se ramène à un DEC-POMDP lorsque $|\mathcal{C}| = 1$, c'est-à-dire lorsqu'il ne peut exister qu'une seule équipe opérationnelle. De même, un TEAM-POMDP est également l'équivalent d'un OPEN-DEC-POMDP à contrôle total. Comme nous le verrons plus loin, cette observation permet d'affirmer que les TEAM-POMDPs ont une complexité au moins égale à celle des DEC-POMDPs.

Le modèle de TEAM-POMDP vient avec une difficulté supplémentaire par rapport aux DEC-POMDPs. Dans ces derniers, les agents reçoivent des observations à chaque instant t. En mettant à jour leur historique d'observations, ils peuvent utiliser leur politique individuelle pour décider quelle action choisir. Or, dans un TEAM-POMDP, les agents qui ne sont pas dans l'équipe opérationnelle ne perçoivent pas d'observations. C'est la raison pour laquelle nous introduisons les notions d'historiques partiels et de politiques partielles.

6.2.2 Historiques et politiques partiels

Définition 6.2.2 - *Historique d'observations partiel individuel* : Soit M un TEAM-POMDP. L'*historique d'observations partiel individuel* $\vec{o}_i^t$ pour un agent $i \in \mathcal{N}$ à l'instant $t < h$

est la suite des *observations partielles* reçues par l'agent jusqu'à l'instant t :

$$\vec{o}_i^t = (\dot{o}_i^1, \dot{o}_i^2, \ldots, \dot{o}_i^t)$$

où

$$\dot{o}_i^{t'} = \begin{cases} o_i^{t'} \in \mathcal{O}_i & \text{si } i \in C^{t'-1} \\ \textit{Rien} & \text{sinon} \end{cases}, \forall t' \in [1,t].$$

L'observation *Rien* désigne une observation manquante.

On peut noter un historique d'observations partiel individuel de façon compacte comme une séquence temporellement indexée d'observations individuelles. Par exemple,

$$(o_i^1, o_i^2, o_i^5) = (o_i^1, o_i^2, \textit{Rien}, \textit{Rien}, o_i^5)$$

correspond à l'historique d'observations partiel individuel d'un agent $i \in \mathcal{N}$ qui n'était pas dans l'équipe opérationnelle aux étapes de décision $t = 2$ et $t = 3$.

L'*historique d'observations partiel joint* à l'instant t est la collection des historiques d'observations partiels individuels des agents présents dans l'équipe opérationnelle $C^t \in \mathcal{C}$:

$$\vec{o}^t = (\vec{o}_i^t \mid i \in C^t).$$

On note $\vec{\mathcal{O}}_i$ l'ensemble des historiques d'observations partiels individuels de l'agent i et $\vec{\mathcal{O}}$ l'ensemble des historiques d'observations partiels joints. Remarquons qu'un historique d'observations partiel individuel $\vec{o}_i^t$ se ramène à un historique d'observations individuel classique, complètement spécifié, lorsque toutes les observations sont présentes, c'est-à-dire si l'agent i n'a jamais quitté l'équipe opérationnelle. Par souci de concision, nous allons simplement parler d'historique individuel pour désigner l'historique d'observations partiel individuel d'un agent i (et de façon similaire pour les historiques joints).

Une *politique partielle individuelle* π_i pour un agent $i \in \mathcal{N}$ est une fonction

$$\pi_i : \vec{\mathcal{O}}_i \to \mathcal{A}_i \cup \{\textit{Quitter}\}$$

où *Quitter* désigne une action n'ayant aucun effet sur le système. Lorsque l'agent effectue cette action, il se retire de l'équipe opérationnelle. En effectuant une action autre que l'action *Quitter*, alors il rejoint l'équipe opérationnelle et effectue son action. On note Π_i l'ensemble des politiques partielles individuelles de l'agent i. Une *politique jointe* π est le tuple correspondant à l'ensemble des politiques partielles individuelles des agents de $\mathcal{N}$:

$$\pi = (\pi_1, \pi_2, \ldots, \pi_n),$$

avec $\pi_i \in \Pi_i$ pour tout agent $i \in \mathcal{N}$. On note Π l'ensemble des politiques jointes.

Résoudre un TEAM-POMDP consiste à trouver une politique jointe séparable optimale, notée π^*, qui maximise la somme espérée des récompenses cumulées sur l'horizon de planification h, en commençant à l'instant $t = 0$ depuis l'état de croyance initial b^0 :

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{h-1} R^t \mid \pi, b^0\right], \tag{6.1}$$

où $\mathbb{E}[\cdot]$ désigne l'opérateur de l'espérance mathématique et où les $(R^t)_{t \in \{0,1,\ldots,h-1\}}$ correspondent aux récompenses obtenues en suivant la politique jointe π depuis l'état de croyance initial b^0.

Notations Avant de continuer, introduisons quelques notations intuitives et pratiques. Soit M un TEAM-POMDP, $C \in \mathcal{C}$ une équipe d'agents de M et $i \in C$ un agent. On note $-C = \mathcal{N} \backslash C$ l'ensemble de la population d'agents privée des agents de C. En particulier, on note $-i = \mathcal{N} \backslash \{i\}$ l'ensemble de tous les agents sauf l'agent i.

Concernant les politiques jointes partielles, on note

$$\pi_C = (\pi_i \mid \forall i \in C)$$

la politique jointe restreinte aux agents de l'équipe C. Pour chaque historique d'observations partiel joint $\vec{o} \in \vec{\mathcal{O}}$, l'action jointe $\pi_C(\vec{o}) = (\pi_i(\vec{o}_i) \mid i \in C)$ correspond au tuple des actions individuelles des agents de C. Notons que si C est l'équipe opérationnelle à l'étape de décision actuelle, alors, pour tout agent $i \in C$, $\pi_i(\vec{o}_i) \neq$ *Quitter*.

Concernant les historiques d'observations partiels joints, on note

$$\vec{o}_C = (\vec{o}_i \mid \forall i \in C)$$

l'historique joint restreint aux agents de C. Pour chaque historique d'observations partiel joint $\vec{o}_C \in \vec{\mathcal{O}}_C$, l'action jointe $\pi(\vec{o}_C) = (\pi_i(\vec{o}_i) \mid i \in C)$ correspond également au tuple des actions individuelles des agents de C.

On remarquera que, pour toute équipe $C \in \mathcal{C}$ et pour tout historique d'observations partiel joint $\vec{o}$, on a $\pi_C(\vec{o}) = \pi_C(\vec{o}_C) = \pi(\vec{o}_C)$.

6.3 Propriétés structurelles

Nous présentons dans cette section quelques propriétés structurelles respectées par certaines instances de notre modèle. Ces définitions et propriétés sont habituellement rencontrées dans le domaine de la théorie des jeux coopératifs, où les agents (en fait, des joueurs), doivent former des coalitions afin de maximiser leurs gains personnels. Les définitions présentées ici sont des extensions des définitions données dans l'état de l'art concernant la formation de coalitions (section 4.3) aux cas multi-agents et stochastique. Exploiter adéquatement ces propriétés peut permettre de mettre en exergue certaines sous-classes de TEAM-POMDPs afin de pouvoir les résoudre plus efficacement.

Par souci de concision, et sans perte de généralité, on va supposer que les fonctions de récompense R_C sont définies sur l'ensemble des états et des actions jointes :

$$R_C : \mathcal{S} \times \mathcal{A}_C \to \mathbb{R},\ \forall C \in \mathcal{C}.$$

6.3.1 Concepts de base

Définition 6.3.1 - *Agent nécessaire* : Soit M un TEAM-POMDP. Un agent $i \in \mathcal{N}$ est *nécessaire* à un agent $j \in \mathcal{N}$ si, pour toute équipe $C \in \mathcal{C}$ telle que $i \notin C$ et $j \in C$, pour tout état $s \in \mathcal{S}$ et pour toute action jointe $a_C \in \mathcal{A}_C$, on a :

$$\begin{cases} R_C(s,a_C) = R_{C\backslash\{j\}}(s,a_{C\backslash\{j\}}), \\ R_{C\cup\{i\}}(s,a_{C\cup\{i\}}) \geqslant R_{C\cup\{i\}\backslash\{j\}}(s,a_{C\cup\{i\}\backslash\{j\}}) \end{cases} \tag{6.2}$$

Si un agent i est nécessaire à un agent j, alors il n'y a aucun intérêt à utiliser l'agent j sans l'agent i : une équipe C qui ne contient pas i fait aussi bien avec ou sans l'agent j. L'utilité d'un agent est conditionnée à la présence *actuelle* d'un second agent dans l'équipe opérationnelle, comme

s'il existait une sorte de symbiose entre les deux agents. Il s'agit d'une condition assez forte puisqu'elle doit être vraie pour tous les états et toutes les actions jointes possibles. Par exemple, le *Problème de la poursuite* (*Pursuit domain*) [Benda et al., 1986], illustré sur la figure 6.2, met en lumière la nécessité de certains agents. Dans ce problème (également étudié dans le domaine du travail d'équipe ad-hoc [Barrett et al., 2012]), une meute de Prédateurs doit coopérer pour encercler et attraper une Proie. Les Prédateurs (nos agents) et la Proie se déplacent sur une grille toroïdale. À chaque étape de décision, les Prédateurs et la Proie choisissent de se déplacer d'une case, tous simultanément. Si possible, la Proie s'éloigne des Prédateurs. Les Prédateurs observent l'état de chacune des cases adjacentes à leur position : soit la case est vide, soit elle contient un Prédateur, soit elle contient la Proie. Les Prédateurs gagnent si l'un d'entre eux se retrouve sur la même case que la Proie, ou si la Proie ne peut plus se déplacer sans se retrouver sur une case occupée par un Prédateur. Dans cet exemple, les agents sont nécessaires les uns aux autres car, étant donné que la Proie observe les cases adjacentes à sa position, un Prédateur seul ne pourra jamais l'attraper et ainsi obtenir la récompense.

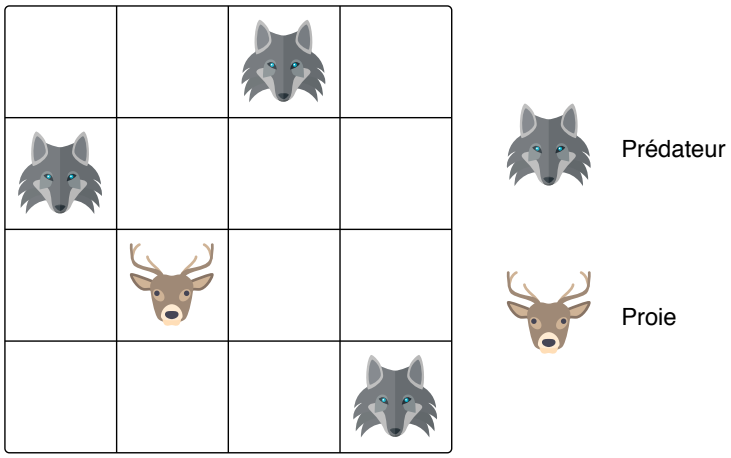


FIGURE 6.2 – Illustration du *Problème de la poursuite*. Suivant la taille de la grille, le nombre de Prédateurs nécessaire pour capturer la Proie peut varier.

Dans d'autres cas, il arrive que l'utilité d'un agent soit conditionnée à la présence *passée* d'un second agent dans l'équipe opérationnelle. Imaginons la situation où une catastrophe naturelle de grande ampleur a provoqué un tremblement de terre ayant causé de nombreux ravages dans une ville développée. Les décombres empêchent les pompiers d'atteindre l'intérieur des bâtiments en ruines afin d'y secourir les possibles survivants. Afin qu'ils puissent accomplir leur mission, il faut donc tout d'abord procéder au dégagement des gravats, par exemple en faisant appel à des engins de déblayage, bulldozers, excavatrices, etc. Tant que la voie n'aura pas été dégagée par ces agents, l'utilité des pompiers sera nulle puisqu'ils ne pourront accomplir aucune action de sauvetage.

Définition 6.3.2 - *TEAM-POMDP positif* : Soit M un TEAM-POMDP. M est *positif* si, pour toute équipe $C \in \mathcal{C}$, pour tout état $s \in \mathcal{S}$ et pour toute action jointe $a_C \in \mathcal{A}_C$, la fonction de récompense $R_C \in \boldsymbol{R}$ est à valeurs réelles positives ou nulles :

$$R_C(s,a_C) \in \mathbb{R}_+.$$

Un cas particulier de TEAM-POMDP positif se présente lorsque les récompenses que peuvent obtenir les agents sont soit 0, soit 1.

Définition 6.3.3 - *TEAM-POMDP simple* : Soit M un TEAM-POMDP. M est *simple* si, pour toute équipe $C \in \mathcal{C}$, pour tout état $s \in \mathcal{S}$ et pour toute action jointe $a_C \in \mathcal{A}_C$, on a :

$$R_C(s,a_C) \in \{0,1\}. \tag{6.3}$$

L'action jointe d'une équipe dans un TEAM-POMDP simple est donc soit *perdante*, soit *gagnante*. Par extension, on parle d'*équipe gagnante* ou *perdante*. Une équipe $C \in \mathcal{C}$ est perdante si, pour tout état $s \in \mathcal{S}$ et pour toute action jointe $a_C \in \mathcal{A}_C$, on a :

$$R_C(s,a_C) = 0. \tag{6.4}$$

Sinon, C est gagnante. Remarquons qu'une équipe peut être gagnante ou perdante sans que le TEAM-POMDP sous-jacent soit simple ou positif.

Le *Problème de la poursuite* de la figure 6.2 peut être modélisé comme un TEAM-POMDP simple : les agents ne sont récompensés que lorsque l'un des Prédateurs se déplace sur une case occupée par la Proie. Cette récompense est donc binaire. De plus, si le nombre de Prédateurs est trop faible, la Proie pourra toujours trouver un moyen de leur échapper. Certaines équipes, composées de trop peu d'agents, sont donc perdantes.

Il reste important de noter qu'une équipe perdante n'est pas nécessairement une *mauvaise* équipe, dans le sens commun où on l'entend, ni qu'un agent qui a besoin d'un autre agent (nécessaire) est *mauvais*. Les définitions données jusqu'à présent, ainsi que les définitions qui seront données dans la suite de cette section, concernent les *fonctions de récompense* du TEAM-POMDP étudié, et non pas la *fonction de valeur* d'une éventuelle politique optimale pour ce TEAM-POMDP. Une équipe dans un TEAM-POMDP simple peut être perdante, il n'empêche qu'elle sera peut-être la seule à pouvoir faire transiter le système dans des états futurs plus favorables (certes pas favorables à elle-même, étant donné qu'elle est perdante, mais favorables à d'autres futures équipes).

Nous faisons donc la distinction entre une *équipe perdante* (qui n'a pas d'utilité immédiate à une étape de décision) et une *équipe dominée* (qui sera toujours moins utile, pour le long terme, qu'une autre équipe, si l'équipe dominée était utilisée à la place cette autre équipe à une étape de décision).

Définition 6.3.4 - *Équipe dominée* : Soit M un TEAM-POMDP. Une équipe $C \in \mathcal{C}$ est *dominée* si, pour toute politique jointe $\pi \in \Pi$, il existe une équipe $C' \neq C \in \mathcal{C}$ telle que :

$$\mathbb{E}\left[\sum_{t=0}^{h-1} R^t \mid \pi,b^0,C^t = C\right] \leqslant \mathbb{E}\left[\sum_{t=0}^{h-1} R^t \mid \pi,b^0,C^t = C'\right]. \tag{6.5}$$

Autrement dit, une équipe est dominée si l'espérance des récompenses que l'on peut obtenir en utilisant cette équipe à une quelconque étape de décision t est inférieure à l'espérance des récompenses que l'on pourrait obtenir en utilisant une autre équipe à sa place. Malheureusement, s'il s'avère aisé de tester si un TEAM-POMDP est positif ou simple, ou si une équipe est gagnante ou perdante, déterminer si une équipe est dominée est une étape plus complexe puisque, selon la définition 6.3.4, cela nécessiterait de tester l'ensemble des politiques partielles jointes.

Nous examinerons plus tard (dans la section 6.4) comment évaluer l'impact à long terme d'une équipe. Pour l'heure, nous continuons avec quelques définitions à propos des TEAM-POMDPs.

6.3.2 Monotonie, suradditivité et convexité

Définition 6.3.5 - *Monotonie* : Un TEAM-POMDP M est *monotone* si ses fonctions de récompense $R_C \in \boldsymbol{R}$ sont monotones, c'est-à-dire, pour toutes équipes $C,C' \in \mathcal{C}$ telles que $C \subseteq C'$, pour tout état $s \in \mathcal{S}$ et pour toutes actions jointes $a_C \in \mathcal{A}_C, a_{C'} \in \mathcal{A}_{C'}$ telles que $a_C \subseteq a_{C'}$, on a :

$$R_C(s,a_C) \leqslant R_{C'}(s,a_{C'}). \tag{6.6}$$

Si un TEAM-POMDP est monotone, cela signifie que les agents n'ont pas d'impact négatif à faire partie de l'équipe opérationnelle. Si l'on ajoute des agents à une équipe opérationnelle, alors peu importe l'action jointe choisie par ces agents, l'effet engendré ne pourra pas être négatif.

On peut émettre quelques réserves concernant la pertinence d'une telle définition. Existent-ils seulement des cas où un TEAM-POMDP peut être monotone ? Imaginons l'une des situations souvent évoquée dans cette thèse : le cas où un feu de forêt se propage et où une équipe de pompiers doit intervenir pour le contenir le plus efficacement possible. Une équipe $C = \{1,2,3\}$ de trois pompiers est envoyée sur place. La situation est hors de contrôle, aussi un quatrième pompier, 4, décide de rejoindre l'équipe sur place. Ce pompier ne va pas agir à l'encontre de ses coéquipiers, aussi serait-on tenté de penser que la situation considérée est monotone : pour tout état $s \in \mathcal{S}$ et toutes actions $a_C \in \mathcal{A}_C$ et $a_4 \in \mathcal{A}_4$, on devrait avoir : $R_C(s,a_C) \leqslant R_{C\cup\{4\}}(s,(a_C,a_4))$.

Or, cela n'est pas évident dans le contexte d'un système **dynamique**, où l'évolution de l'état du système d'une étape de décision à l'autre n'est pas uniquement due aux actions des agents dans le système. Par exemple, le feu peut subitement s'aggraver suite à la levée du vent, et ce même si davantage de pompiers ont été mobilisés. Bien souvent, la propriété de monotonie, comme les autres propriétés présentées ci-après, n'est vérifiée que pour certaines instances de systèmes **statiques** (voir la section 2.1.3 pour un bref rappel sur la différence entre systèmes dynamiques et statiques). Heureusement, ces systèmes ne sont ni les plus rares, ni les moins intéressants à étudier, ni les moins adaptés à la formation dynamique d'équipes. Le *Problème de la poursuite* est une instance de TEAM-POMDP statique : comme la Proie fuit les Prédateurs, si ceux-ci n'agissent pas pour la capturer, ils n'obtiendront jamais de récompenses. De plus, ajouter un Prédateur ne sera jamais contre-productif, quels que soient l'état du système et l'action choisie.

Définition 6.3.6 - *Sur-additivité* : Un TEAM-POMDP M est *sur-additif* si ses fonctions de récompense $R_C \in \boldsymbol{R}$ sont sur-additives, c'est-à-dire, pour toutes équipes disjointes $C,C' \in \mathcal{C}$ telles que $C \cap C' = \varnothing$ et $C \cup C' \in \mathcal{C}$, pour tout état $s \in \mathcal{S}$ et pour toutes actions jointes $a_C \in \mathcal{A}_C, a_{C'} \in \mathcal{A}_{C'}$, on a :

$$R_C(s,a_C) + R_{C'}(s,a_{C'}) \leqslant R_{C\cup C'}(s,(a_C,a_{C'})). \tag{6.7}$$

M est *additif* si

$$R_C(s,a_C) + R_{C'}(s,a_{C'}) = R_{C\cup C'}(s,(a_C,a_{C'})). \tag{6.8}$$

La définition de sur-additivité spécifie que deux équipes disjointes peuvent être au moins meilleures en se regroupant en une seule équipe. Il n'y a pas d'intérêt à utiliser l'une ou l'autre : il faut utiliser la combinaison des deux.

Proposition 6.3.1 : Soit M un TEAM-POMDP positif. Si M est sur-additif, alors M est monotone.

Démonstration : Soient $C,C' \in \mathcal{C}$ deux équipes telles que $C \subseteq C'$, $s \in S$ un état et $a_C \in \mathcal{A}_C, a_{C'} \in \mathcal{A}_{C'}$ deux actions jointes telles que $a_C \subseteq a_{C'}$.

Comme $C \subseteq C'$, on peut réécrire C' sous la forme $C' = C \cup (C' \backslash C)$. On a alors :

$$\begin{aligned} R_{C'}(s,a_{C'}) &= R_{C\cup(C'\backslash C)}(s,a_{C\cup(C'\backslash C)}) \\ &\geqslant R_C(s,a_C) + R_{C'\backslash C}(s,a_{C'\backslash C}) \\ &\geqslant R_C(s,a_C) \end{aligned} \tag{6.9}$$

Le passage de la première ligne à la deuxième ligne se fait en remarquant que C et $C' \backslash C$ sont deux équipes disjointes, ce que nous permet d'appliquer la définition 6.3.6 de sur-additivité. Le passage de la deuxième ligne à la troisième ligne est immédiat dans le cas où M est positif. ▫

Définition 6.3.7 - *Convexité* : Un TEAM-POMDP M est *convexe* si, pour toutes équipes $C \subseteq C' \in \mathcal{C}$, pour tout état $s \in \mathcal{S}$, pour toutes actions jointes $a_C \in \mathcal{A}_C, a_{C'} \in \mathcal{A}_{C'}$ telles que $a_C \subseteq a_{C'}$, et pour tout agent $i \in \mathcal{N}$, $i \notin C'$, jouant l'action individuelle $a_i \in \mathcal{A}_i$ et tel que $C \cup \{i\}, C' \cup \{i\} \in \mathcal{C}$, on a :

$$R_{C\cup\{i\}}(s,(a_C,a_i)) - R_C(s,a_C) \leqslant R_{C'\cup\{i\}}(s,(a_{C'},a_i)) - R_{C'}(s,a_{C'}). \tag{6.10}$$

La propriété de convexité peut être écrite de façon équivalente :

$$R_C(s,a_C) + R_{C'}(s,a_{C'}) \leqslant R_{C\cup C'}(s,(a_C,a_{C'})) + R_{C\cap C'}(s,(a_C,a_{C'})). \tag{6.11}$$

Cette transformation se fait de la façon suivante. Soient $C \subseteq C' \in \mathcal{C}$ deux équipes et $i \in \mathcal{N}$ un agent tel que $i \notin C'$. Considérons les deux équipes $C \cup \{i\}$ et C'. Selon la propriété de convexité 6.11, en remplaçant C par $C \cup \{i\}$ dans l'équation, on a :

$$R_{C\cup\{i\}}(s,a_{C\cup\{i\}}) + R_{C'}(s,a_{C'}) \leqslant R_{C\cup\{i\}\cup C'}(s,a_{C\cup\{i\}\cup C'}) + R_{C\cup\{i\}\cap C'}(s,a_{C\cup\{i\}\cap C'}). \tag{6.12}$$

Comme $C \subseteq C'$ et $i \notin C'$, on a $C \cup \{i\} \cup C' = C' \cup \{i\}$, et $C \cup \{i\} \cap C' = C$. Cela nous permet de simplifier l'équation 6.12 :

$$R_{C\cup\{i\}}(s,a_{C\cup\{i\}}) + R_{C'}(s,a_{C'}) \leqslant R_{C'\cup\{i\}}(s,a_{C'\cup\{i\}}) + R_C(s,a_C), \tag{6.13}$$

ce qui nous donne, par soustractions de chaque côté de l'inégalité :

$$R_{C\cup\{i\}}(s,a_{C\cup\{i\}}) - R_C(s,a_C) \leqslant R_{C'\cup\{i\}}(s,a_{C'\cup\{i\}}) - R_{C'}(s,a_{C'}). \tag{6.14}$$

Comme $a_{C\cup\{i\}} = (a_C,a_i)$ et $a_{C'\cup\{i\}} = (a'_C,a_i)$, on retrouve bien l'équation 6.10.

Proposition 6.3.2 : Soit M un TEAM-POMDP positif. Si M est convexe, alors il est sur-additif.

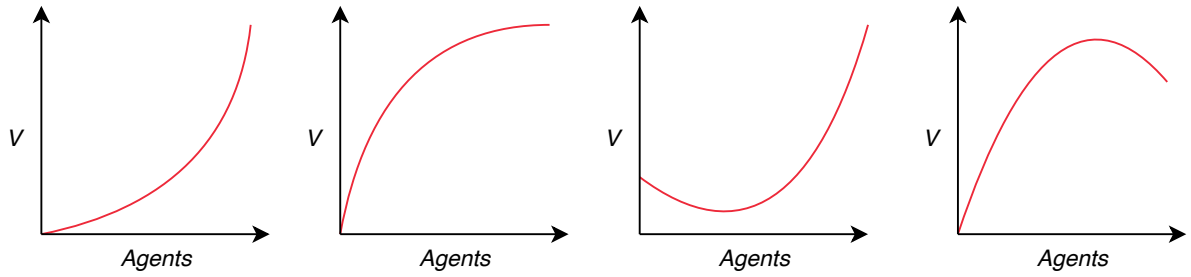
Ce résultat découle trivialement de la propriété de convexité lorsque deux équipes sont disjointes.

6.3.3 Illustrations

La figure 6.3 illustre quelques types de fonctions de récompense que l'on peut croiser lorsque l'on travaille avec un TEAM-POMDP. Faisons tout d'abord deux remarques :

— Bien que les courbes présentées soient continues, les fonctions de récompense d'un TEAM-POMDP sont discrètes sur l'espace des agents, des états et des actions. Il ne s'agit que d'illustrations permettant de visualiser des tendances.

— L'axe des abscisses augmente vers la droite avec le nombre d'agents présents dans l'équipe opérationnelle. Cette représentation n'est compatible qu'avec des TEAM-POMDPs **homogènes**, où tous les agents sont identiques (mêmes ensembles d'actions et d'observations). La figure 6.3 n'est là, encore une fois, que pour mettre en relief certaines tendances.



(a) Convexité (b) Sous-modularité (c) Récompenses non-monotones

FIGURE 6.3 – Exemples de différents types de fonctions de récompense pour un TEAM-POMDP homogène.

La figure 6.3a représente le cas de la convexité, où l'ajout d'agents dans l'équipe opérationnelle va significativement augmenter les récompenses que cette équipe va générer, et ce de plus en plus au fur et à mesure que l'on ajoute des agents.

La figure 6.3b quant à elle illustre le cas inverse de la convexité. C'est ce que l'on appelle la **sous-modularité**. La sous-modularité est liée à la *loi des rendements décroissants* : dans le cadre d'un système multi-agents, il s'agit de l'idée selon laquelle ajouter de plus en plus d'agents à l'équipe opérationnelle sera de moins en moins profitable, sans pour autant que cela ne devienne préjudiciable. L'exemple du feu de forêt est sous-modulaire : à partir d'un certain point, ajouter de nouveaux pompiers n'apportera plus aucune valeur si l'équipe en place est déjà assez grande. Ce phénomène de saturation est commun lorsque l'on considère des systèmes multi-agents coopératifs. Les systèmes sous-modulaires suivent une logique particulièrement intéressante : il est possible de se servir de beaucoup moins d'agents qu'il y en a de disponible pour obtenir des récompenses presque aussi bonnes. Dans un domaine de recherche comme celui de la planification multi-agents, où la complexité de résolution est directement liée au nombre d'agents pour lequel il faut planifier, il s'agit donc d'une information capitale qui doit être d'exploitée.

Enfin, les figures 6.3c illustrent d'autres cas, non présentés formellement ici. À gauche, on observe tout d'abord un phénomène de baisse de la valeur de l'équipe, puis une remontée rapide. Imaginons des agents mobiles autonomes qui doivent procéder à la récolte d'un champ. Initialement, le coût d'un agent est trop élevé pour qu'il soit rentable au producteur. Ajouter des agents pour récolter ne lui permet pas de de rentabiliser ses dépenses immédiatement, mais lui coûte toujours plus cher en matériel. Au bout d'un moment, le nombre d'agents est suffisamment élevé pour que les revenus excèdent finalement les dépenses.

Sur la figure 6.3c à droite, le comportement est inverse : ajouter des agents à l'équipe est très vite profitable, puis de moins en moins, puis finalement plus du tout lorsque le nombre d'agents est trop élevé. On peut illustrer cela avec l'exemple de drones de rangement dans l'entrepôt d'une entreprise. Ces drones sont chargés de saisir les colis et de les transporter d'une zone à une autre. L'entrepôt est grand et peut contenir jusqu'à un grand nombre d'agents. Toutefois, à partir d'une certaine limite de saturation, les risques de collisions entre drones (et donc de frais conséquents pour l'entreprise) deviennent trop élevés par rapport à l'efficacité du service rendu.

6.3.4 Complexité algorithmique

Nous en venons maintenant aux résultats théoriques de notre travail. On commence par introduire l'*opérateur de restriction* d'un TEAM-POMDP à un DEC-POMDP, opérateur qui nous servira à établir notre résultat de complexité.

Définition 6.3.8 - *Restriction* : Soit M un TEAM-POMDP. Le DEC-POMDP $M|_C$ est la *restriction de* M aux agents de l'équipe $C \in \mathcal{C}$, et est défini par :

$$M|_C = (C,\mathcal{S},\mathcal{A}_C,\mathcal{O}_C,D_C R_C,b^0,h). \tag{6.15}$$

Cette définition permet simplement de formaliser la réduction d'un TEAM-POMDP à un DEC-POMDP lorsque l'on ne considère qu'une seule équipe opérationnelle possible (c'est-à-dire, pour tout $t < h$, $C^t = C$). En somme, il est possible de voir un TEAM-POMDP M comme une collection de DEC-POMDPs $\{M|_C \mid C \in \mathcal{C}\}$ qui, lors de l'exécution, alternent les uns avec les autres au gré des décisions transmises par la politique jointe.

Lemme 6.3.1 : Soit M un TEAM-POMDP. Si M est sur-additif, alors

$$\pi^*_M = \pi^*_{M|_{\mathcal{N}}}. \tag{6.16}$$

Ce résultat indique que, pour résoudre de façon optimale un TEAM-POMDP sur-additif M, il est suffisant de trouver la politique optimale du DEC-POMDP défini par la restriction de M à la grande équipe $\mathcal{N}$ de tous les agents. C'est un résultat qui découle de la définition 6.3.6 de sur-additivité, et qui dit que la grande équipe $\mathcal{N}$ de tous les agents, si elle suit une politique optimale, sera toujours au moins aussi performante que n'importe quelle autre équipe.

Proposition 6.3.3 : Soit M un TEAM-POMDP et $K \in \mathbb{Z}$ un nombre entier relatif. Trouver une politique jointe pour M générant un gain d'au moins K est un problème NEXP-complet.

Démonstration : Il nous faut montrer d'une part qu'il s'agit d'un problème appartenant à la classe NEXP, et montrer d'autre part qu'il s'agit d'un problème NEXP-difficile. Nous nous servons du fait que résoudre de façon optimale un DEC-POMDP (avec deux agents ou plus) est un problème NEXP-complet [Bernstein et al., 2000].

D'une part, il est possible d'encoder un TEAM-POMDP dans un DEC-POMDP de taille polynomiale (par rapport à la taille du TEAM-POMDP). Considérons un TEAM-POMDP $\Phi = (\mathcal{N},\mathcal{C},\mathcal{S},\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{O}},\boldsymbol{D},\boldsymbol{R},b^0,h)$. Le DEC-POMDP $\Phi' = (\mathcal{N},\mathcal{S}',\mathcal{A}',\mathcal{O}',D',R',b^0,h)$ permet d'encoder Φ en taille polynomialement plus grande que la taille de Φ. L'espace d'état de Φ' est $\mathcal{S}' = \mathcal{S} \times \mathcal{C}$. Les actions $\mathcal{A}'_i \in \mathcal{A}'$ d'un agent $i \in \mathcal{N}$ de Φ' sont encodées dans l'ensemble $\mathcal{A}'_i = \mathcal{A}_i \cup \{\textit{Quitter}\}$

et ses observations $\mathcal{O}'_i \in \mathcal{O}'$ dans l'ensemble $\mathcal{O}'_i = \mathcal{O}_i \cup \{Rien\}$. Il est ensuite possible de définir la fonction de dynamique $D' : \mathcal{S}' \times \mathcal{A}' \times \mathcal{O}' \times \mathcal{S}' \to [0,1]$ par

$$D'((s,C),a,o,(s',C')) = D_C(s,a_C,o_{C'},s')$$

et la fonction de récompense $R' : \mathcal{S}' \times \mathcal{A}' \times \mathcal{S}'$ par

$$R'((s,C),a,(s',C')) = R_C(s,a_C,s').$$

Puisqu'il est possible de représenter le TEAM-POMDP Φ dans le DEC-POMDP Φ' dont la taille est polynomiale par rapport à la taille de Φ, et comme résoudre un DEC-POMDP est un problème dans la classe de complexité NEXP, il suit que résoudre un TEAM-POMDP est également un problème dans la classe NEXP.

D'autre part, comme nous l'avons vu, si un TEAM-POMDP est sur-additif, alors il suffit de résoudre le DEC-POMDP issu de la restriction du TEAM-POMDP à la grande équipe $\mathcal{N}$. Comme un TEAM-POMDP n'est pas nécessairement monotone, et comme la restriction est un opérateur simplifiant (puisqu'il permet d'omettre une partie des agents, donc des actions et des observations), alors il suit que résoudre un TEAM-POMDP est au moins aussi dur que résoudre un DEC-POMDP. Résoudre un TEAM-POMDP est donc un problème NEXP-difficile et, puisqu'il est dans NEXP, également un problème NEXP-complet. ▫

6.4 Indices de pouvoir

Lorsque l'on considère des équipes d'agents, il est clair que, selon la situation de la tâche à résoudre, certaines d'entre elles se révéleront être plus efficaces que d'autres. Cela est dû au fait qu'un agent $i \in \mathcal{N}$ possède son propre ensemble d'actions individuelles $\mathcal{A}_i$, et que deux agents distincts peuvent donc avoir des ensembles disjoints d'actions. Dans le cas le plus extrême, tous les agents peuvent même être différents et chacun peut apporter sa contribution d'une façon ou d'une autre à un certain moment. Le même raisonnement peut s'appliquer à deux équipes d'agents, chacune constituée d'un ensemble distinct d'agents : l'une peut être adéquate dans une situation tandis que l'autre peut être totalement inefficace. Notons également que, même si tous les agents sont homogènes et possèdent des ensembles d'actions similaires, faire appel à la plus grande équipe opérationnelle possible ne sera pas toujours la meilleure solution. Certains agents peuvent interférer les uns avec les autres, induire des coûts de déploiement ou d'exploitation, etc.

Les **indices de pouvoir** sont des mesures de l'influence des agents (ou, dans le cas de la théorie des jeux, des joueurs). Plus un agent a un indice de pouvoir élevé, plus son influence sera importante ou positive pour le groupe. Ces indices sont souvent utiles pour révéler une distribution du pouvoir entre les agents qui n'est pas forcément évidente au premier abord. Par exemple, l'indice de Shapley-Shubick [Shapley and Shubik, 1954] et l'indice de Banzhaf [Banzhaf III, 1964] sont deux moyens de mesurer le pouvoir des joueurs dans un jeu de vote.

Ce n'est pas tant l'impact d'un agent que l'impact d'une équipe d'agents qui nous intéresse. Les agents réussissent ou échouent en équipe. De plus, et nous avons déjà abordé ce point dans la section 6.3.1, déterminer l'impact immédiat d'une équipe d'agents ne permet pas d'évaluer son importance à long terme. Peut-être que seuls certains agents, très coûteux mais dotés d'actions uniques, peuvent permettre d'atteindre, plus tard dans le processus de décision, des états du système où la récompense est suffisamment grande pour compenser le coût initialement induit par l'utilisation de ces agents. Il faut donc pouvoir estimer l'indice de pouvoir d'un agent ou

d'une équipe d'agents pour déterminer si il ou elle peut se révéler être utile pour le long terme. La manière la plus simple de déterminer l'utilité à long terme d'une équipe d'agents à un certain instant est de calculer la différence entre les récompenses espérées que l'on peut encore recevoir avec et sans ces agents. C'est ce que l'on appelle l'*utilité différentielle*.

6.4.1 Utilité différentielle

Considérons un TEAM-POMDP et π une politique jointe. À un certain historique d'observations partiel joint $\vec{o}^t$ dans un état s^t, on peut calculer la valeur de cette politique à cette paire historique-état via les équations 6.1 et 3.31. Il s'agit de la valeur de la récompense immédiate que l'équipe opérationnelle va recevoir à cet historique et cet état, plus la valeur estimée des récompenses futures que les prochaines équipes opérationnelles vont cumuler en suivant la politique π. À l'étape de décision t, la politique jointe π prescrit une action jointe $\pi_{C^t}(\vec{o}^t)$ à l'équipe opérationnelle $C^t \in \mathcal{C}$. Pendant ce temps, le reste des agents de la réserve (l'ensemble $\mathcal{N} \backslash C^t$) prend l'action *Quitter*.

Il est alors possible d'obtenir la valeur apportée par un sous-ensemble d'agents $C \in C^t$ en considérant ce que la récompense cumulée deviendrait si les agents de cette sous-équipe C ne faisaient plus partie de l'équipe opérationnelle C^t à l'étape de décision courante. En d'autres termes, si l'action jointe $\pi_{C^t \backslash C}(\vec{o}^t)$ était prise plutôt que l'action jointe $\pi_{C^t}(\vec{o}^t)$ – c'est-à-dire, si, pour tout agent $i \in C$,

$$\pi_i(\vec{o}_i^t) = \textit{Quitter}. \tag{6.17}$$

Cette valeur est appelée l'**utilité différentielle** (*difference utility*) [Wolpert et al., 1999, Agogino and Tumer, 2004, Devlin et al., 2014]. Dans le cadre de la théorie de jeux, on parle également de **regret** [Bell, 1982, Loomes and Sugden, 1982, Fishburn, 2013].

Définition 6.4.1 - *Utilité différentielle* : Soit M un TEAM-POMDP, π une politique jointe et $C \in \mathcal{C}$ une équipe. L'*utilité différentielle* des agents de C dans l'état $s^t \in \mathcal{S}$ à l'historique $\vec{o}^t \in \vec{\mathcal{O}}$ est

$$\lambda_C^\pi(s^t, \vec{o}^t) = V^\pi(s^t, \vec{o}^t) - V^\pi(s^t, \vec{o}^t_{-C}). \tag{6.18}$$

$V^\pi(s^t, \vec{o}^t)$ est la valeur de la politique jointe telle que décrite par l'équation 3.31. $V^\pi(s^t, \vec{o}^t_{-C})$ est la valeur de cette même politique, excepté que l'on considère que les agents de C ne font pas partie de l'équipe opérationnelle à cet instant t (tous les agents de C prennent l'action *Quitter*). Les valeurs $V^\pi(s^t, \vec{o}^t))$ et $V^\pi(s^t, \vec{o}^t_{-C})$ ne diffèrent alors que par la récompense immédiate que les politiques π et π_{-C} génèrent à l'instant t (puis par les transitions vers les prochains états qu'elles induisent). Bien qu'il soit possible de calculer l'utilité différentielle de n'importe quelle équipe $C \in \mathcal{C}$, cela ne fait vraiment sens, dans le cas d'un TEAM-POMDP, que lorsqu'on la calcule pour un sous-ensemble d'agents $C \subseteq C^t$ de l'équipe opérationnelle C^t. Notons que cela ne signifie pas que les agents de C vont rester hors du système et agir avec l'action *Quitter* pour tout le reste du processus une fois l'étape de décision t passée. Lorsque $C = \{i\}$, alors on parle de λ_i^π comme de l'**utilité marginale** de l'agent $i \in \mathcal{N}$.

Lorsque $\lambda_C^\pi(s^t, \vec{o}^t) > 0$, alors l'équipe C a une influence positive sur la résolution de la tâche. L'équipe C n'est pas dominée (au sens de l'équation 6.3.4) puisque, en suivant la politique π, on peut trouver une équipe (l'équipe $C^t \backslash C$, où C^t est l'équipe opérationnelle telle que prescrite par la politique π) qui aurait fait moins bien.

L'utilité différentielle telle que définie possède certains inconvénients qui limitent son applicabilité en pratique. Tout d'abord, elle ne peut pas être utilisée durant l'étape de planification

pour trouver les meilleures équipes opérationnelles à utiliser à chaque historique puisque l'utilité différentielle *a besoin* d'une politique jointe π prédéfinie pour être calculée. Elle ne peut pas non plus être utilisée par les agents durant l'exécution du processus car elle nécessite que ceux-ci aient accès à l'état réel du système s^t ainsi qu'aux historiques d'observations des autres agents, ce qui n'est pas le cas. De façon générale, calculer une fonction d'utilité différentielle de la forme $\lambda_C^\pi : \mathcal{S} \times \vec{\mathcal{O}}_C \to \mathbb{R}$ ne sera pas utile pour faire de la formation dynamique d'équipes de façon décentralisée, puisque la fonction λ_C^π elle-même ne sera pas séparable (un agent ne pourra pas en extraire son utilité personnelle). Cela soulève un problème épineux : un agent i ne peut pas déterminer sa propre utilité λ_i^π sans connaître celle des autres agents. Même s'il pouvait la calculer de façon indépendante, elle ne lui serait pas utile pour savoir quand être ou ne pas être membre de l'équipe opérationnelle, sauf s'il connaissait également l'utilité de ses coéquipiers. Les agents ont besoin d'un *signal global* pour se synchroniser.

⁎ **Signal global.** Imaginons la situation suivante, que nous avons déjà abordée en introduction de cette thèse (tableau 2.1). Une brigade de pompiers est dépêchée sur le lieu d'un incendie pour éteindre le feu se propageant à grande vitesse. Aucune habitation n'est touchée, uniquement des champs immenses et des lieux boisés. L'objectif des pompiers est de se séparer pour parvenir à éteindre le feu le plus rapidement possible, avant que la totalité de la forêt ne soit détruite par les flammes. Ils cherchent donc à minimiser la surface brûlée. Un satellite géostationnaire surveille la progression du feu et envoie régulièrement des photographies à la caserne de pompiers pour la tenir au courant des progrès de la brigade (figure 6.4). Le flux de photographies satellitaires indique aux pompiers restés à la caserne comment évolue le feu, et donne une estimation de la surface brûlée par les flammes. Il s'agit d'un signal global, une information sur laquelle chaque pompier de la caserne peut se baser, individuellement, pour décider s'il doit ou non rejoindre la brigade sur le terrain.

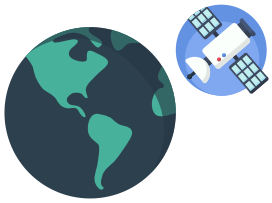


FIGURE 6.4 – Un satellite possède un point de vue plus global de la situation, et apporte donc une information supplémentaire, différente de l'observation jointe que les agents au sol peuvent effectuer.

On peut faire le parallèle avec un TEAM-POMDP. La brigade de pompiers sur le terrain correspond à l'équipe opérationnelle. Les pompiers restés à la caserne forment la réserve d'agents. Les agents de la réserve n'observent rien de ce qu'il se passe dans le système. Seul le flux de photographies envoyées par le satellite ne trouve pas de parallèle dans le cadre d'un TEAM-POMDP. Ce signal global n'est pas forcément une agrégation des observations locales des agents de l'équipe opérationnelle. C'est une observation plus globale, liée à l'état du système, qui apporte une information différente de l'observation locale d'un agent. Si l'on a préalablement planifié aux agents comment agir lorsqu'ils sont dans l'équipe opérationnelle en fonction de leurs observations locales, et lorsqu'il sont dans la réserve en fonction du signal global, alors il serait possible pour un agent de mesurer son utilité relative par rapport aux autres agents. Bien qu'il s'agisse d'une approche qui mérite que l'on s'y intéresse, cela va demeurer hors de la portée de cette thèse.

Dans la suite de ce chapitre, on examine un autre critère d'évaluation pour comparer l'effi-

cacité d'une équipe par rapport aux autres : son classement Elo.

6.4.2 Le système de classement Elo

Déterminer l'équipe la plus adaptée à un certain historique d'observations peut être vu comme trouver quelle équipe est la gagnante d'une compétition entre les différentes équipes. Intuitivement, l'équipe *gagnante* à un certain historique d'observations (et donc, la plus adaptée à cet historique) est celle qui va, en moyenne, accumuler la plus grande récompense lorsqu'on l'utilise à cette étape de décision. On va se baser sur le populaire classement Elo pour calculer les valeurs relatives de chaque équipe, et ainsi décider laquelle employer en fonction de la situation. Cette approche est centralisée par essence. En effet, comme nous venons de le voir, obtenir l'indice de pouvoir d'une équipe ne peut pas permettre aux agents de cette équipe de prendre des décisions concernant leur entrée ou sortie de l'équipe opérationnelle de manière décentralisée.

Le système de classement Elo, inventé par le professeur de physique et joueur d'échecs américain d'origine hongroise Arpad Elo au début des années 1960, a été conçu pour évaluer la force des joueurs d'échecs [Elo, 1978]. Le système assigne à chaque joueur, qu'il soit amateur ou professionnel, un score, son **classement Elo**, qui lui permet de comparer son niveau de compétence au jeu à celui des autres joueurs. L'idée de base du système de classement Elo est de mettre à jour le classement Elo d'un joueur après chaque match joué en fonction du niveau de compétence de son adversaire. Une victoire sur un joueur ayant un classement Elo plus faible ne va que légèrement augmenter le classement Elo, tandis qu'une victoire contre un joueur avec un classement plus élevé va induire un gain plus important (et de façon équivalente pour les défaites). Ce qui rend le système de classement Elo attrayant est qu'il ne faut généralement que quelques matchs pour évaluer le niveau réel d'un joueur.

Outre les échecs, le système de classement Elo et ses variantes ont également été utilisés dans les jeux compétitifs en ligne, le sport professionnel, le Go, etc. D'autres systèmes de classement et de notation ont depuis émergé, soit pour les échecs, comme le système Glicko [Glickman, 1998] (qui prend en compte l'incertitude sur le classement réel d'un joueur), soit pour le matchmaking de jeux vidéo en ligne, comme le système de classement bayésien de Microsoft, TrueSkill$^{\text{TM}}$ [Herbrich et al., 2007] (qui peut considérer des matchs entre plus de deux joueurs).

Considérons un historique d'observations partiel joint $\vec{o}^t$ à l'instant t, et deux équipes d'agents C_1,$C_2 \in \mathcal{C}$. On ne fait pas d'hypothèses concernant les équipes C_1 et C_2. Par exemple, l'une peut être un sous-ensemble de l'autre. Dans le système de classement Elo adapté aux TEAM-POMDPs, les équipes C_1 et C_2 commencent respectivement avec des classements Elo

$$E(\vec{o}^t,C_1),E(\vec{o}^t,C_2) \in \mathbb{R}$$

à l'historique $\vec{o}^t$. Ces deux valeurs peuvent être arbitrairement choisies et ne pas nécessairement contenir d'information sur la vraie utilité différentielle de chacune de ces équipes. De plus, elles ne nous informent pas non plus sur l'utilité relative d'une équipe par rapport à l'autre. Il faut donc qu'elles soient initialisées puis continuellement mises à jour afin de correspondre à la vraie valeur relative des équipes. L'équation de mise à jour que l'on utilise est celle traditionnellement utilisée dans le domaine des échecs. Supposons que l'on ait effectué un match entre les équipes C_1 et $C_2 \in \mathcal{C}$ à l'historique $\vec{o}^t$ (où le sens de "match" entre deux équipes sera défini ci-dessous), alors le nouveau classement Elo de l'équipe C_1 après son match contre C_2 est :

$$E(\vec{o}^t,C_1) \leftarrow E(\vec{o}^t,C_1) + \kappa(W_{C_1,C_2} - L_{C_1,C_2}), \tag{6.19}$$

où

— $\kappa \in \mathbb{R}$ est une constante permettant de contrôler la volatilité et la stabilité du classement,

— W_{C_1,C_2} est le résultat *réel* du match entre C_1 and C_2, tel que

$$W_{C_1,C_2} = \begin{cases} 1 & \text{si } C_1 \text{ bat } C_2 \\ 0.5 & \text{en cas d'égalité} \\ 0 & \text{sinon,} \end{cases} \tag{6.20}$$

— et L_{C_1,C_2} est le résultat *attendu* du match entre C_1 and C_2. Il s'agit de la probabilité de victoire de l'équipe C_1 sur l'équipe C_2, calculée avec :

$$L_{C_1,C_2} = \frac{1}{1 + 10^{-D_{C_1,C_2}}} \tag{6.21}$$

avec

$$D_{C_1,C_2} = \frac{E(\vec{o}^t,C_2) - E(\vec{o}^t,C_1)}{\xi}, \tag{6.22}$$

et $\xi \in \mathbb{R}$ est une constante utilisée pour contrôler la propagation des classements. Elle permet de limiter la gamme de classements Elo possibles, et peut être choisie arbitrairement. Dans le cas des échecs, on choisit généralement $\xi = 400$.

Par exemple, supposons que l'équipe C_1 a battu l'équipe C_2 et que $L_{C_1,C_2} = 0$, c'est-à-dire que la probabilité que C_1 gagne contre C_2 était nulle (car C_2 était considérée trop forte pour perdre contre C_1). Alors le nouveau classement Elo de C_1 est $E^{\vec{o}^t}_{C_1} + \kappa$. Lorsque la probabilité de victoire L_{C_1,C_2} équivaut exactement au résultat réel W_{C_1,C_2} du match, alors les classements Elo des deux équipes ne sont pas modifiés. De plus, lorsque l'équipe gagnante augmente son classement Elo de x points, l'équipe perdante perd elle exactement x points.

En effectuant assez de matchs entre une équipe et toutes les autres, et en mettant à chaque fois à jour le classement Elo de cette équipe via l'équation 6.19, la valeur va converger vers le vrai niveau de compétence de l'équipe.

6.4.3 Matchs

Rappelons-nous que notre objectif est de déterminer quelles équipes sont les plus adaptées à un certain historique d'observations partiel joint, et cela pour chaque historique possible. Une façon de déterminer si une équipe C_1 est meilleure qu'une équipe C_2 à un historique $\vec{o}^t$ est de comparer la somme des récompenses futures que l'on peut obtenir après avoir utilisé C_1 ou C_2 à l'historique $\vec{o}^t$.

Le protocole que l'on utilise pour un **match** est le suivant.

1. Choisir une équipe $C_1 \in \mathcal{C}$.
2. Choisir une action jointe $a^t_{C_1} \in \mathcal{A}_{C_1}$ et exécuter cette action jointe à l'historique joint courant, dans l'état courant s^t.
3. Générer une *récompense cumulée* sur les $h - t$ étapes de décision restantes

$$\Delta_{C_1} = R_{C_1}(s^t,a^t_{C_1}) + F^{h-t} \tag{6.23}$$

où $R_{C_1}(s^t,a^t_{C_1})$ est la récompense immédiate obtenue après avoir agi avec $a^t_{C_1}$ dans l'état s^t, telle que spécifiée par la fonction de récompense $R_{C_1} \in \boldsymbol{R}$ du TEAM-POMDP, et où F^{h-t} est une estimation de la somme des récompenses futures pour les $h - t$ étapes de décision restantes.

On exécute ces trois étapes pour l'équipe C_2 et on obtient ainsi une autre récompense cumulée Δ_{C_2}. Le gagnant entre C_1 et C_2 est défini comme l'équipe ayant générée la plus grande récompense cumulée, c'est-à-dire, pour reprendre l'équation 6.20 :

$$W_{C_1,C_2} = \begin{cases} 1 & \text{si } \Delta_{C_1} > \Delta_{C_2} \\ 0.5 & \text{si } \Delta_{C_1} = \Delta_{C_2} \\ 0 & \text{sinon.} \end{cases} \tag{6.24}$$

Dans la section 6.5, on va décrire explicitement comment ce protocole de match peut être appliqué durant la phase de planification, et comment est calculée l'heuristique F^{h-t}.

6.4.4 Volatilité et stabilité

Le facteur $\kappa \in \mathbb{R}$ dans l'équation 6.19 est le paramètre de *volatilité*. Il s'agit du montant maximum qu'un classement Elo peut gagner ou perdre après un match. Si κ est grand, une victoire ou une défaite aura un plus grand impact sur le nouveau classement Elo. À l'inverse, si κ est petit, le classement Elo ne va pas énormément changer à chaque mise à jour. Dans le cas des échecs, κ est choisi grand (environ 32, selon la compétition) pour les joueurs ayant peu de matchs à leur actif, et plus petit (environ 16) pour les joueurs plus expérimentés. En effet, le système nécessite parfois de nombreux matchs pour déterminer le niveau de compétence exact d'un nouveau joueur. Par conséquent, utiliser un κ grand permet d'atteindre rapidement le classement Elo adéquat. Une fois que celui-ci est atteint, un κ plus petit peut être utilisé pour éviter les fluctuations trop importantes après chaque victoire ou défaite contre un joueur ayant un niveau de jeu similaire.

6.5 L'algorithme MELO

Dans cette section, on se propose de modifier l'algorithme POS-UCT tel que présenté dans le chapitre précédent en section 5.5, afin de le coupler avec le système de classement Elo ci-dessus. Notre but ici est de calculer une politique jointe, mais pas forcément séparable. Contrairement à l'algorithme BRD-POS-UCT, où POS-UCT était utilisé pour calculer la meilleure réponse d'un agent à un ensemble de politiques fixes, nous utilisons ici POS-UCT pour calculer directement une politique jointe. Dans un second temps, nous entrelaçons l'algorithme POS-UCT adapté au cas multi-agents avec le système de classement Elo pour créer notre algorithme : **recherche arborescente Monte-Carlo avec classement Elo** (MELO, *Monte-Carlo Tree Search with Elo rating system*).

Rappelons que l'une des propriétés les plus intéressantes des algorithmes basés sur MCTS est que ceux-ci sont conçus pour n'explorer que les portions significatives de l'espace de recherche. Cela signifie que les états indésirables, où les actions ne peuvent que mener à de mauvaises récompenses à long terme, sont rarement visités. L'arbre construit par l'algorithme est donc asymétrique et favorise les états prometteurs, ce qui est une caractéristique souhaitable lorsque l'on considère des jeux ou des problèmes avec un facteur de branchement élevé et où de nombreux états ont une récompense espérée faible. C'est là que nous pensons que l'algorithme MCTS pourrait être utile pour le problème de formation dynamique d'équipes. Étant donné que de nombreuses équipes peuvent avoir une valeur moyenne relativement faible à un certain historique d'observations joint, il n'est pas utile d'essayer toutes les actions jointes de ces équipes pour vérifier si l'une d'entre elles est réellement bonne. Si nous nous attendons à ce qu'une équipe

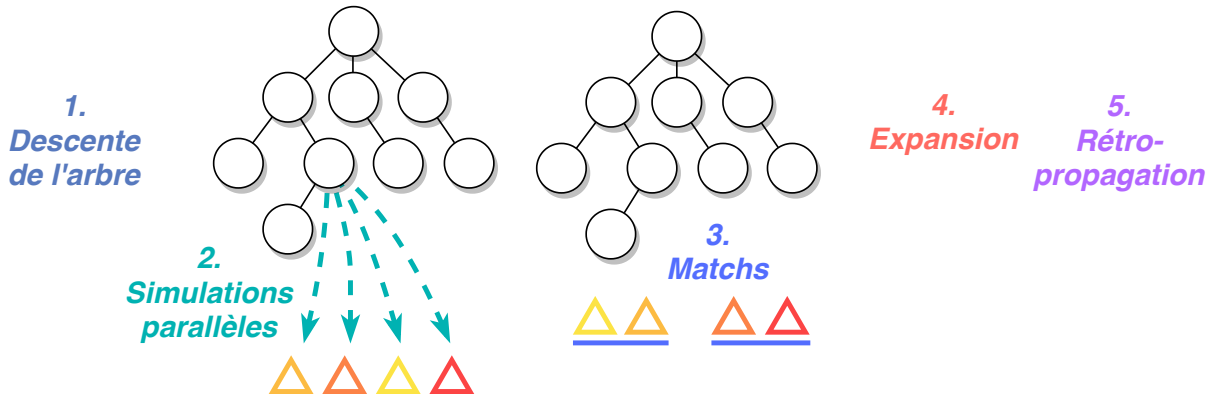


FIGURE 6.5 – Illustration des cinq étapes principales de l'algorithme MELO. Seules les étapes 2 et 3 sont représentées en détail. Les étapes 1, 4 et 5 sont sensiblement les mêmes que celles de l'algorithme POS-UCT (figure 5.3).

soit, en moyenne, toujours meilleure qu'une autre équipe, nous pourrons nous focaliser sur celle-ci et ainsi élaguer de grandes portions de l'espace de recherche en ignorant certaines équipes à certains historiques joints.

6.5.1 Recherche arborescente Monte-Carlo avec classement Elo

L'algorithme 4 présente les grandes lignes de l'algorithme de recherche arborescente Monte-Carlo avec classement Elo, tandis que l'algorithme 5 en donne une version détaillée. La figure 6.5 présente les différentes étapes de l'algorithme, et illustre plus particulièrement celles qui varient par rapport à l'algorithme POS-UCT.

Algorithme 4 : L'algorithme MELO.

Données : Un TEAM-POMDP et $maxIter \in \mathbb{N}$
Résultat : Une politique jointe π

$iter \leftarrow 0$
Initialiser le nœud racine $\vec{o}^0$
tant que $iter < maxIter$ **faire**
  $\vec{o}^x \leftarrow$ `Descente`$(\vec{o}^0)$
  $\mathbf{\Delta} \leftarrow$ `SimlationsParallèles`$(\vec{o}^x)$
  $C^* \leftarrow$ `Matchs`$(\vec{o}^x, \mathbf{\Delta})$
  $\Delta^* \leftarrow \Delta_{C^*} \in \mathbf{\Delta}$
  `Expansion`$(\vec{o}^x, C^*)$
  `Rétropropagation`$(\vec{o}^x, \Delta^*)$
  $iter \leftarrow iter + 1$
retourner π

Dans le contexte d'un TEAM-POMDP, l'objectif de l'algorithme MELO est de construire incrémentalement une politique jointe, de l'étape de décision 0 à l'étape de décision $h - 1$. Pour cela, on construit un arbre, appelé *arbre de recherche*, qui va au final correspondre à une politique jointe stochastique. En soi, la structure de l'arbre de recherche est très semblable à celle de l'arbre construit via l'algorithme POS-UCT appliqué à un POMDP (algorithme 2).

Algorithme 5 : L'algorithme MELO (version étendue).

Données : Un TEAM-POMDP et $maxIter \in \mathbb{N}$
Résultat : Une politique π

Fonction `Résoudre` **:**
  $iter \leftarrow 0$
  Initialiser le nœud racine $\vec{o}^0$
  $\Psi \leftarrow [\varnothing]$
  tant que $iter < maxIter$ **faire**
    $\vec{o}^x \leftarrow$ `Descente`$(\vec{o}^0, \Psi)$
    $\mathbf{\Delta} \leftarrow$ `SimlationsParallèles`$(\vec{o}^x, \Psi)$
    $C^* \leftarrow$ `Matchs`$(\vec{o}^x, \mathbf{\Delta})$
    $\Delta^* \leftarrow \Delta_{C^*} \in \mathbf{\Delta}$
    `Expansion`$(\vec{o}^x, C^*, \Psi)$
    `Rétropropagation`$(\Delta^*, \vec{o}^x, \Psi)$
    $iter \leftarrow iter + 1$
  pour tous $\vec{o}^t$ **faire**
    $\pi(\vec{o}^t) \leftarrow \underset{a_C \in \mathcal{A}}{\operatorname{argmax}} \frac{Q(\vec{o}^t, a_C)}{N(\vec{o}^t, a_C)}$
  retourner π_i

Fonction `Descente`$(\vec{o}^0, \Psi)$ **:**
  $\vec{o}^t \leftarrow \vec{o}^0$
  tant que $\vec{o}^t$ *n'est pas extensible* **faire**
    $\bar{a}_C^t \leftarrow \underset{a_C \in \mathcal{A}}{\operatorname{argmax}} \frac{Q(\vec{o}^t, a_C)}{N(\vec{o}^t, a_C)} + \chi \cdot \sqrt{\frac{\ln N(\vec{o}^t)}{N(\vec{o}^t, a_C)}}$
    Effectuer $\bar{a}_C^t$, recevoir o_C^{t+1} et R^{t+1}
    $\vec{o}^t \leftarrow (\vec{o}^t, o_C^{t+1})$
    $\Psi \leftarrow \Psi \cup [\bar{a}_C^t, o_C^{t+1}, R^{t+1}]$
  retourner $\vec{o}_i^t$

Fonction `Matchs`$(\vec{o}^x, \mathbf{\Delta})$ **:**
  Trier $\mathbf{\Delta}$ par ordre croissant
  pour tous $\Delta_{C_i} \in \mathbf{\Delta}$ **faire**
    $C_1 \leftarrow C_i$
    $C_2 \leftarrow C_{i+1}$
    $E(\vec{o}^x, C_1) \leftarrow E(\vec{o}^x, C_1) + \kappa(W_{C_1,C_2} - L_{C_1,C_2})$
    $E(\vec{o}^x, C_2) \leftarrow E(\vec{o}^x, C_2) + \kappa(W_{C_2,C_1} - L_{C_2,C_1})$
  retourner $\underset{C}{\operatorname{argmax}}\, E(\vec{o}^x, C)$

Fonction `SimlationsParallèles`$(\vec{o}_i^x, \Psi)$ **:**
  $\Psi^\star \leftarrow \Psi$
  $\mathbf{\Delta} \leftarrow [\,]$
  pour tous $C \in \mathcal{C}$ **faire**
    $t \leftarrow x$
    $\Psi \leftarrow \Psi^\star$
    Choisir $a_C^t \in \mathcal{A}_C$ aléatoirement
    Effectuer a^t, recevoir o^t et R^{t+1}
    $t \leftarrow t + 1$
    tant que $t < h$ **faire**
      Choisir $a^t \in \mathcal{A}$ aléatoirement
      Effectuer a^t, recevoir o^t et R^{t+1}
      $\Psi \leftarrow \Psi \cup [\bar{a}_C^t, o_C^{t+1}, R^{t+1}]$
      $t \leftarrow t + 1$
    $\Delta_C \leftarrow 0$
    pour tous $R^t \in \Psi$ **faire**
      $\Delta_C \leftarrow \Delta_C + R^t$
    $\mathbf{\Delta} \leftarrow \mathbf{\Delta} \cup \Delta_C$
  $\Psi \leftarrow \Psi^\star$
  retourner $\mathbf{\Delta}$

Fonction `Expansion`$(\vec{o}_i^x, C^*, \Psi)$ **:**
  Choisir $a_C^x \in \mathcal{A}_C$ telle que $N(\vec{o}^x, a_C^x) = 0$
  Effectuer a_C^x, recevoir o_C^{x+1} et R^{x+1}
  $f(\vec{o}^x) \leftarrow (\vec{o}^x, o_C^{x+1})$
  Concaténer le nœud $f(\vec{o}^x)$ à $\vec{o}^x$
  $\Psi \leftarrow \Psi \cup [a_C^x, o_C^{x+1}, R^{x+1}]$
  retourner $f(\vec{o}^x)$

Fonction `Rétropropagation`$(\Delta, \vec{o}^x, \Psi)$ **:**
  $\vec{o}^t \leftarrow \vec{o}^x$
  tant que $t \geqslant 0$ **faire**
    Extraire a^t de Ψ
    $Q(\vec{o}^t, a^t) \leftarrow Q(\vec{o}^t, a^t) + \Delta$
    $N(\vec{o}^t, a^t) \leftarrow N(\vec{o}^t, a^t) + 1$
    $t \leftarrow t - 1$

Simplement, les nœuds stockent ici des historiques d'observations partiels joints plutôt que individuels. Formellement, chaque nœud à la profondeur t dans l'arbre de recherche correspond à un historique d'observations partiel joint $\vec{o}^t$ et stocke quatre informations essentielles :

1. une observation partielle jointe $o \in \mathcal{O}$,
2. un vecteur N de taille $|\boldsymbol{\mathcal{A}}|$ de **nombre de visites**, tel que $N(\vec{o}^t,a_C)$ est le nombre de fois que l'action jointe $a_C \in \mathcal{A}_C$ a été essayée à l'historique partiel joint $\vec{o}^t$,
3. un vecteur Q de taille $|\boldsymbol{\mathcal{A}}|$ de **valeurs estimées**, tel que $Q(\vec{o}^t,a_C)$ correspond à la somme des récompenses cumulées pour agir à l'historique $\vec{o}^t$ avec l'action jointe a_C,
4. un vecteur E de taille $|\mathcal{C}|$ de **classements Elo**, tel que $E(\vec{o}^t,C)$ est le classement Elo de l'équipe C à l'historique $\vec{o}^t$.

Remarquons que, durant l'implémentation de l'algorithme, il n'est pas nécessaire d'initialiser les vecteurs N, Q et E avec leur taille maximale ($|\boldsymbol{\mathcal{A}}|$, $|\boldsymbol{\mathcal{A}}|$ et $|\mathcal{C}|$, respectivement), ni de donner des valeurs par défaut à toutes les actions jointes de toutes les équipes. Il est possible d'augmenter au besoin la taille de ces vecteurs au fur et à mesure que les actions jointes des différentes équipes sont essayées par l'algorithme. De façon analogue à ce que nous avons dit concernant l'algorithme BRD-POS-UCT, un nœud de l'arbre fera référence à un historique d'observations partiel joint.

L'arbre de recherche est construit en répétant cinq étapes.

1. Descente de l'arbre. Cette étape est identique à l'étape de descente de l'arbre de l'algorithme POS-UCT telle que décrite dans l'algorithme 2. On descend l'arbre de recherche en sélectionnant les actions jointes prometteuses. À chaque nœud, l'action jointe $\bar{a}_C$ choisie est celle qui maximise UCB, de façon similaire à ce qui a été présenté avec l'équation 5.14 :

$$\bar{a}_C = \underset{a_C \in \mathcal{A}}{\operatorname{argmax}} \frac{Q(\vec{o}^t,a_C)}{N(\vec{o}^t,a_C)} + \chi \cdot \sqrt{\frac{\ln N(\vec{o}^t)}{N(\vec{o}^t,a_C)}} \tag{6.25}$$

où

$$N(\vec{o}^t) = \sum_{C \in \mathcal{C}} \sum_{a_C \in \mathcal{A}_C} N(\vec{o}^t,a_C) \tag{6.26}$$

est le nombre total de visites du nœud courant, et χ est le paramètre d'exploration. Lorsqu'une action est choisie, une observation partielle jointe est émise, on descend dans l'arbre de recherche vers cette nouvelle observation (en concaténant un nouveau nœud au besoin) et ainsi de suite jusqu'à atteindre un nœud à étendre. Dans MELO, un nœud est *extensible* si toutes les actions jointes de l'équipe *ayant le meilleur classement Elo* n'ont pas été essayées au moins une fois à l'historique correspondant.

2. Simulations parallèles. Cette étape est assez différente de l'algorithme POS-UCT. Dans l'algorithme POS-UCT tel que présenté plus haut, quand un nœud extensible est atteint, on l'étend directement en essayant aléatoirement une action (individuelle) qui n'a pas encore été essayée, puis on simule une trajectoire d'actions aléatoires jusqu'à atteindre l'horizon de planification.

Ici, durant les *simulations parallèles*, on procède comme dans le protocole de matchs décrit dans la section 6.4.3. Supposons que nous sommes à l'historique partiel joint $\vec{o}^t$. En parallèle, et *indépendamment* pour chaque équipe $C \in \mathcal{C}$, on commence par choisir une action jointe $a_C^t \in \mathcal{A}_C$, on l'exécute à cet historique joint puis on génère une trajectoire aléatoire en sélectionnant

des actions jointes aléatoires d'équipes aléatoires, jusqu'à atteindre l'horizon. À la fin de cette trajectoire, une récompense Δ_C est générée pour l'équipe C qui a initialement agi avec l'action jointe a_C^t. Cette récompense correspond à celle décrite dans l'équation 6.23 :

$$\Delta_C = R_C(s^t,a_C^t) + R^{t+1} + R^{t+2} + \cdots + R^{h-1}. \tag{6.27}$$

Il s'agit de la récompense immédiate $R_C(s^t,a_C^t)$ plus la somme des récompenses

$$F^{h-t} = R^{t+1} + R^{t+2} + \cdots + R^{h-1}, \tag{6.28}$$

où chaque $R^{t'}$ est obtenu en agissant avec l'action jointe aléatoire d'une équipe aléatoire à l'étape de décision $t' \in [t+1,h-1]$.

3. Matchs par paires. Une fois que les récompenses cumulées

$$\boldsymbol{\Delta} = [\Delta_{C_1},\Delta_{C_2},\ldots,\Delta_{C_{|\mathcal{C}|}}]$$

ont été générées, une par équipe, on les trie par ordre croissant et on effectue une session de matchs par paires entre les équipes. Cela signifie que l'équipe qui a produit la plus grande récompense cumulée à la fin de ses simulations parallèles affrontera la deuxième meilleure équipe. Et cette deuxième meilleure équipe affrontera (également) la troisième meilleure équipe, etc. Pour chaque match entre deux équipes C_1 et C_2, on met à jour les classements Elo $E(\vec{o}^t,C_1)$ et $E(\vec{o}^t,C_2)$ à l'historique partiel joint $\vec{o}^t$ avec l'équation 6.19. Remarquons que chaque équipe aura donc deux matchs (et verra donc son classement être mis à jour deux fois) sauf la meilleure et la pire équipe, qui n'auront qu'un seul match.

Par exemple, considérons trois équipes $\{A,B,C\} \subseteq \mathcal{C}$ avec des récompenses cumulées respectives telles que $\Delta_A > \Delta_B > \Delta_C$ à la fin des simulations parallèles. Alors les matchs par paires seront : A *vs.* B et B *vs.* C.

4. Expansion. Cette étape consiste à faire grandir l'arbre de recherche. Remarquons que pendant l'étape des simulations parallèles, aucun nouveau nœud n'a été ajouté à l'arbre de recherche. Cette étape, avec l'étape des matchs par paires, ne servaient qu'à mettre à jour les classements Elo des différentes équipes. De toutes les équipes de $\mathcal{C}$, une, notée $C^* \in \mathcal{C}$, possède le meilleur classement Elo à l'historique partiel joint $\vec{o}^t$:

$$\exists C^* \in \mathcal{C},\forall C \in \mathcal{C},C \neq C^*,E(\vec{o}^t,C^*) \geqslant E(\vec{o}^t,C). \tag{6.29}$$

On étend l'arbre de recherche en agissant avec une action aléatoire (ou une action qui n'a pas encore été essayée, si possible) $a_{C^*} \in \mathcal{A}_{C^*}$ de cette meilleure équipe $C^* \in \mathcal{C}$. Étendre l'arbre de recherche *après* l'étape des simulations parallèles et non avant (comme dans l'algorithme POS-UCT), nous assure de ne favoriser que les meilleures équipes et nous évite ainsi d'avoir à explorer les mauvaises actions jointes d'équipes sous-optimales.

Dans le nouveau nœud créé, les classements Elo de toutes les équipes sont initialisés avec les classements Elo du nœud parent que l'on vient d'étendre.

5. Rétro-propagation de la récompense. La rétro-propagation de la récompense cumulée est la dernière étape, et elle est exactement identique à celle de l'algorithme POS-UCT. La récompense cumulée $\Delta^* = \Delta_{C^*}$ de l'équipe C^* ayant le meilleur classement Elo à la fin de l'étape des simulations parallèles est retro-propagée à tous les nœuds et actions jointes visités

durant les étapes de descente de l'arbre et d'expansion. Ces nœuds sont mis à jour de la façon suivante :

$$Q(\vec{o}_i^t,a_i) \leftarrow Q(\vec{o}_i^t,a_i) + \Delta^*. \tag{6.30}$$

Quant aux nombres de visites des nœuds visités, ils sont incrémentés de un :

$$N(\vec{o}_i^t,a_i) \leftarrow N(\vec{o}_i^t,a_i) + 1. \tag{6.31}$$

La répétition de ces cinq étapes constitue l'algorithme MELO. À chaque itération, les états et observations sont échantillonnés initialement selon l'état de croyance b^0 puis normalement selon les fonctions de transition et d'observation du modèle. En effectuant un nombre suffisant d'itérations, les classements Elo des différentes équipes et les valeurs estimées des actions jointes vont refléter leur vraie valeur dans les historiques partiels concernés.

La terminaison de l'algorithme MELO est semblable à celle de l'algorithme POS-UCT. Une fois le budget de calcul alloué dépassé, l'arbre de recherche contient une politique jointe stochastique, que l'on peut rendre déterministe en sélectionnant, à chaque historique, l'action jointe gloutonne :

$$\bar{a}_C = \underset{a_C \in \mathcal{A}}{\operatorname{argmax}} \frac{Q(\vec{o}^t,a_C)}{N(\vec{o}^t,a_C)}. \tag{6.32}$$

6.5.2 Matchs des meilleures équipes

Dans une itération de l'algorithme, après la descente de l'arbre, on effectue des simulations parallèles pour chaque équipe possible, on génère une récompense cumulée pour chacune d'entre elles et on effectue une série de matchs par paires. Cependant, le nombre d'équipes possibles peut être très conséquent selon la taille de la population initiale $\mathcal{N}$ du TEAM-POMDP. Pour remédier à cela, et rendre notre algorithme plus pratique à appliquer, on suggère d'effectuer des simulations parallèles (et donc, des mises à jour du classement Elo) uniquement entre les $\alpha \in \mathbb{N}$ meilleures équipes à un historique joint, tel que $1 < \alpha \leqslant |\mathcal{C}|$. On requiert que α soit plus grand que 1, car il faut pouvoir effectuer au moins un match pour mettre à jour les classements Elo. Ainsi, les équipes plus faibles verront leur classement Elo être moins souvent mis à jour.

Considérons l'exemple suivant avec trois équipes $\mathcal{C} = \{A,B,C\}$, et supposons que leurs classements Elo à l'historique partiel joint actuel soient, respectivement, 1520, 1480 et 1500. Si l'on prend $\alpha = 2$, alors on n'effectuera que deux simulations en parallèle, une pour l'équipe A, l'autre pour l'équipe C, car elles ont les plus hauts classements Elo des trois équipes. Supposons à présent que C sorte vainqueur du match A *vs.* C. Les nouveaux classements Elo pourront devenir 1470, 1480 et 1550 pour A, B et C, respectivement. La prochaine fois que cet historique partiel joint sera visité, les simulations parallèles, matchs et mises à jour de classement auront donc lieu entre les équipes B et C.

Utiliser uniquement les α meilleures équipes durant les phases de simulations parallèles et de matchs par paires ne garantit pas que l'algorithme MELO trouvera plus facilement de meilleures politiques jointes. Il se peut même que l'espace des solutions devienne trop élagué, ce qui rendrait certaines solutions inatteignables. Il s'agit néanmoins d'une heuristique raisonnable pour diminuer le nombre de simulations parallèles et de matchs à effectuer, tout en nous assurant que les bonnes équipes seront testées et leur classement Elo mis à jour assez régulièrement.

6.5.3 Justification des classements Elo

À ce stade, il est légitime de se demander pourquoi maintenir des classements Elo est une meilleure idée que de comparer directement les récompenses cumulées moyennes des différentes

équipes. En somme, la phase de matchs par paires ne pourrait-elle pas être écartée, et l'expansion de l'arbre de recherche se faire avec une action aléatoire de l'équipe qui génère, en moyenne, la meilleure récompense cumulée, plutôt que de l'équipe ayant le meilleur classement Elo ?

La réponse rapide est que le classement Elo permet d'obtenir une valeur *relative* de la force d'une équipe par rapport aux autres équipes, là où la récompense cumulée moyenne est une valeur *absolue* qui mesure la performance de l'équipe à résoudre la tâche. La récompense cumulée obtenue par une équipe à la fin d'une simulation parallèle n'est pas forcément représentative de la valeur réelle des récompenses que cette équipe peut générer. Il se peut qu'une simulation parallèle n'utilise que les pires équipes et les pires actions jointes de ces équipes, générant au final une récompense cumulée très faible. Il faut donc échantillonner un grand nombre de trajectoires et de récompenses, pour toutes les équipes, pour obtenir une valeur représentative de la récompense cumulée moyenne de ces équipes. Or, avec un nombre fixe d'itérations dans l'algorithme, et du fait que le facteur de branchement dans l'arbre peut être grand, toutes les actions jointes de toutes les équipes ne seront pas essayées un nombre suffisant de fois à chaque historique joint pour nous permettre d'obtenir une estimation correcte de la valeur de toutes les équipes. Ce phénomène est également présent dans l'algorithme MCTS de base, mais il est particulièrement impactant lorsque les espaces des actions jointes et des équipes sont grands, ce qui est le cas quand on considère de nombreuses équipes possibles. Notre idée avec le classement Elo est de pouvoir atténuer les effets d'un nombre insuffisant d'échantillons en trouvant rapidement et en se concentrant sur les équipes les plus prometteuses. Une équipe n'est pas jugée "bonne" par rapport à sa capacité à résoudre la tâche, mais plutôt par rapport à ses performances comparées aux autres équipes. Le système Elo étant généralement utilisé, dans le domaine des échecs, pour parvenir à déterminer en peu de matchs le placement réel d'un joueur par rapport aux autres joueurs, nous l'utilisons également à cette fin.

Finalement, remarquons que le classement Elo permet de comparer des équipes même s'il n'y a jamais eu de matchs entre elles. Considérons un exemple avec trois équipes $\mathcal{C} = \{A,B,C\}$. Si on organise deux matchs, A *vs.* C où A est vainqueur, et B *vs.* C, où C est vainqueur, alors on peut dire que A est meilleure que B même s'il n'y a pas eu de matchs entre elles (notons qu'il s'agit simplement d'une illustration, dans la pratique il faudrait comparer les classements Elo de chaque équipe et effectuer plusieurs matchs A *vs.* C et B *vs.* C). Comme nous l'avons vu dans la section 6.5.2, nous n'effectuons de matchs qu'entre les $\alpha \leqslant |\mathcal{C}|$ meilleures équipes. Utiliser des classements Elo nous permet donc de maintenir les rapports de force entre les équipes même si toutes ne sont pas essayées un grand nombre de fois.

6.5.4 Séparabilité

La politique jointe calculée par l'algorithme MELO est, par définition, non séparable. Cela signifie que l'on ne peut pas l'écrire sous la forme $\pi = (\pi_i \mid \forall i \in \mathcal{N})$. Cela est dû au fait que notre algorithme construit un unique arbre de recherche, correspondant à une politique jointe. Cette politique jointe peut associer différentes actions individuelles pour un même historique d'observations partiel individuel, en fonction des historiques des autres agents, voire peut ne pas associer d'actions du tout car certains historiques individuels peuvent être tout simplement absents de l'arbre. La figure 6.6 illustre ce problème dans le cadre d'un DEC-POMDP.

La non-séparabilité est à la fois l'une des forces et des faiblesses de notre approche : on peut trouver rapidement les bonnes équipes, mais il sera complexe de pouvoir appliquer la politique de façon décentralisée. On identifie ici plusieurs façons de contourner ce problème. Soit $i \in \mathcal{N}$ et $\vec{o}_i^t$ son historique d'observation à l'instant t.

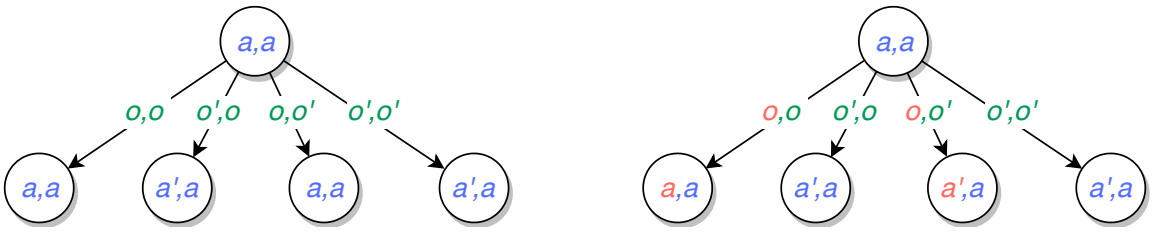


FIGURE 6.6 – Politiques jointes pour un DEC-POMDP avec deux agents, ayant chacun deux actions $\{ a, a' \}$ et deux observations $\{ o, o' \}$. **Gauche :** Une politique jointe séparable. **Droite :** Une politique jointe non séparable. Si l'agent 1 fait l'observation o, il doit effectuer l'action a si l'agent 2 fait l'observation o, et l'action a' sinon.

Action aléatoire. La première solution, si l'historique individuel $\vec{o}_i^t$ est absent de l'arbre, est simplement d'associer une action individuelle aléatoire à cet historique. Dans la pratique, plutôt que de renseigner les parties manquantes de la politique individuelle, l'agent agira simplement avec une action aléatoire s'il lui arrive de rencontrer l'un de ces historiques d'observations manquants.

Action la plus probable. Si plusieurs actions individuelles $a_i,a_i',a_i'',\ldots$ existent pour un même historique d'observations individuel $\vec{o}_i^t$, il y en a (peut-être) une qui est plus souvent utilisée que les autres. Notons

$$\mathcal{A}_i^{\vec{o}_i^t} = \{a_i,a_i',a_i'',\ldots,a_i^{(k)}\} \tag{6.33}$$

l'ensemble non-vide des actions de l'agent i prescrites au moins une fois à l'historique $\vec{o}_i^t$ par la politique jointe π. L'agent peut décider d'agir avec l'action individuelle $a_i \in \mathcal{A}_i^{\vec{o}_i^t}$ qui est la plus souvent prescrite par la politique jointe compte tenu des historiques des autres agents :

$$a_i = \operatorname*{argmax}_{a_i \in \mathcal{A}_i^{\vec{o}_i^t}} \sum_{a_i' \in \mathcal{A}_i^{\vec{o}_i^t}} \mathbb{1}_{a_i,a_i'} \tag{6.34}$$

où

$$\mathbb{1}_{x,y} = \begin{cases} 1 & \text{si } x = y, \\ 0 & \textit{sinon} \end{cases} \tag{6.35}$$

est la fonction indicatrice. Remarquons que si la politique jointe π était séparable, alors, pour tout historique individuel $\vec{o}_i^t$, on aurait $a_i = a_i', \forall a_i,a_i' \in \mathcal{A}_i^{\vec{o}_i^t}$. Il n'y aurait donc qu'une seule et unique action prescrite pour cet historique, peu importe les historiques des autres agents.

Action optimiste. Enfin, une autre façon pourrait être de sélectionner l'action individuelle *optimiste*, c'est-à-dire l'action a_i qui maximise la récompense espérée à l'historique joint $\vec{o}^t$:

$$a_i = \operatorname*{argmax}_{a_i \in \mathcal{A}_i^{\vec{o}_i^t}} \mathbb{E}\left[\sum_t^{h-1} R^t \mid \pi,b^0,a_i^t = a_i\right]. \tag{6.36}$$

D'autres recherches restent à être menées pour s'assurer de la viabilité de telles méthodes de séparation, ou pour trouver un moyen de maintenir la séparabilité future de la politique jointe au fur et à mesure de la construction de l'arbre de recherche. En attendant, l'algorithme MELO ne peut que calculer des politiques centralisées, et nous ne nous servirons pas des pistes données dans cette section.

6.6 Conclusion

Ce chapitre a présenté le problème de la formation dynamique d'équipes sous incertitude sous un nouvel angle. Nous avons mis de côté le calcul de meilleures réponses étudié au chapitre précédent pour nous focaliser sur les propriétés internes de notre modèle de décision. Les TEAM-POMDPs et les OPEN-DEC-POMDPs à contrôle total sont des modèles similaires en soi, proches des DEC-POMDPs mais qui permettent d'analyser plus efficacement le problème de la formation et de la formation dynamique d'équipes sous incertitude.

Comme un TEAM-POMDP est une généralisation du modèle de POMDP décentralisé, il souffre de la même complexité algorithmique. Lorsque le nombre d'agents dans la population augmente, le problème visant à trouver une politique jointe optimale devient exponentiellement plus dur. D'où le besoin de trouver des solutions approximatives qui autorisent un bon passage à l'échelle. Notre idée, avec l'algorithme MELO, est de rapidement exclure des équipes d'agents du processus de décision en remarquant que certaines d'entre elles sont inefficaces, comparées à d'autres équipes, dans certaines situations (une situation correspondant, ici, à un historique joint d'observations). Nous nous sommes basés sur le système de classement Elo pour classer les équipes et identifier rapidement celles qui sont efficaces à un historique donné. Afin de pouvoir faire de la planification sous incertitude, nous avons intégré ce système d'indices de pouvoir dans l'algorithme POS-UCT afin de calculer hors-ligne des politiques jointes non séparables. Les futures recherches doivent porter sur la conception d'algorithmes et de méthodes pouvant produire des politiques jointes séparables optimales, tout en s'appuyant sur certaines des caractéristiques liées aux problèmes ouverts : intuitivement, si un agent peut sortir du système, alors la complexité de résolution est censée pouvoir diminuer.

Quatrième partie

Évaluation expérimentale

Chapitre 7

Benchmarks

Sommaire

7.1 Introduction 116
7.2 Le problème de la zone sinistrée 116
7.2.1 Description 116
7.2.2 Modèle 117
7.2.3 Objectifs 119
7.3 Le problème de l'entrepôt de stockage 119
7.3.1 Description 119
7.3.2 Modèle 120
7.3.3 Objectifs 121
7.4 Conclusion 121

« Les États-Unis ne sont pas un modèle parfait du monde. »

Randall Munroe

7.1 Introduction

Afin d'évaluer nos approches pour faire de la formation dynamique d'équipes sous incertitude, et comme il n'existe pas d'autres travaux dans ce domaine, il nous faut introduire nos propres jeux de tests. Les benchmarks couramment utilisés pour évaluer les DEC-POMDPs ne constituent pas des modèles suffisants pour représenter des situations où les entrées et sorties des agents peuvent avoir des effets bénéfiques sur la résolution de la tâche. Dans ce chapitre, on présente deux problèmes que nous avons mis au point et qui vont nous permettre d'évaluer les différents algorithmes présentés dans les chapitres 5 et 6, mais également de mettre en lumière certaines propriétés intéressantes que nous avons étudiées dans la section 6.3. Pour chacun de nos jeux de tests, on introduit également des **états buts**, qui correspondent aux états où la tâche est résolue.

Comme les OPEN-DEC-POMDPs à contrôle total et les TEAM-POMDPs sont des modèles équivalents, nous allons supposer, sauf mention contraire, que toutes les expérimentations sont faites dans le contexte d'un TEAM-POMDP. Chaque agent i dispose d'une action *Quitter* et d'une observation *Rien*.

7.2 Le problème de la zone sinistrée

Ce premier scénario prend place dans un environnement dynamique dont nous avons déjà parlé dans cette thèse. Il s'agit d'une extension du benchmark *Fire Fighting*, extrait du domaine de la *RoboCup Rescue* (une compétition de robots de sauvetage) [Kitano and Tadokoro, 2001], régulièrement utilisé pour évaluer les performances des algorithmes de résolution de DEC-POMDPs [Oliehoek et al., 2008]. Nous allons nous référer à ce benchmark sous le nom de *Problème de la zone sinistrée*. https://xkcd.com/2128/

7.2.1 Description

Suite à une catastrophe naturelle de grande ampleur près d'une zone habitée, une large partie des infrastructures de la ville a été touchée et des feux se sont déclenchés à plusieurs endroits, menaçant de se propager aux bâtiments proches. La dangerosité de la situation et la difficulté d'accès au terrain empêchent le déploiement d'une équipe de professionnels. À la place, afin de faire face à la menace d'un incendie généralisé dans toute la ville, une brigade de robots autonomes est mobilisée. Cette brigade est composée de robots pompiers, équipés de réservoirs d'eau, et de robots bulldozers, chargés de déblayer les zones saccagées afin que les pompiers puissent éteindre les feux. Ces robots doivent coopérer dans un environnement complexe, dynamique et partiellement inconnu afin d'accomplir leur mission.

Comme les robots sont limités en termes de ressources (eau et énergie), et que chaque robot induit un coût de déploiement, utiliser la totalité des robots disponibles tout le temps n'est pas optimal. Lorsque les débris bloquent complètement l'accès aux bâtiments, les robots pompiers sont inefficaces. Inversement, lorsque le feu se propage dangereusement, envoyer plus de pompiers est nécessaire. Il convient ainsi d'adapter l'équipe à la situation sur place afin de ne pas mobiliser plus de ressources que nécessaire.

La figure 7.1 illustre le *Problème de la zone sinistrée*.

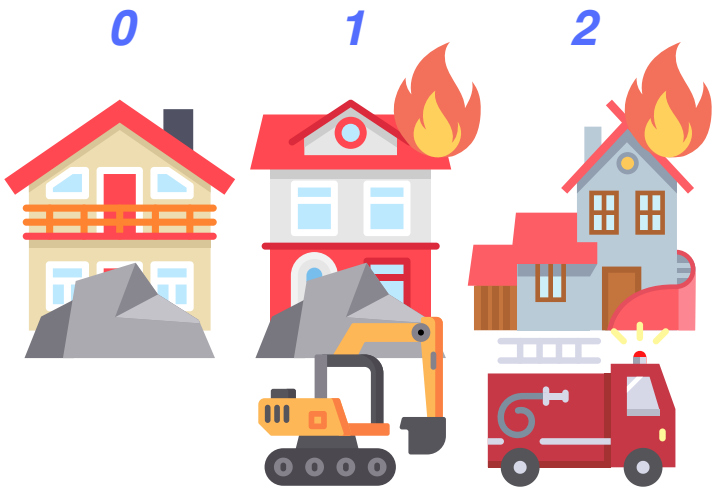


FIGURE 7.1 – Illustration du *Problème de la zone sinistrée*.

7.2.2 Modèle

On considère une population $\mathcal{N} = \{1,2,\dots,n\}$ de n agents dont l'objectif est d'éteindre un feu se propageant parmi m maisons $\mathcal{M} = \{1,2,\dots,m\}$.

Espace d'états. Un agent $i \in \mathcal{N}$ possède trois informations à chaque instant t.

1. Un **niveau de ressources** $d_i^t \in \mathbb{N}$ et une capacité maximale $D_i \in \mathbb{N}_*$, telle que $d_i^t \leqslant D_i, \forall t$.
2. Un **coût** $c_i \in \mathbb{R}_+$ pour faire partie de l'équipe opérationnelle.
3. Un **coefficient d'inefficacité** $w_i > 1$, un coefficient multiplicateur associé à son coût lorsque l'agent effectue une action différente de l'action *Quitter* et que $d_i^t = 0$.

Une maison $j \in \mathcal{M}$ possède quant à elle deux informations à chaque instant t.

1. Un **niveau de feu** $f_j^t \in \mathbb{N}$, décrivant à quel point la maison j est en train de brûler. Si $f_j^t = 0$, alors la maison ne brûle pas. Plus f_j^t est élevé et plus grave est la situation à cette maison. Il y a $F \geqslant 0$ niveaux de feu au maximum ($f_j^t < F, \forall t$).
2. Un **niveau de gravats** $g_j^t \in \mathbb{N}$, décrivant s'il y a des gravats qui empêchent l'accès la maison m. Si $g_j^t = 0$, alors l'accès est libre, les agents peuvent aller à cette maison et y combattre le feu. Sinon, si $g_j^t > 0$, alors la voie doit préalablement être dégagée pour pouvoir accéder à cette maison. Il y a $G \geqslant 0$ niveaux de gravats au maximum ($g_j^t < G, \forall t$).

Un état

$$s^t = \left(\left\{[f_1^t, g_1^t], \dots, [f_m^t, g_m^t]\right\}, \left\{d_1^t, \dots, d_n^t\right\}\right) \in \mathcal{S} \tag{7.1}$$

est un tuple de taille $n + m$ décrivant, à chaque étape de décision t, l'état du feu et des gravats à chaque maison ainsi que le niveau de ressources restantes de chaque agent. Initialement, le premier état s^0 est tiré aléatoirement selon une distribution de probabilité uniforme sur l'ensemble des états.

Afin d'éteindre le feu, la population $\mathcal{N}$ est composée de deux types d'agents, chaque type ayant ses propres caractéristiques :

1. les **pompiers** peuvent observer le feu à une maison et le combattre pour faire baisser le niveau de feu associé ;
2. les **bulldozers** peuvent observer les gravats à une maison et également faire baisser le niveau de gravats associé.

Actions. Pour chacun des deux types d'agents, l'ensemble d'actions est relativement similaire : un agent peut décider d'aller à une maison et, selon qu'il est pompier ou bulldozer, soit essayer d'éteindre le feu, soit dégager la voie des gravats qui en bloquent l'accès. Chaque agent possède donc m actions différentes, une pour chaque maison, en plus de l'action spéciale *Quitter* pour sortir de l'équipe opérationnelle. Un pompier, même s'il décide de se rendre à une maison $j \in \mathcal{M}$ telle que $f_j^t > 0$, ne pourra pas combattre le feu à cette maison si la voie est bloquée (c'est-à-dire si $g_j^t > 0$). Pour qu'un pompier $i \in \mathcal{N}$ puisse faire baisser le niveau de feu à une maison $j \in \mathcal{M}$, il faut donc que ce pompier ait un niveau de ressources suffisant ($d_i^t > 0$) et que la voie vers cette maison soit libre ($g_j^t = 0$). Supposons que les pompiers aient un niveau de ressources positif. Si une maison brûle et qu'aucun pompier n'y est présent, alors son niveau de feu va augmenter d'une unité avec une probabilité 0,8 si au moins une maison voisine brûle, et avec une probabilité 0,4 si aucune maison voisine ne brûle. Un pompier seul à une maison va abaisser le niveau de feu de cette maison d'une unité avec une probabilité 1 si aucune maison voisine ne brûle, et avec une probabilité 0,6 sinon. Finalement, deux ou plus de deux pompiers présents à une même maison vont toujours éteindre le feu complètement. Un pompier consomme de façon déterministe une unité de ressource par étape de décision chaque fois qu'il effectue une action $a_i \in \mathcal{A}_i$, jusqu'à un minimum de 0. De même, il récupère de façon déterministe une unité de ressource chaque fois qu'il effectue l'action *Quitter*, jusqu'à un maximum de D_i.

De leur côté, les bulldozers n'ont aucune influence sur l'évolution du feu, et ne peuvent donc ni faire baisser le niveau de feu à une maison, ni empêcher le feu de se propager aux maisons voisines. En revanche, chaque bulldozer présent à une maison fera toujours baisser le niveau de gravats à cette maison d'une unité s'il possède un niveau de ressources positif. Comme pour les pompiers, un bulldozer consomme une unité de ressource par action, et récupère une unité de ressource lorsqu'il n'est pas opérationnel.

Observations. Les agents font des observations différentes selon qu'ils sont pompiers ou bulldozers. Un pompier n'observe pas directement le niveau de feu, mais uniquement si oui ou non il y a du feu. Si $f_j^t = 0$, la probabilité que l'agent observe du feu est 0,2. Si $f_j^t = 1$, la probabilité qu'il observe du feu est 0,5. Enfin, si $f_j^t > 1$, alors l'agent ne peut pas se tromper et observe toujours du feu avec une probabilité 1. Les bulldozers quant à eux peuvent observer si oui ou non une maison est bloquée par des gravats. Si $g_j^t = 0$, la probabilité que l'agent observe des gravats devant la maison est 0,2. Sinon, si $g_j^t > 0$, la probabilité que l'agent observe des gravats est 0,8.

Enfin, chaque agent peut observer si oui ou non il lui reste des ressources. Si un agent i a un niveau de ressources positif $d_i^t > 0$, il observe qu'il possède bien des ressources avec une probabilité 1.

Récompenses. Une **équipe opérationnelle** $C^t = (n_P^t, n_B^t)$, est un sous-ensemble des agents de $\mathcal{N}$, composé de n_P^t pompiers et de n_B^t bulldozers (tel que $n_P^t + n_B^t \leqslant n$). La récompense générée par l'équipe C^t à l'instant t dans l'état s^t est

$$R_{C^t}(s_t) = -\Big(\sum_{j \in \mathcal{M}} f_j^t + \sum_{i \in C^t} \hat{c}_i^t\Big), \tag{7.2}$$

où $\hat{c}_i^t$ est le **coût ajusté**, défini par :

$$\hat{c}_i^t = \begin{cases} w_i c_i & \text{si } d_i^t = 0 \\ c_i & \text{sinon.} \end{cases} \tag{7.3}$$

Un **état but** s^* est défini comme un état dans lequel le niveau de feu de chaque maison est réduit à zéro :

$$\sum_{j \in \mathcal{M}} f_j^t = 0. \tag{7.4}$$

On note B^* l'ensemble des états buts.

7.2.3 Objectifs

Le *Problème de la zone sinistrée* prend place dans un environnement dynamique, où l'état du système est amené à évoluer de lui-même si les agents n'agissent pas. Ce benchmark permet de pénaliser indirectement les séquences d'équipes composées uniquement de pompiers (puisque, sans bulldozers, celles-ci n'auront pas accès au feu) sans explicitement pénaliser la présence de gravats devant les maisons. En effet, la totalité du feu peut être éteinte même s'il reste des maisons bloquées par des gravats. L'objectif des agents est d'éteindre le feu et non pas de déblayer les gravats devant chaque maison. Ce benchmark permet également d'illustrer la notion d'agents nécessaires, telle que présentée dans la définition 6.3.1.

Lorsque le coût des agents est nul ($c_i = 0, \forall i \in \mathcal{N}$) et qu'ils possèdent des ressources illimitées ($D_i = \infty, \forall i \in \mathcal{N}$), alors la meilleure équipe à employer à tout instant est toujours la plus grande équipe, celle composée de tous les agents de $\mathcal{N}$. Il s'agit alors du cas sur-additif (définition 6.3.6). En revanche, suivant le coût des agents, ainsi que du coefficient d'inefficacité, il peut être intéressant de ne pas toujours faire appel à la plus grande équipe.

Ce benchmark, comme nous l'avons dit, est une extension du *Fire Fighting* [Oliehoek et al., 2008], où les maisons en flammes ne sont pas bloquées par des gravats et où il n'est donc pas nécessaire de faire appel à des bulldozers : les seuls agents sont des pompiers et leur ressources sont illimitées. Dans nos expérimentations, nous nous référerons au terme *Fire Fighting* pour désigner ce scénario en environnement fermé où les agents ne peuvent pas entrer et sortir du système.

Le *Problème de la zone sinistrée* est un domaine très modulable, où l'on peut spécifier librement le nombre d'agents disponibles, le nombre de maisons à protéger, les différents niveaux de feu et de gravats, etc. Dans la totalité de nos expérimentations, sauf mention contraire, nous fixons $F = 3$, qui indique trois niveaux de feu possibles : 0 (pas de feu), 1 (feu modéré) et 2 (feu puissant). Lorsque cela s'applique, nous fixons également $G = 2$, qui indique alors deux niveaux de gravats possibles : 0 (voie barrée) ou 1 (voie accessible). En général, on considérera qu'il y a toujours trois maisons : $\mathcal{M} = \{0,1,2\}$.

7.3 Le problème de l'entrepôt de stockage

Ce second scénario prend place dans un environnement statique. Nous allons nous référer à ce benchmark sous le nom de *Problème de l'entrepôt de stockage*.

7.3.1 Description

Évoluant sur le marché très concurrentiel de la vente en ligne, une entreprise a décidé de se moderniser pour augmenter ses bénéfices. Pour cela, elle a investi dans une flottille de n drones programmables afin d'occuper des tâches de logistique à l'intérieur de son entrepôt de stockage. Chaque matin, l'entreprise reçoit de nombreux produits de ses fournisseurs, produits qu'elle va devoir expédier à ses clients. À la réception des différents colis, un sous-ensemble – une équipe –

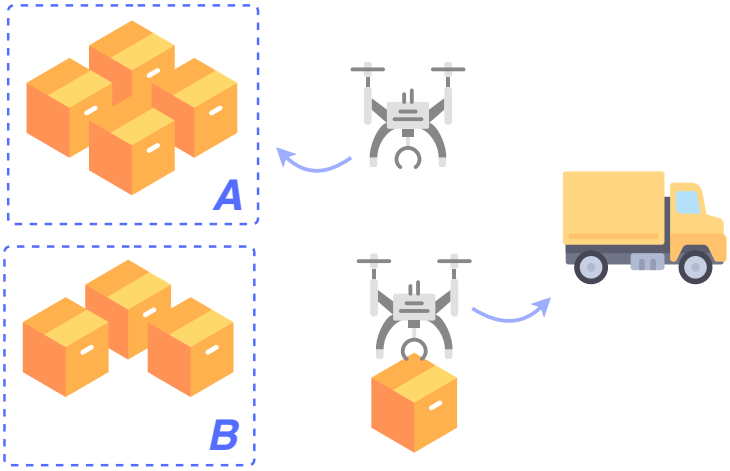


FIGURE 7.2 – Illustration du *Problème de l'entrepôt de stockage.*

de drones est chargé de faire transiter les colis depuis l'intérieur de l'entrepôt jusqu'au camion de livraison.

Employer trop peu de drones dans l'équipe va induire des temps d'attente trop longs pour le camion de livraison, ce qui est contre-productif pour l'entreprise qui cherche naturellement à maximiser le nombre de colis expédiés par unité de temps. De façon similaire, employer la totalité des n drones peut s'avérer coûteux suivant la quantité de travail réelle à effectuer. De plus, les drones peuvent interférer les uns avec les autres et provoquer des collisions, conduisant ainsi également à des coûts supplémentaires et de mauvaises performances. L'entreprise souhaite donc gérer le nombre de drones dans l'équipe opérationnelle afin de s'adapter au nombre de colis présents dans l'entrepôt.

La figure 7.2 illustre le *Problème de l'entrepôt de stockage.*

7.3.2 Modèle

On considère une population $\mathcal{N} = \{1,2,\ldots,n\}$ de n agents.

Espace d'états. Tous les agents sont identiques. Ce sont des robots autonomes travaillant dans un entrepôt où ils doivent transporter des colis depuis l'intérieur de l'entrepôt jusqu'à un camion de livraison qui attend à l'extérieur. L'entrepôt est divisé en deux zones de stockage, la zone A et la zone B. Au début de la journée, il y a jusqu'à un nombre $P > 0$ de colis en provenance des fournisseurs qui arrivent dans l'entrepôt et qui sont aléatoirement répartis entre les zones A et B.

Un état

$$s^t = (P_A^t, P_B^t) \in \mathcal{S} \tag{7.5}$$

décrit le nombre de colis P_A^t dans la zone A et P_B^t dans la zone B en attente d'être chargés dans le camion de livraison. À $t = 0$, on a $P_A^0 + P_B^0 \leqslant P$.

Actions. Un agent peut décider d'aller dans l'une ou l'autre des deux zones de l'entrepôt, prendre un colis et l'emmener jusqu'au camion de livraison. Chaque agent possède ainsi deux actions pour résumer ce processus, une action pour chaque zone de l'entrepôt, A et B. Comme il s'agit de robots bien conçus, la probabilité qu'un agent effectue une action avec succès (chargement d'un colis depuis une zone de l'entrepôt vers le camion de livraison) est 0,95. S'il réussit

son action, alors le nombre de colis en attente dans la zone concernée diminue d'une unité. Si l'agent échoue, alors le colis reste dans la zone de stockage. De plus, puisque la navigation dans l'entrepôt peut s'avérer difficile, avoir trop d'agents dans une même zone de l'entrepôt induit une baisse des performances. La probabilité qu'un agent réussisse son action diminue d'un montant fixe $0 < \lambda < 1$, appelé **facteur de décroissance**, pour chaque agent supplémentaire présent dans la zone avec lui. Par exemple, si trois agents sont présents dans une zone, la probabilité de réussir à charger un colis depuis cette zone est $max(0; 0{,}95 - 3\lambda)$.

Observations. Un agent ne peut pas compter le nombre de colis dans l'entrepôt. Il peut juste savoir s'il y a des colis dans une zone ou pas. Après avoir effectué une action, l'agent observe avec une probabilité certaine s'il reste ou non des colis dans la zone.

Récompenses. Une **équipe opérationnelle** $C^t \subseteq \mathcal{N}$ est un sous-ensemble des agents. La récompense générée par l'équipe C^t à un instant t dans l'état s^t correspond au nombre de colis correctement chargés dans le camion de livraison :

$$R_{C^t}(s^{t-1},s^t) = (P_A^{t-1} - P_A^t) + (P_B^{t-1} - P_B^t). \tag{7.6}$$

Un **état but** est défini comme un état dans lequel tous les colis ont été chargés dans le camion de livraison lorsque $t = h$, c'est-à-dire si $P_A^h = P_B^h = 0$. On note B^* l'ensemble des états buts (ici B^* ne contient qu'un seul état).

7.3.3 Objectifs

Contrairement au *Problème de la zone sinistrée*, le *Problème de l'entrepôt de stockage* prend place dans un environnement statique : le nombre de colis à l'intérieur de l'entrepôt ne va jamais évoluer si les agents ne les transportent pas vers le camion. En cela, il s'agit d'un problème plus simple que le *Problème de la zone sinistrée*, qui va nous permettre de tester nos algorithmes avec un plus grand nombre d'agents. On peut remarquer qu'il serait aisément possible de rendre ce problème dynamique plutôt que statique, par exemple en faisant arriver des colis dans chaque zone de l'entrepôt au fur et à mesure de l'exécution de la tâche.

De plus, ce benchmark permet de pénaliser des équipes d'agents trop grandes sans ajouter un coût "artificiel" aux agents : ce sont les performances des agents sur la résolution de la tâche qui sont impactées par la composition de l'équipe (contrairement au *Problème de la zone sinistrée*, où le coût des agents est directement inclus dans la fonction de récompense).

7.4 Conclusion

Ce bref chapitre nous aura permis d'introduire les deux benchmarks que nous allons utiliser pour évaluer les algorithmes présentés dans cette thèse. Les deux problèmes développés représentent des situations crédibles, plus proches d'applications réelles que les benchmarks régulièrement utilisés pour l'évaluation de DEC-POMDPs. Ils vont nous permettre notamment de travailler avec un grand nombre d'agents possible, et nous allons voir qu'utiliser tous les agents à disposition n'est pas toujours la meilleure solution.

Le chapitre suivant présente nos résultats.

Chapitre 8

Résultats

Sommaire

8.1 Introduction 124
8.2 Évaluation de l'algorithme BRD-POS-UCT 125
8.2.1 Comparaison avec les environnements fermés 126
8.2.2 Gestion de la taille de l'équipe opérationnelle 127
8.2.3 Influence de l'état de présence 128
8.3 Évaluation de l'algorithme MELO 128
8.3.1 Mise en évidence de la sous-modularité 129
8.3.2 Valeurs non-monotones 131
8.3.3 Pertinence des classements Elo 133
8.3.4 Stratégies de descente 138
8.3.5 Définitions de nœuds extensibles 139
8.3.6 Matchs des meilleures équipes 141
8.4 Conclusion 141

« Des résultats ! Voyons, j'ai obtenu beaucoup de résultats. Je connais quelques milliers de choses qui ne fonctionneront pas. »

Thomas Edison

8.1 Introduction

Dans ce chapitre, nous rapportons les différents résultats que nous avons obtenus et tels que nous les avons publiés. Dans ces expérimentations, nous n'allons pas nous comparer à des méthodes ou algorithmes existants, puisqu'il n'existe pas de tels algorithmes *dédiés* pour le problème TARS de planification et de formation dynamique d'équipes sous incertitude – si ce n'est en étudiant le problème sous l'angle d'un DEC-POMDP classique. On cherche plutôt à mettre en lumière quelques caractéristiques des TEAM-POMDPs, comme la propriété de sur-additivité par exemple, telle que donnée en définition 6.3.6.

Critères de performance. Nous donnons ici trois critères nous permettant d'évaluer les algorithmes présentés dans cette thèse.

1. **Valeur.** C'est le critère le plus important, celui qui permet de mesurer la valeur des politiques jointes calculées par nos algorithmes. Lorsque cela est possible, on peut ainsi comparer nos valeurs aux valeurs optimales théoriques. Plus la valeur de la politique jointe calculée est proche de la valeur optimale, plus les agents agissent de manière optimale. Un TEAM-POMDP est *résolu* si l'on trouve la politique jointe générant une valeur optimale.
2. **Taux de succès.** Ce critère correspond à la capacité d'une politique jointe à effectivement atteindre un état but lorsque l'horizon de planification est atteint. Étant donné que l'état initial de chaque problème est tiré aléatoirement selon la distribution de probabilité b^0 du TEAM-POMDP, il peut arriver que certaines instances du problème soient plus simples que d'autres (par exemple, si, à $t = 0$, peu de maisons sont en feu ou s'il n'y a pas de gravats qui bloquent les accès). Une politique jointe, si elle n'est pas optimale, ne pourra donc pas toujours atteindre un état but. Le **taux de succès** Λ_π d'une politique jointe π évaluée n fois est :
$$\Lambda_\pi = \sum_{i=0}^{n} \frac{1}{n} S(\pi) \tag{8.1}$$
où
$$S(\pi) = \begin{cases} 1 \text{ si } s_\pi^h \in B^* \\ 0 \text{ sinon} \end{cases} \tag{8.2}$$
et où s_π^h est l'état final lorsque la politique jointe π a été exécutée jusqu'à l'instant h.
3. **Efficacité.** Enfin, ce troisième critère évalue la pertinence d'un agent à faire partie de l'équipe opérationnelle. Par exemple, un pompier qui n'a plus de ressources en eau disponibles a une efficacité nulle s'il reste dans l'équipe opérationnelle et que des maisons sont toujours en flammes. Pareillement, l'efficacité d'un agent est nulle s'il ne parvient pas à transporter un colis depuis une zone de l'entrepôt vers le camion (soit parce qu'il n'y a pas de colis dans sa zone, soit parce que son action échoue). L'**efficacité** Υ_π d'une politique jointe π est définie comme étant le nombre moyen d'agents **efficaces** présents dans l'équipe opérationnelle à chaque étape de décision. Un agent $i \in \mathcal{N}$ est **efficace** à l'instant t si :
   - dans le *Problème de la zone sinistrée*, si c'est un pompier, il effectue l'action d'aller à une maison $j \in \mathcal{M}$ telle que $f_j^t > 0$, $g_j^t = 0$ et si $d_i^t > 0$;
   - dans le *Problème de la zone sinistrée*, si c'est un bulldozer, il effectue l'action d'aller à une maison $j \in \mathcal{M}$ telle que $g_j^t > 0$ et si $d_i^t > 0$;
   - dans le *Problème de l'entrepôt de stockage*, s'il choisit la zone $X \in \{A,B\}$, $P_X^{t+1} = P_X^t - 1$.

Agent 0		Agent 1	
Historique	→ Action	Historique	→ Action
∅	1	∅	2
0	1	0	0
0-0	1	0-0	0
0-0-0	1	0-0-0	2
0-0-1	1	0-0-1	2
0-1	1	0-1	0
0-1-0	1	0-1-0	0
0-1-1	1	0-1-1	0
1	1	1	0
1-0	1	1-0	0
1-0-0	1	1-0-0	2
1-0-1	1	1-0-1	2
1-1	1	1-1	0
1-1-0	2	1-1-0	2
1-1-1	2	1-1-1	2

Tableau 8.1 – Exemple de politiques individuelles à horizon 4 calculées par l'algorithme BRD-POS-UCT pour le *Fire Fighting* ($n = 2$,$m = 3$,$F = 3$,$G = 0$). **Observations** : 0 = Feu observé, 1 = Pas de feu observé. **Actions** : 0 = Aller à la première maison, 1 = Aller à la deuxième maison, 2 = Aller à la troisième maison.

Algorithmes évalués. Les algorithmes que nous évaluons sont ceux présentés dans cette thèse.

1. **BRD-POS-UCT.** Algorithme de calcul de meilleures réponses, où chaque meilleure réponse est construite avec POS-UCT, une adaptation de l'algorithme UCT aux environnements partiellement observables et stochastiques.
2. **MELO.** Algorithme POS-UCT multi-agents avec système de classement Elo, où les équipes sont mises en compétition afin de trouver rapidement les plus appropriées à chaque historique d'observations.
3. **MA-POS-UCT.** Algorithme POS-UCT adapté aux cas multi-agents. Il s'agit d'une version de l'algorithme MELO où les classements Elo des équipes ne sont pas pris en compte. Nous en parlerons plus en détail dans la section 8.3.

8.2 Évaluation de l'algorithme BRD-POS-UCT

Dans cette section, nous évaluons les performances et analysons les caractéristiques de l'algorithme BRD-POS-UCT, qui calcule des politiques jointes séparables en se basant sur l'algorithme de recherche de meilleures réponses BRD, et où chaque meilleure réponse est calculée avec l'algorithme POS-UCT.

La figure 8.1 donne un exemple de politique jointe séparable. Elle est composée de quatre colonnes : la première liste les historiques d'observations du premier agent, la deuxième ses actions associées, la troisième les historiques d'observations du second agent et la dernière ses actions associées. Chaque ligne, pour chaque agent, correspond à une **règle de décision**, et la

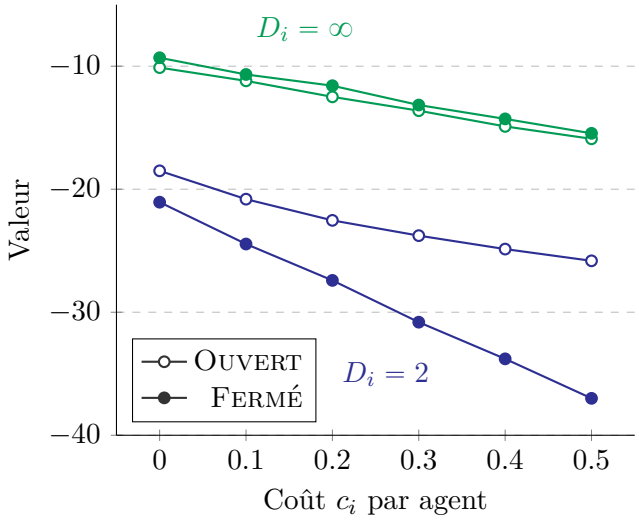


FIGURE 8.1 – Valeurs des politiques jointes calculées à horizon 6 avec l'algorithme BRD-POS-UCT pour le *Problème de la zone sinistrée* (nœuds vides) et le *Fire Fighting* (nœuds pleins) pour différents coûts par agent ($n = 2$,$m = 3$,$c_i \in [0{,}0.5]$,$w_i = 3$,$F = 3$,$G = 0$). Sur les deux courbes du haut, les agents ont des ressources illimitées ($D_i = \infty$). Pour les deux courbes du bas, les agents sont limités à deux unités de ressources ($D_i = 2$).

liste de ses règles de décision constituent sa politique individuelle (qui pourrait tout aussi bien être représentée sous forme arborescente). Par exemple, pour l'agent 0, la règle de décision

$$1\text{-}1\text{-}1 \longrightarrow 2$$

indique que le premier agent doit se rendre à la troisième maison (action 2, les maisons étant indexées à partir de 0) s'il observe trois fois de suite qu'il n'y a aucun feu (observation 1) à la deuxième maison. Ici, on remarque que le premier agent tend à rester (presque) toujours à la deuxième maison, tandis que le second agent alterne entre la première et la troisième maison. Il est en effet important de bien contrôler la seconde maison puisqu'il s'agit de la seule qui possède deux maisons voisines, ce qui la rend plus vulnérable à la propagation du feu.

8.2.1 Comparaison avec les environnements fermés

Pour commencer, nous cherchons à montrer les avantages d'utiliser un modèle ouvert plutôt qu'un modèle fermé dès que les agents ont des raisons d'entrer et de sortir de l'équipe opérationnelle durant l'exécution du processus. Deux facteurs peuvent pousser les agents à ne pas faire partie de l'équipe opérationnelle : leur coût et le manque de ressources.

La figure 8.1 donne les valeurs de politiques jointes calculées avec BRD-POS-UCT dans le *Problème de la zone sinistrée*, en fonction du coût par agent. Dans ce scénario-là, on ne considère aucun gravats devant les maisons ($G = 0$) et uniquement des équipes de pompiers, avec un maximum de $n = 2$ pompiers par équipe. En faisait varier le paramètre de coût, la figure montre que, si l'on considère des ressources limitées et des coûts élevés, utiliser un TEAM-POMDP nous permet de calculer de meilleures politiques jointes qu'avec un modèle fermé comme un DEC-POMDP (ici, avec le benchmark *Fire Fighting*) où les agents ne peuvent pas sortir de l'équipe pour récupérer des ressources – ressources qui finiront par s'épuiser.

Dans le cas où la quantité de ressources autorisée est illimitée ($D_i = \infty$), alors il est toujours optimal d'utiliser la totalité des pompiers, particulièrement lorsque le coût des agents est nul. On

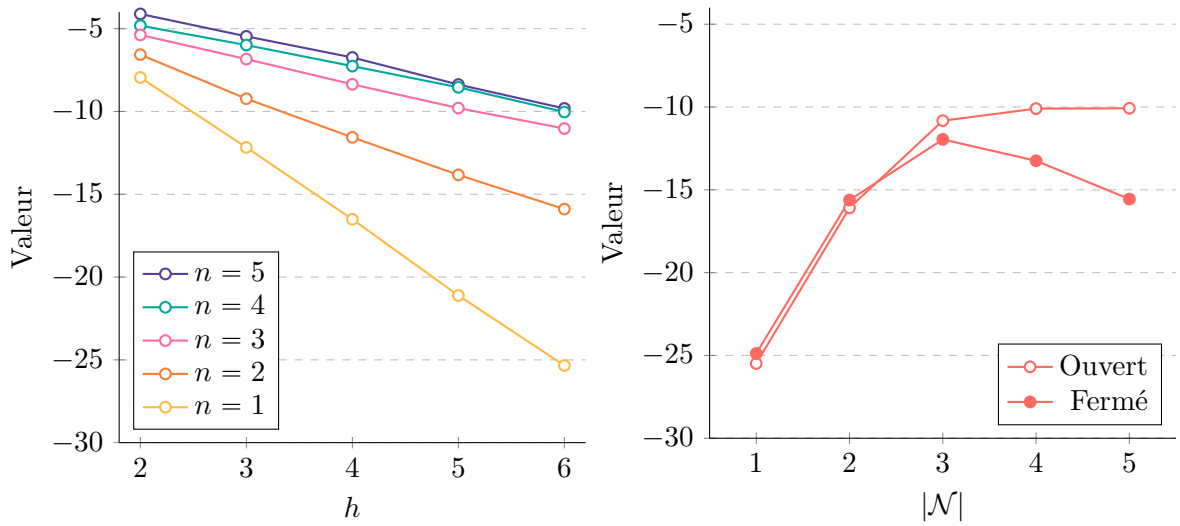


FIGURE 8.2 – Valeur des politiques jointes calculées avec l'algorithme BRD-POS-UCT pour le *Problème de la zone sinistrée* avec différents nombres de pompiers ayant des ressources illimitées ($n \in \{1,2,3,4,5\}$,$m = 3$,$c_i = 0.5$,$D_i = \infty$,$F = 3$,$G = 0$). **Gauche :** Pour différents horizons de planification h. **Droite :** À horizon 6, pour des populations de tailles $|\mathcal{N}|$ différentes, pour des systèmes à la fois ouverts et fermés.

remarque que la courbe pour les systèmes ouverts (—○—) est légèrement inférieure mais néanmoins très proche de la courbe pour les systèmes fermés (—●—). Cela est dû au fait que l'algorithme BRD-POS-UCT recherche dans l'espace des politiques localement optimales seulement avec des simulations de Monte-Carlo aléatoires. Comme notre l'algorithme teste toutes les actions possibles (en particulier, l'action *Quitter*) au moins une fois à chaque nœud visité, il peut arriver que des actions sous-optimales génèrent tout de même de bonnes récompenses, et des actions optimales de moins bonnes récompenses. Une politique individuelle peut ainsi suggérer à un agent de quitter l'équipe opérationnelle, ce qui n'est pas utile quand il n'a pas de ressources à récupérer.

Dans le cas où la quantité de ressources autorisée est limitée ($D_i = 2$), considérer des systèmes ouverts permet naturellement d'amortir la baisse de performance qu'induit l'augmentation des coûts des agents. La baisse de qualité des politiques jointes calculées pour un système ouvert (—○—) est de seulement **28,4%** pour le passage d'un coût de 0 à un coût de 0,5, tandis que la baisse, pour un système fermé (—●—), est de **43,1%** pour le passage d'un coût de 0 à un coût de 0,5.

8.2.2 Gestion de la taille de l'équipe opérationnelle

Nous voulons maintenant montrer l'influence du nombre d'agents dans un système ouvert. Nous nous attendons à mettre en lumière le phénomène suivant : considérer des équipes opérationnelles toujours plus grandes n'est utile que jusqu'à un certain point, point à partir duquel ajouter de nouveaux agents dans l'équipe opérationnelle ne va qu'entraîner des redondances, voire des baisses de performance. De plus, nous souhaitons montrer que l'algorithme BRD-POS-UCT est capable de contrôler la taille de l'équipe opérationnelle afin de garantir un bon équilibre entre le coût des agents et leur utilité dans la résolution de la tâche.

Nos résultats sont présentés sur la figure 8.2. On considère le problème du *Fire Fighting*

où chaque pompier possède une quantité illimitée de ressources ($D_i = \infty$) et a un coût fixe ($c_i = 0{,}5$). La plus grande équipe opérationnelle devrait fournir les meilleurs résultats, car plus le nombre de pompiers est grand et plus vite le feu est éteint. Il est important de remarquer que résoudre le problème du *Fire Figthing* avec cinq agents et seulement trois maisons et trois niveaux de feu est facile : à $t = 0$, il suffit de répartir deux agents sur deux maisons différentes et un dernier agent sur la dernière maison, puis à $t = 1$ de conserver au moins un agent sur la dernière maison, pour que le feu soit complètement éteint, peu importe la répartition initiale du feu sur les maisons. Avec quatre ou trois pompiers seulement, la tâche est plus ardue, en ce sens qu'il faudra parfois plus de deux étapes de décision pour éteindre le feu.

À gauche sur la figure 8.2, les valeurs des politiques jointes calculées sont sensiblement identiques lorsque la population est composée de trois (—○— $n = 3$), quatre (—○— $n = 4$) ou cinq (—○— $n = 5$) pompiers, mais bien plus faibles pour des populations de seulement un (—○— $n = 1$) ou deux (—○— $n = 2$) pompiers. Autrement dit, ce n'est pas vraiment moins efficace de limiter la taille de la population à trois agents plutôt qu'à cinq agents. Considérons le cas d'une population à cinq agents. Avoir cinq pompiers dans l'équipe opérationnelle est contre-productif : le coût à un instant t d'une équipe de cinq agents est de $0{,}5 * 5 = 2{,}5$, soit plus que la perte engendrée par une maison avec un niveau de feu puissant ($f_j = 2$ au maximum lorsque $F = 3$, pour toute maison $j \in \mathcal{M}$). Seulement trois ou quatre agents permettent d'obtenir des résultats presque aussi satisfaisants, même sur la totalité de l'horizon de planification. L'algorithme BRD-POS-UCT est ainsi **capable de limiter la taille de l'équipe opérationnelle** afin d'amoindrir les coûts des agents de la population, même quand la récompense sous-jacente du problème présente une structure clairement monotone, comme c'est le cas avec le *Fire Fighting*.

À droite sur la figure 8.2, ce phénomène est encore plus visible. La valeur d'une politique jointe où l'on force la présence de quatre ou cinq agents dans l'équipe opérationnelle (—●— Fermé) est inférieure à la valeur d'une politique partielle jointe où les agents peuvent rejoindre et quitter le système à tout moment (—○— Ouvert). Un comportement localement optimal émerge avec trois agents. Utiliser plus de trois agents n'est pas utile, et l'algorithme BRD-POS-UCT **permet de ne pas employer les agents inutiles**, même s'ils sont disponibles.

8.2.3 Influence de l'état de présence

L'état de présence, défini dans la section 5.4.3, mesure le taux de présence d'un agent dans l'équipe opérationnelle lorsqu'il suit une certaine politique individuelle. Plus son taux de présence est élevé, plus souvent il fera partie de l'équipe opérationnelle. Le faible taux de présence d'un agent peut être expliqué par plusieurs raisons, notamment si cet agent est redondant, s'il induit un coût supplémentaire non compensé, s'il provoque des interférences avec les autres agents, etc. Nous avons mené de multiples expérimentations pour tenter de mettre en lumière les bénéfices que cette heuristique de sélectionner des agents peut apporter.

Même en faisant varier le nombre d'agents, le nombre d'itérations dans l'algorithme, les coûts des agents ainsi que leurs ressources, les résultats que nous avons obtenus ne nous ont pas permis de conclure sur la pertinence de l'état de présence lors du processus de sélection des agents dans l'algorithme BRD – sinon que son influence est négligeable.

8.3 Évaluation de l'algorithme MELO

Nous étudions à présent les performances de l'algorithme MELO, la modification de l'algorithme POS-UCT au cas multi-agents, qui produit des politiques jointes non séparables en calculant et en maintenant le classement Elo des équipes dans les nœuds de l'arbre de recherche.

h	*Grande équipe*	*Sous-équipes*
2	17	**17**
3	270	**202**
4	2951	**732**
5	13119	**1609**
6	28766	**2072**

Tableau 8.2 – Nombre de règles de décision composant la politique jointe moyenne calculée avec l'algorithme MA-POS-UCT, à différents horizons de planification h, pour le *Problème de la zone sinistrée* ($n = 4$,$m = 3$,$F = 3$,$G = 2$,$c_i = 0$). La *Grande équipe* = (2,2) est l'équipe composée de deux pompiers et de deux bulldozers. Les *Sous-équipes* sont l'ensemble de toutes les équipes possibles : deux pompiers et deux bulldozers, deux pompiers et un bulldozer, etc.

Pour évaluer la pertinence de l'algorithme MELO, nous le comparons à une version simplifiée, où les classements Elo des équipes ne sont pas pris en compte. Rappelons-le, le classement Elo est utilisé dans l'algorithme MELO lors de la phase d'expansion de l'arbre pour ne pas avoir à essayer des équipes non prometteuses. Hormis cela, le classement Elo n'est utilisé nulle part ailleurs dans l'algorithme. Nous appelons **MA-POS-UCT** (*Multi-Agent Partially Observable and Stochastic Upper Confidence bound for Trees*) l'algorithme MELO où la phase d'expansion n'est pas faite en effectuant *une action jointe aléatoire de l'équipe avec le meilleur classement Elo*, mais en effectuant *une action jointe aléatoire d'une équipe aléatoire*. Dans ce nouvel algorithme, il n'est donc pas nécessaire de maintenir les classements Elo des équipes, ni de passer par les étapes de simulations parallèles et de matchs par paires. C'est la raison pour laquelle nous nous référons à cet algorithme sous le nom de MA-POS-UCT : il s'agit d'une version multi-agents de l'algorithme POS-UCT, où chaque nœud de l'arbre de recherche correspond à un historique d'observations partiel joint.

L'algorithme MA-POS-UCT va nous être utile, dans un premier temps, pour calculer rapidement des politiques jointes non séparables pour un TEAM-POMDP. Nous allons nous en servir pour illustrer certaines propriétés suivies par nos deux benchmarks.

8.3.1 Mise en évidence de la sous-modularité

Nous commençons par analyser l'algorithme MA-POS-UCT avec le *Problème de la zone sinistrée*. Rappelons-le, contrairement au *Fire Fighting*, les maisons peuvent ici être bloquées par des gravats et des bulldozers doivent être employés pour déblayer l'accès aux maisons. Dans tout ce qui va suivre, on fixe un coût nul à chaque agent ($c_i = 0, \forall i \in \mathcal{N}$).

Les résultats présentés sur les figures 8.2, 8.3 et 8.4 donnent un aperçu des avantages offerts par le modèle TEAM-POMDP dans le cas de ce benchmark. Deux cas sont comparés : le premier cas, *Grande équipe*, consiste à planifier en se servant uniquement de l'équipe composée de tous les agents de la population $\mathcal{N}$. Le second cas, *Sous-équipes*, consiste pour sa part à planifier en considérant toutes les équipes possibles de $\mathcal{C}$, y compris la plus grande équipe $\mathcal{N}$. Dans le premier cas, l'équipe opérationnelle est donc fixe, tandis que dans le second, l'équipe opérationnelle peut évoluer au cours de la résolution de la tâche.

Dans toutes nos expériences, nous avons considéré des populations $\mathcal{N}$ composées de deux pompiers et de deux bulldozers, soit $n = 4$ agents distincts. La *Grande équipe* désigne l'équipe comprenant tous les agents de $\mathcal{N}$. Elle est notée (2,2). Les *Sous-équipes* constituent l'ensemble $\mathcal{C} = 2^{\mathcal{N}}$ de toutes les équipes arrangeables avec les agents de $\mathcal{N}$. Il s'agit de l'ensemble noté

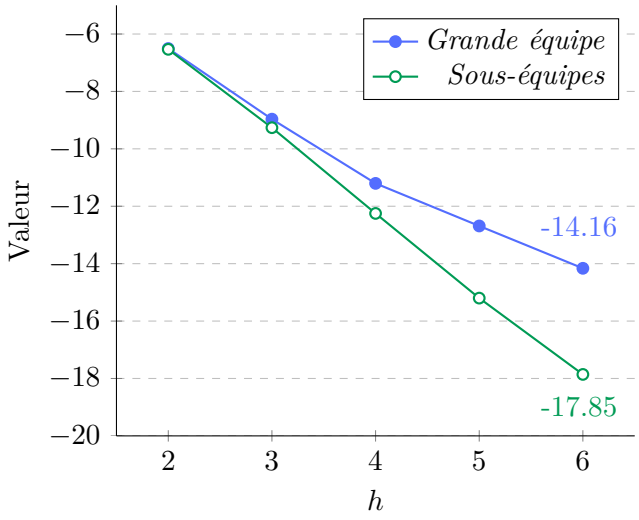


FIGURE 8.3 – Valeurs des politiques jointes calculées à divers horizons avec l'algorithme MA-POS-UCT pour la *Grande équipe* (—●—) et les *Sous-équipes* (—○—) pour le *Problème de la zone sinistrée* ($n = 4$,$m = 3$,$F = 3$,$G = 2$,$c_i = 0$).

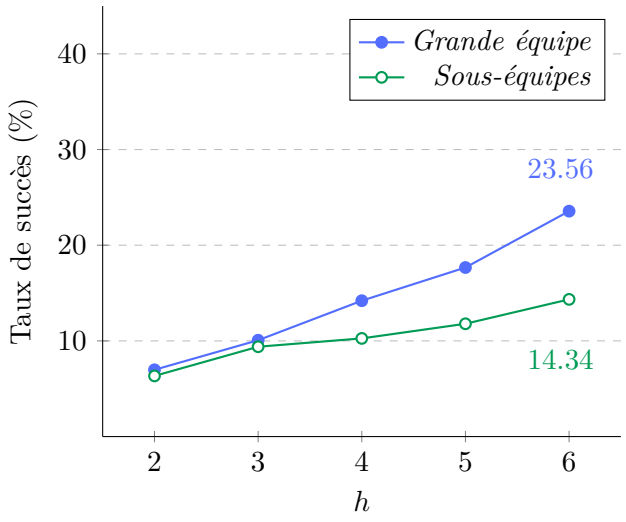


FIGURE 8.4 – Taux de succès des politiques jointes calculées à divers horizons avec l'algorithme MA-POS-UCT pour la *Grande équipe* (—●—) et les *Sous-équipes* (—○—) pour le *Problème de la zone sinistrée* ($n = 4$,$m = 3$,$F = 3$,$G = 2$,$c_i = 0$).

	Valeur	Taux de succès	Efficacité	Temps
Grande équipe	-13.7486	26%	30.88%	38s
Équipe (2,1)	-13.5857	28.71%	**37.98%**	**25s**

Tableau 8.3 – Valeurs des politiques, taux de succès, efficacités et temps de calcul pour calculer une politique jointe à l'horizon 6 avec l'algorithme MA-POS-UCT, pour le *Problème de la zone sinistrée* (n = 4,m = 3,F = 3,G = 2,c_i = 0), pour deux équipes distinctes : la *Grande équipe* (2,2) et une sous-équipe (2,1).

{(2,2),(2,1),(1,2),(2,0),(0,2),(1,1),(1,0),(0,1),(0,0)}. Le niveau de feu à chaque maison est $F = 3$ (trois niveaux de feu : 0, 1 et 2) et le niveau de gravats à chaque maison est $G = 2$ (deux niveaux de gravats : 0, 1). Sauf mention contraire, les politiques jointes sont calculées à horizon $h = 6$. Tous les résultats sont moyennés sur dix tests indépendants.

Le tableau 8.2 donne le nombre moyen de règles de décision calculées par l'algorithme MA-POS-UCT, à différents horizons, lorsque l'on planifie pour la *Grande équipe* uniquement ou pour les *Sous-équipes*. On remarque que, en utilisant les *Sous-équipes*, même lorsque l'on autorise la grande équipe (2,2), nous sommes capable de produire des politiques jointes d'ordres de grandeur bien inférieurs à ce que l'on produit pour la *Grande équipe* uniquement (à titre de comparaison, sur le tableau 8.1, la politique jointe calculée à horizon 4 comprend 30 règles de décision pour le *Fire Fighting*, 15 règles par agent). Cela s'explique par le fait que l'algorithme MA-POS-UCT est capable d'explorer des régions de l'espace de solutions là où les grandes équipes (celles qui font de grandes observations jointes) ne seront pas nécessairement utiles, ce qui permet un gain de **92.79%** en espace mémoire requis à horizon 6, et ce, sans avoir à maintenir de classement Elo pour les équipes. Cependant, ce gain d'espace pour les *Sous-équipes* se solde par une baisse de la valeur de la politique jointe, chutant de **-14.16** à **-17.85** à horizon 6, et un taux de succès chutant de **23.56%** à **14.34%** à horizon 6, comme montré sur les figures 8.3 et 8.4. Ces performances sont expliquées par le fait que notre algorithme n'atteint en général que des optimums locaux.

Une étude approfondie des politiques jointes calculées montre qu'il n'y a que peu de différence entre l'utilisation de la grande équipe (2,2) ou de l'équipe, plus petite, (2,1) (deux pompiers et un seul bulldozer). Cela fait écho à ce que l'on disait dans la section 6.3.3 : si un problème est sous-modulaire, alors on peut obtenir, avec des équipes plus petites, des récompenses proches de celles que l'on obtiendraient avec des équipes plus grandes. Le tableau 8.3 donne quelques points de comparaison entre la *Grande équipe* (2,2) et l'équipe (2,1). Bien que les valeurs des politiques jointes et les taux de succès ne montrent pas de différences statistiquement significatives, il est intéressant de noter que, en plus d'être plus rapide à calculer (une différence de **13s**), la politique jointe de l'équipe (2,1) a également une efficacité supérieure de **7.1%**. Ce résultat suggère qu'un seul bulldozer est suffisant pour planifier de façon optimale à horizon 6 dans le *Problème de la zone sinistrée*.

8.3.2 Valeurs non-monotones

Nous continuons notre étude de l'algorithme MA-POS-UCT avec une analyse du *Problème de l'entrepôt de stockage*. La figure 8.5 présente les valeurs des politiques jointes calculées pour des équipes de taille fixe, et la figure 8.6 présente les efficacités et taux de succès moyens des politiques jointes calculées pour des équipes de tailles variables.

Dans le contexte du *Problème de l'entrepôt de stockage*, le taux de succès est défini comme la probabilité que tous les colis soient chargés dans le camion de transport lorsque l'exécution

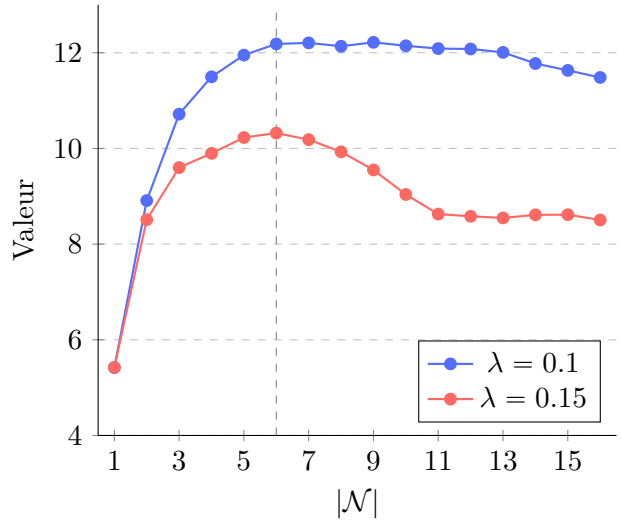


FIGURE 8.5 – Valeurs des politiques jointes calculées à horizon 6 pour des équipes de taille fixe $|\mathcal{N}|$ (c'est-à-dire $\mathcal{C} = \{\mathcal{N}\}$) pour le *Problème de l'entrepôt de stockage*, pour des facteurs de décroissance à $\lambda = 0.1$ (—●—) et $\lambda = 0.15$ (—●—).

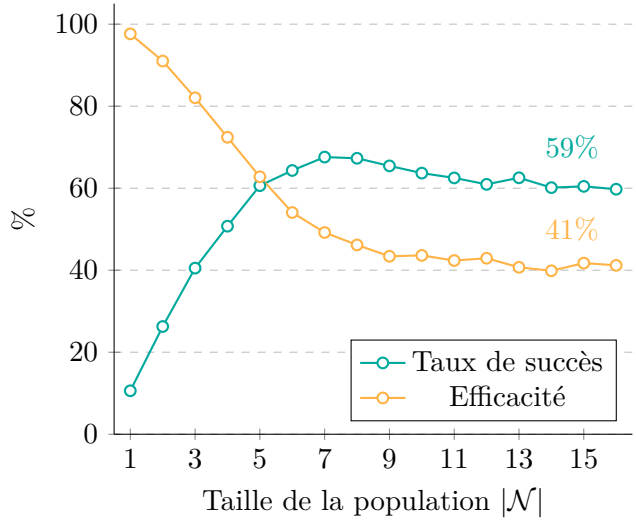


FIGURE 8.6 – Taux de succès (—○—) et efficacités (—○—) des politiques jointes calculées à horizon 6 pour des équipes de taille variable pour le *Problème de l'entrepôt de stockage*.

atteint l'horizon de planification (c'est-à-dire, lorsque le dernier état visité s^h est un état but) ; l'efficacité est quant à elle décroît lorsque les agents essaient mais échouent à charger des colis dans le camion de transport.

Nous avons effectué des tests avec au plus vingt colis ($P = 20$), initialement aléatoirement distribués entre les deux zones A et B. Cela signifie qu'il peut éventuellement y avoir moins de vingt colis dans l'entrepôt à $t = 0$. Les politiques sont calculées à horizon 6. Les résultats que nous avons obtenus mettent en lumière les caractéristiques particulières de ce benchmark. La figure 8.5 montre les valeurs des politiques jointes calculées pour des équipes de taille fixe, pour deux valeurs de facteur de décroissance différentes. Ici, le système est fermé : les agents ne peuvent pas quitter ou rejoindre l'équipe opérationnelle. Employer *trop peu* d'agents – bien qu'ils soient efficaces – ne permettra pas de charger tous les colis dans le camion de transport d'ici à la fin de temps imparti (c'est-à-dire d'ici à ce que l'horizon de planification soit atteint). Au contraire, employer *trop* d'agents va créer des interférences entre eux, les empêchant ainsi de résoudre leur tâche. Lorsque la probabilité que deux agents entrent en collision augmente peu avec le nombre d'agents (—● $\lambda = 0{,}1$), augmenter la taille de l'équipe est presque toujours profitable : les collisions potentielles seront toujours compensées par le nombre de colis chargés. En revanche, dès que la probabilité de collision augmente légèrement plus rapidement avec le nombre d'agents (—● $\lambda = 0{,}15$), cela n'est plus vrai : la concavité de la courbe de valeurs relativement à la taille de l'équipe suggère que le problème est **sous-additif** à partir d'un certain point (la ligne verticale grise sur la figure 8.5).

La figure 8.6 met en parallèle les taux de succès (—○—) et les efficacités (—○—) des politiques jointes calculées pour des équipes de tailles variables, dans un contexte ouvert où les agents peuvent donc entrer et sortir de l'équipe opérationnelle. L'axe des abscisses correspond à la population totale mais pas forcément à la taille de l'équipe opérationnelle. On remarque que l'efficacité est inversement corrélée au taux de succès. Un agent seul a un taux de succès quasi-nul mais une efficacité maximale, car, étant donné qu'il ne peut entrer en collision avec aucun autre agent, il ne peut (quasiment) pas échouer à charger un colis dans le camion de transport. De façon remarquable, on voit que ces deux indicateurs convergent, pour des populations de tailles supérieures à $|\mathcal{N}| = 8$, à **59%** pour le taux de succès, et **41%** pour l'efficacité. Le taux de succès reste stable et ne décroît pas pour des populations de taille 9 et plus car notre algorithme est capable de garder la taille de l'équipe opérationnelle à des valeurs qui lui permette de rester efficace. Un examen attentif des politiques jointes calculées montre en effet que les équipes de taille 7, 8 et 9 sont les équipes les plus utilisées.

8.3.3 Pertinence des classements Elo

Nous évaluons l'algorithme MELO à l'aide de trois scénarios différents du *Problème de la zone sinistrée*.

1. **Scénario 1**. Dans ce premier scénario, il n'y a pas de gravats qui bloquent l'accès aux maisons, c'est-à-dire, pour toute maison $j \in \mathcal{M}$ à l'instant t, $g_j^t = 0$. La population d'agents est composée de deux pompiers et d'aucun bulldozer ($n = 2$). L'ensemble des différentes équipes possibles est $\mathcal{C} = \{(0,0),(1,0),(2,0)\}$.
2. **Scénario 2**. Ce scénario correspond au cas normal du *Problème de la zone sinistrée*. Il peut y avoir un niveau de gravats positif devant chaque maison : c'est-à-dire, pour toute maison $j \in \mathcal{M}$ à l'instant t, $g_j^t \in \{0,1\}$. La population d'agents est composée de deux pompiers et de deux bulldozers ($n = 4$). L'ensemble des différentes équipes possibles est $\mathcal{C} = \{(0,0),(1,0),(2,0),(0,1),(1,1),(2,1),(0,2),(1,2)(2,2)\}$.

	Équipes	États	Actions jointes	Observations jointes
Scénario 1	3	27	13	7
Scénario 2	9	216	169	49
Scénario 3	9	216	169	49
Fire Fighting	1	27	9	4

Tableau 8.4 – Données relatives aux trois scénarios étudiés. Les informations du *Fire Fighting* sont données à titre comparatif.

	MELO	MA-POS-UCT
Scénario 1	**6,68**	10,28
Scénario 2	263,08	**183,87**
Scénario 3	244,34	**186,01**

Tableau 8.5 – Temps de calcul moyens (en secondes) pour calculer une politique jointe à horizon 6 avec les algorithmes MELO et MA-POS-UCT, pour chaque scénario, en 10000 itérations.

3. **Scénario 3**. Ce dernier scénario est similaire au Scénario 2, excepté que chaque agent présent dans l'équipe opérationnelle induit un coût fixe de 0,5, déduit de la récompense à chaque étape de décision où cet agent est dans l'équipe opérationnelle.

Le tableau 8.4 donne quelques informations supplémentaires concernant ces trois scénarios. Le Scénario 1 est intrinsèquement plus simple que les autres. Pour information, le tableau 8.5 donne quelques indicateurs en termes de temps de calculs pris par les algorithmes MELO et MA-POS-UCT pour résoudre chacun des trois scénarios. Sans surprise, l'algorithme MELO est plus lent pour les Scénarios 2 et 3 (qui sont équivalents en termes de complexité).

Les résultats de nos expériences concernant ces trois scénarios sont donnés sur les figures 8.7, 8.8, 8.9 et 8.10. Nous y comparons l'évolution des valeurs des politiques jointes calculées par les algorithmes MA-POS-UCT et MELO au fur et à mesure de leur construction durant les différentes itérations de ces algorithmes. Tous les résultats sont moyennés sur dix tests indépendants, et chaque algorithme effectue 10000 itérations, sauf mention contraire (une itération

h	Optimal	MELO
2	-4.383	-4.383
3	-5.736	-5.885
4	-6.578	-6.952
5	-7.069	-7.846
6	-7.175	-8.684

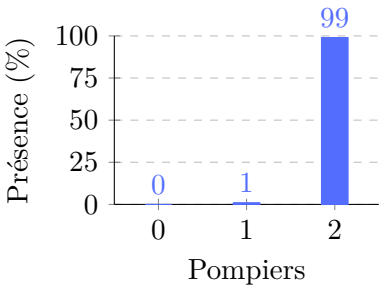


Tableau 8.6 – **Gauche :** Valeurs des politiques jointes calculées avec l'algorithme MELO pour le *Fire Fighting* ($n = 2$,$m = 3$,$F = 3$,$G = 0$,$c_i = 0$), comparées aux valeurs optimales, pour différents horizons de planification h. L'algorithme MELO travaille sur l'ensemble des équipes possibles (aucun, un ou deux pompiers). **Droite :** Taux d'utilisation de chaque équipe dans les politiques jointes calculées à horizon 6.

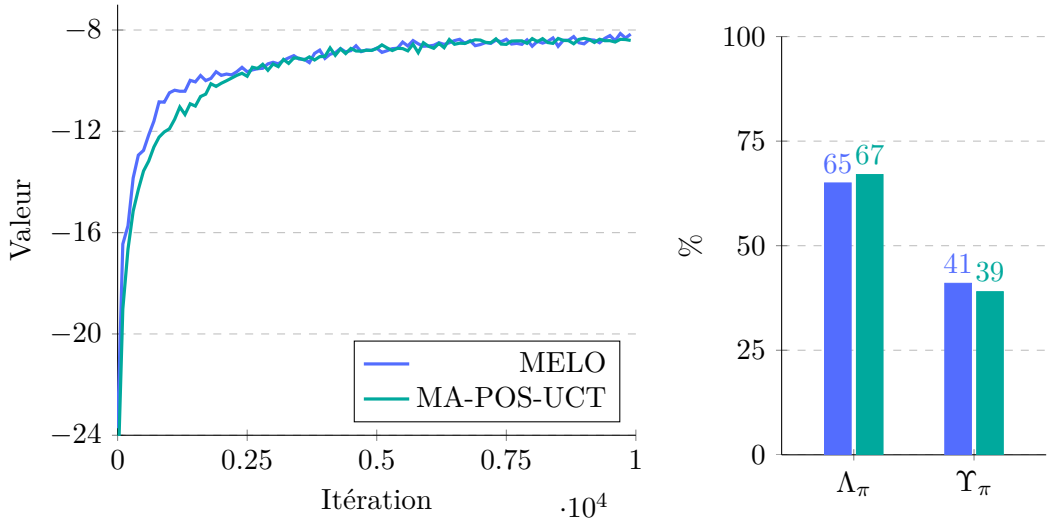


FIGURE 8.7 – **Scénario 1. Gauche :** Évolution des valeurs des politiques jointes calculées à horizon 6 au fur et à mesure de l'exécution des algorithmes MELO (——) et MA-POS-UCT (——) pour le Scénario 1 du *Problème de la zone sinistrée* ($G = 0$,$\mathcal{C} = \{(0,0),(1,0),(2,0)\}$). **Droite :** Taux de succès Λ_π et efficacités Υ_π moyens, en pourcentages.

correspondant aux différentes étapes de l'algorithme : descente de l'arbre, simulations parallèles, matchs par paires, expansion et rétro-propagation de la récompense).

Scénario 1. Tout d'abord, nous avons souhaité évaluer la pertinence de l'algorithme MELO pour résoudre des tâches plus simples qu'un TEAM-POMDP. En d'autres termes, si l'on se donne un ensemble d'équipes toutes inefficaces sauf une, notre algorithme est-il capable de détecter l'équipe efficace et de l'utiliser pour résoudre le problème – réduisant ainsi ce problème à de la planification pour un DEC-POMDP ? Le Scénario 1 est conçu précisément pour modéliser cette situation. Les deux équipes avec aucun pompier (0,0) et un seul pompier (1,0) sont clairement sous-optimales. La seule équipe qui est assurée de toujours réussir mieux que les deux autres est la grande équipe (2,0), composée de deux pompiers. C'est elle qui devrait donc toujours être utilisée par l'algorithme.

Le tableau 8.6 montre les valeurs moyennes des politiques jointes calculées par l'algorithme MELO à différents horizons dans le contexte du Scénario 1, et les compare aux valeurs optimales. Malgré le fait qu'il s'agisse d'une méthode approximative, l'algorithme MELO est capable de calculer des politiques jointes quasi-optimales. Cela est de moins en moins vrai lorsque l'horizon de planification augmente, mais les valeurs obtenues restent dans le même ordre de grandeur. Un examen attentif des politiques jointes calculées montre que l'algorithme MELO est capable de quasi-parfaitement détecter l'équipe (2,0) avec deux pompiers comme étant la seule équipe efficace, comme indiqué à droite du tableau 8.6. En moyenne, **99%** des règles de décision vont prescrire l'équipe avec deux pompiers. Le **1%** restant concernant l'équipe avec un seul pompier peut s'expliquer par le fait que si le feu est très faible, voire inexistant à $t = 0$, un seul pompier peut être suffisant. Bien que les résultats ne soient pas présentés dans le tableau 8.6, l'algorithme MA-POS-UCT obtient des scores similaires.

La figure 8.7 compare les valeurs des politiques jointes calculées par les algorithmes MELO et MA-POS-UCT dans le contexte du Scénario 1, où aucun gravats ne bloquent les maisons et où seuls des pompiers peuvent agir. Les deux algorithmes ont des comportements similaires :

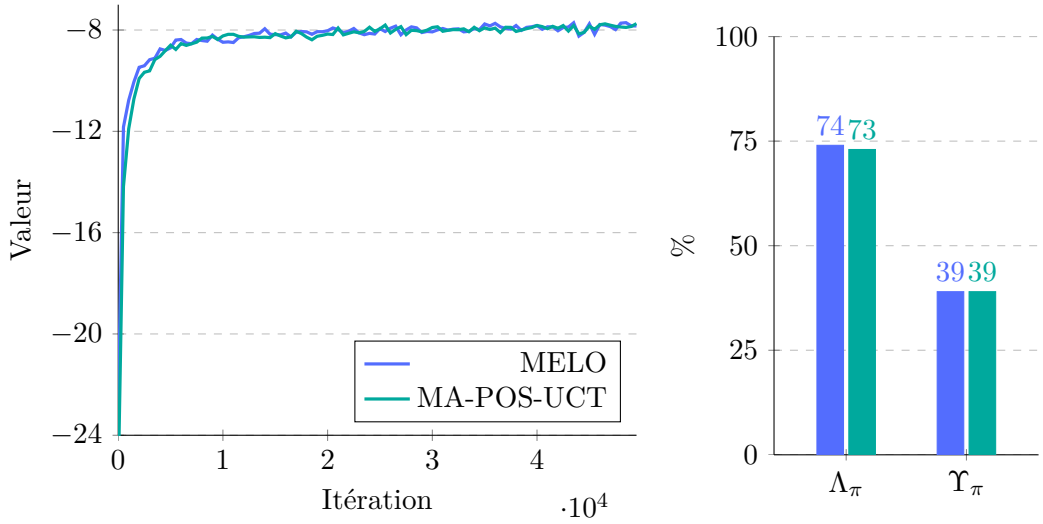


FIGURE 8.8 – **Scénario 1** – **Gauche :** Évolution des valeurs des politiques jointes calculées à horizon 6 au fur et à mesure de l'exécution des algorithmes MELO (——) et MA-POS-UCT (——) pour le Scénario 1 du *Problème de la zone sinistrée* ($G = 0$,$\mathcal{C} = \{(0,0),(1,0),(2,0)\}$), sur 50000 itérations. **Droite :** Taux de succès Λ_π et efficacités Υ_π moyens, en pourcentages.

après une rapide augmentation durant les premières itérations de l'algorithme, les valeurs des politiques convergent lentement vers environ **-8,6**, pour les deux algorithmes. On n'observe pas de différence notable entre les deux algorithmes. La figure montre que les taux de succès et les efficacités des agents sont semblables pour les deux algorithmes. Dans environ **66%** des cas, le feu est complètement éteint au bout de l'horizon de planification, et **40%** des pompiers sont efficaces en moyenne durant l'exécution. Notons bien que le taux de succès est correct sans être excellent : une politique jointe optimale de DEC-POMDP a un taux de succès de 100% pour le *Fire Fighting* à horizon 6. La valeur optimale qu'il est possible d'atteindre à cet horizon est -7,175. Notre algorithme est donc proche, sans être optimal toutefois, et la tendance montrée par les courbes indique que la valeur de la politique jointe calculée semble lentement converger vers la valeur optimale. À titre de comparaison, sur la figure 8.8, nous avons fait tourner nos algorithmes sur 50000 itérations. Une fois de plus, les deux algorithmes atteignent une valeur identique, à **-7.8**, ce qui est plus proche de l'optimal, et un taux de succès de **73%**. On peut remarquer que la valeur n'atteint pas un plateau, mais qu'elle continue d'augmenter, très lentement, au fil des itérations de l'algorithme. Quoi qu'il en soit, dans le Scénario 1, maintenir des classements Elo et s'en servir pour optimiser le choix des équipes ne semble donc pas pertinent.

Scénario 2. La tâche est ici beaucoup plus ardue et le choix des équipes (et des actions jointes) est plus large. La difficulté de la tâche est sans équivoque comparée au Scénario 1, puisque pour ce scénario, les politiques jointes calculées n'ont que **10%** à **11%** de taux de succès après six étapes de décision. Les résultats présentés sur la figure 8.9 sont toutefois plus encourageants concernant notre algorithme. L'algorithme MELO (——) arrive à trouver des politiques générant une valeur de **-20,056** en moyenne au bout de 10000 itérations, là ou l'algorithme MA-POS-UCT (——) ne va atteindre que **-20,99**. Pour ce domaine, l'algorithme MELO fournit donc de meilleurs résultats que l'approche plus simple et moins dirigée de l'algorithme MA-POS-UCT.

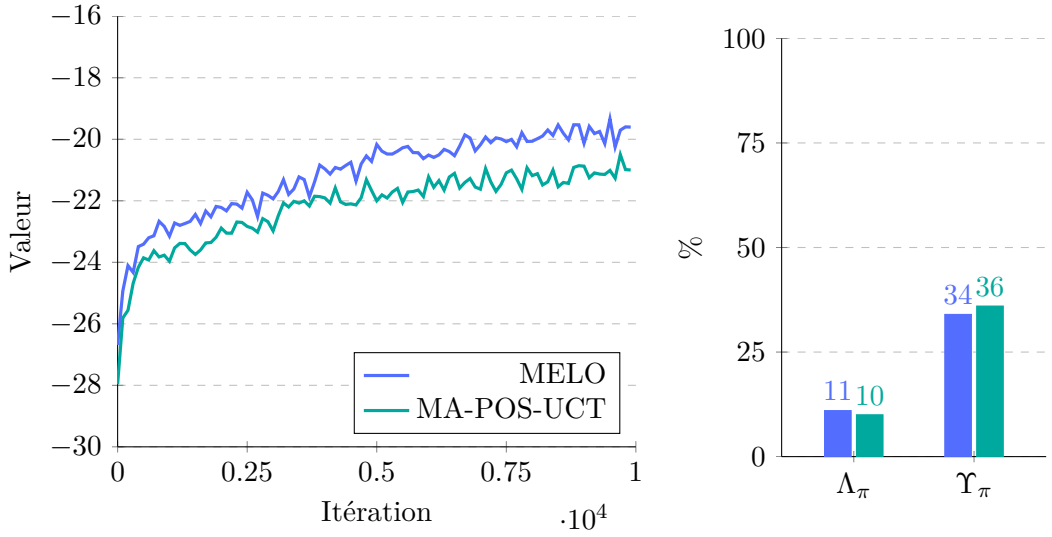


FIGURE 8.9 – **Scénario 2** – **Gauche :** Évolution des valeurs des politiques jointes calculées à horizon 6 au fur et à mesure de l'exécution des algorithmes MELO (——) et MA-POS-UCT (——) pour le Scénario 2 du *Problème de la zone sinistrée* ($G = 2$,$\mathcal{C} = \{(0,0),(1,0),(2,0),(0,1),(1,1),(2,1),(0,2),(1,2)(2,2)\}$). **Droite :** Taux de succès Λ_π et efficacités Υ_π moyens, en pourcentages.

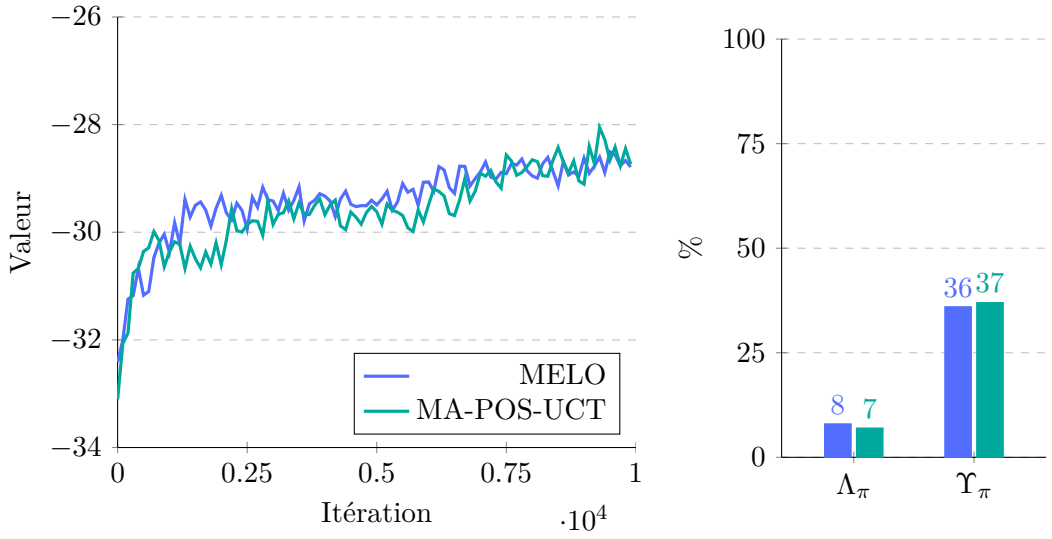


FIGURE 8.10 – **Scénario 3** – **Gauche :** Évolution des valeurs des politiques jointes calculées à horizon 6 au fur et à mesure de l'exécution des algorithmes MELO (——) et MA-POS-UCT (——) pour le Scénario 3 du *Problème de la zone sinistrée* ($G = 2$,$\mathcal{C} = \{(0,0),(1,0),(2,0),(0,1),(1,1),(2,1),(0,2),(1,2)(2,2)\}$,$c_i = 0{,}5$). **Droite :** Taux de succès Λ_π et efficacités Υ_π moyens, en pourcentages.

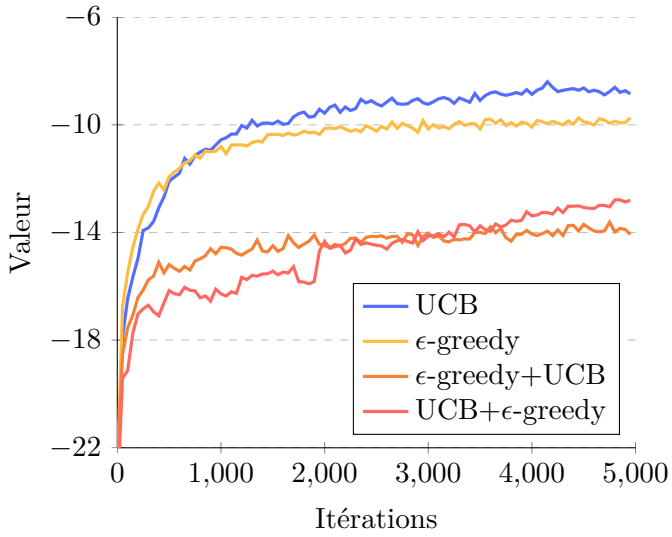


FIGURE 8.11 – Valeurs des politiques jointes calculées à horizon 6 avec l'algorithme MELO pour le Scénario 1 en faisant varier la stratégie de descente de l'arbre de recherche ($\epsilon = 0.01$).

Scénario 3. Lorsque l'on inclut un coût non nul aux agents, les deux algorithmes MELO et MA-POS-UCT performent de nouveau de façon similaire, une méthode n'étant pas significativement meilleure que l'autre, comme présenté sur la figure 8.10. La difficulté est encore supérieure : les politiques jointes calculées à horizon 6 n'ont que **7%** à **8%** de taux de succès. De plus, la valeur de la politique jointe calculée évolue relativement peu entre la première et la dernière itération de l'algorithme, passant de environ **-33** à environ **-28,7** après 10000 itérations. Notons que l'apparente variabilité sur les valeurs n'est due qu'à l'échelle de l'axe des ordonnées.

8.3.4 Stratégies de descente

La stratégie de descente de l'arbre de recherche dans tous les algorithmes basés sur MCTS que nous avons étudiés dans cette thèse repose sur l'usage de la formule UCB. Pour rappel, dans l'algorithme MELO, l'action jointe $\bar{a}_C$ choisie à un historique d'observations joint $\vec{o}$ durant l'étape de descente est :

$$\bar{a}_C = \underset{a_C \in \mathcal{A}}{\operatorname{argmax}} \frac{Q(\vec{o},a_C)}{N(\vec{o},a_C)} + \chi \cdot \sqrt{\frac{\ln N(\vec{o})}{N(\vec{o},a_C)}} \tag{8.3}$$

où

$$N(\vec{o}) = \sum_{C \in \mathcal{C}} \sum_{a_C \in \mathcal{A}_C} N(\vec{o},a_C) \tag{8.4}$$

Cette action jointe spécifie aussi directement l'équipe $C \in \mathcal{C}$ qui doit être opérationnelle à ce moment-là pour exécuter cette action.

Nous avons voulu tester des stratégies autres qu'UCB pour sélectionner l'équipe et l'action jointe lors de la phase de descente de l'arbre de recherche. Nous avons essayé les trois approches suivantes.

1. **ϵ-greedy.** Soit $0 \leqslant \epsilon \ll 1$. À un historique $\vec{o}$, on choisit avec une probabilité $1 - \epsilon$ l'action jointe $\bar{a}_C \in \mathcal{A}_C$ d'une équipe $C \in \mathcal{C}$ qui est la meilleure action jointe en moyenne, c'est-à-

dire telle que

$$\bar{a}_C = \underset{a_C \in \mathcal{A}}{\operatorname{argmax}} \frac{Q(\vec{o},a_C)}{N(\vec{o},a_C)}, \tag{8.5}$$

sinon on choisit une action jointe aléatoire d'une équipe aléatoire avec une probabilité ϵ.

2. **ϵ-greedy+UCB.** Soit $0 \leqslant \epsilon \ll 1$. On commence par choisir avec une probabilité $1-\epsilon$ l'équipe $\bar{C} \in \mathcal{C}$ qui est la meilleure équipe en moyenne, c'est-à-dire telle que

$$\bar{C} = \underset{C \in \mathcal{C}}{\operatorname{argmax}} \sum_{a_C \in \mathcal{A}_C} \frac{1}{|\mathcal{A}_C|} \frac{Q(\vec{o},a_C)}{N(\vec{o},a_C)}, \tag{8.6}$$

sinon on choisit une équipe aléatoire. Une fois l'équipe choisie, on sélectionne l'action de cette équipe qui maximise UCB.

3. **UCB+ϵ-greedy.** Soit $0 \leqslant \epsilon \ll 1$. On commence par choisir l'équipe $\bar{C} \in \mathcal{C}$ qui maximise UCB, puis on choisit avec une probabilité $1-\epsilon$ la meilleure action jointe $\bar{a}_{\bar{C}}$ en moyenne de cette équipe $\bar{C}$, c'est-à-dire telle que

$$\bar{a}_{\bar{C}} = \underset{a_{\bar{C}} \in \mathcal{A}_{\bar{C}}}{\operatorname{argmax}} \frac{Q(\vec{o},a_{\bar{C}})}{N(\vec{o},a_{\bar{C}})}, \tag{8.7}$$

sinon on choisit une action jointe aléatoire de l'équipe $\bar{C}$.

La figure 8.11 présente certains des résultats que l'on a pu obtenir en effectuant une comparaison entre ces trois stratégies de descente, en plus de la stratégie UCB de base. Les résultats sont sans équivoque : UCB (——) est plus efficace que les autres stratégies testées. On remarque qu'il est particulièrement contre-productif de sélectionner l'action jointe séparément de l'équipe, comme le font les stratégies ϵ-greedy+UCB (——) et UCB+ϵ-greedy (——). Rechercher directement dans l'espace de toutes les actions jointes *et* de toutes les équipes est plus efficace et évite de supprimer des ensembles d'actions jointes potentiellement efficaces. Utiliser uniquement la stratégie ϵ-greedy (——) produit de meilleurs résultats, mais néanmoins toujours inférieurs à la stratégie UCB (——).

8.3.5 Définitions de nœuds extensibles

Nous avons également essayé de faire varier la définition de *nœud extensible*. La première étape de tout algorithme de type MCTS, la descente de l'arbre de recherche, prend fin lorsque l'on rencontre un nœud extensible. Dans l'algorithme MCTS de base, un nœud est extensible si toutes les actions n'ont pas été essayées au moins une fois à ce nœud. Dans notre cas, le nombre d'actions jointes à essayer à chaque nœud dans le cadre de l'algorithme MELO peut éventuellement être très important. C'est la raison pour laquelle nous avons décidé de définir un nœud extensible comme étant un nœud dans lequel toutes les actions de l'équipe ayant le meilleur classement Elo n'ont pas été essayées au moins une fois.

Ici, nous essayons deux nouvelles définitions de nœuds extensibles que nous comparons à notre définition originelle.

1. **All.** Un nœud est extensible si toutes les actions n'ont pas été essayées au moins une fois à ce nœud (c'est la définition régulièrement rencontrée dans le cadre de l'algorithme MCTS).
2. **AllOfBest.** Un nœud est extensible si toutes les actions de l'équipe ayant le meilleur classement Elo n'ont pas été essayées au moins une fois à ce nœud (c'est notre définition originelle).

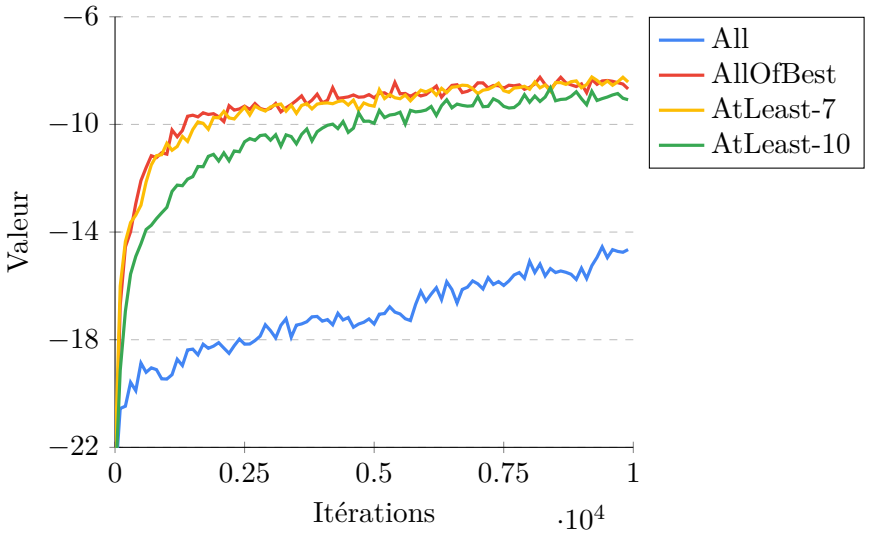


FIGURE 8.12 – Valeurs des politiques jointes calculées à horizon 6 avec l'algorithme MELO pour le Scénario 1 en faisant varier la définition de nœud extensible ($x \in \{7,10\}$).

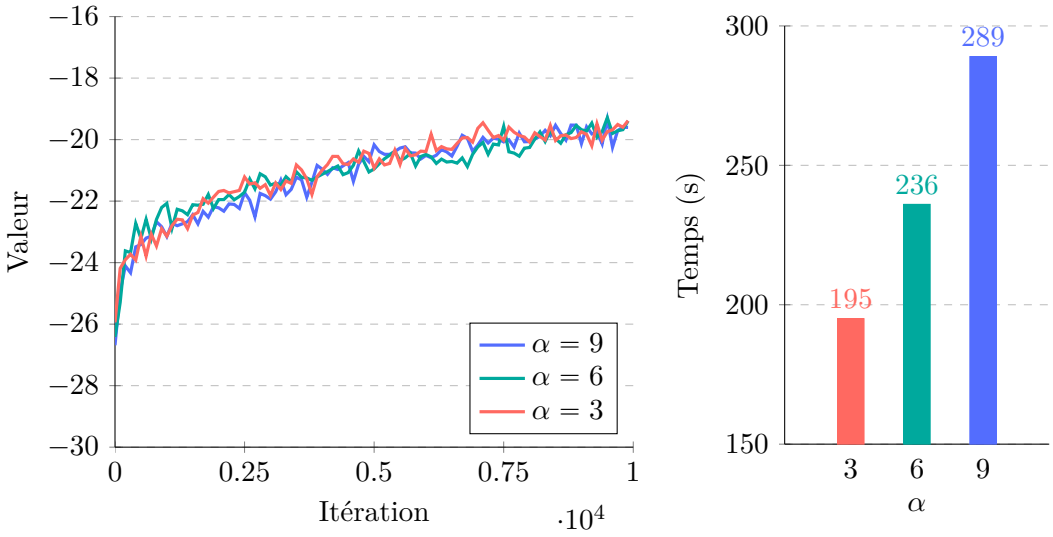


FIGURE 8.13 – **Gauche :** Évolution des valeurs des politiques jointes calculées à horizon 6 au fur et à mesure de l'exécution de l'algorithme MELO pour le Scénario 2 du *Problème de la zone sinistrée* ($G = 2, \mathcal{C} = \{(0,0),(1,0),(2,0),(0,1),(1,1),(2,1),(0,2),(1,2)(2,2), \alpha \in \{3,6,9\}\}$), pour différentes valeurs de α. **Droite :** Temps de calcul moyens, en secondes.

3. **AtLeast-x.** Un nœud est extensible si au moins $x > 0$ actions n'ont pas été essayées au moins une fois à ce nœud.

La figure 8.12 présente les résultats obtenus par l'algorithme MELO sur le Scénario 1 lorsque l'on choisit différentes définitions de nœud extensible. Nous avons testé plusieurs valeurs de x pour la définition AtLeast-x. On remarque que les définitions AllOfBest (—) et AtLeast-7 (—) produisent des résultats très semblables, en convergeant toutes deux vers environ **-8.57**. De son côté, la définition All (—) sous-performe très largement, puisqu'elle ne permet d'atteindre que **-14.65** à la fin des 10000 itérations, sans qu'il soit possible de dire si l'algorithme a convergé ou si la qualité peut encore continuer d'augmenter. Remarquons également que AtLeast-13 = All, puisqu'il y a exactement 13 actions jointes possibles dans le Scénario 1. En prenant des valeurs pour x comprises entre 7 et 13, les courbes obtenues se situent toutes entre AtLeast-7 et AtLeast-13. Par exemple, AtLeast-10 (—) donne des résultats légèrement inférieurs à AtLeast-7. Finalement, en abaissant encore x en-dessous de 7, nous n'avons pas pu obtenir des valeurs supérieures à celles obtenues avec la définition AllOfBest.

8.3.6 Matchs des meilleures équipes

Dans la section 6.5.2, nous avons décrit un moyen pour permettre de diminuer le nombre de calculs effectués dans une itération de l'algorithme MELO. Nous avons émis l'hypothèse qu'il doit être possible de ne considérer que les $\alpha > 2$ équipes ayant le meilleur classement Elo durant les phases de simulations parallèles et de matchs par paires pour obtenir des résultats aussi satisfaisants que si l'on mettait à jour le classement Elo de toutes les équipes. Jusqu'à présent, dans tous les résultats présentés, α était choisi pour correspondre exactement au nombre d'équipes maximum disponible dans chaque scénario (c'est-à-dire, respectivement 3, 9 et 9). Nous avons donc fait varier le paramètre α dans le cas du Scénario 2 (le Scénario 1 étant trop simple). Nos résultats sont présentés sur la figure 8.13.

Il n'y a pas de différence statistiquement significative entre les résultats obtenus pour chacune des trois valeurs testées pour α. La seule différence se situe au niveau du temps de calcul moyen mis par l'algorithme MELO pour effectuer les 10000 itérations. Lorsque $\alpha = 9$ (—), c'est-à-dire lorsque l'on effectue les matchs entre toutes les équipes, l'algorithme nécessite en moyenne **289s** de calcul, alors qu'il ne lui faut que **195s** lorsque $\alpha = 3$ (—). Ce résultat est même compétitif avec le temps de calcul nécessité par l'algorithme MA-POS-UCT de **183s**, tel que reporté sur le tableau 8.5, alors que l'algorithme MELO performe mieux que l'algorithme MA-POS-UCT sur le Scénario 2.

8.4 Conclusion

Ce chapitre nous aura permis de procéder au test et à l'évaluation des algorithmes que nous avons introduits dans les chapitres précédents de cette thèse. Les résultats présentés nous ont permis de mettre en valeur certaines qualités de nos algorithmes. Parmi celles-ci, il y a la capacité qu'a l'algorithme BRD-POS-UCT à contrôler la taille de l'équipe opérationnelle lorsque les agents ont un coût non nul mais que l'emploi d'un grand nombre d'agents semble profitable pour résoudre la tâche efficacement. Ou encore, l'efficacité de l'algorithme MELO pour détecter les équipes performantes et les équipes clairement inefficaces dans le scénario simple du *Fire Fighting*. De plus, nous avons montré comment réduire le temps de calcul de l'algorithme MELO en diminuant le nombre de matchs effectués à chaque itération sans sacrifier la qualité des solutions trouvées.

Nos résultats, bien que préliminaires, sont clairement encourageants. L'emploi de l'algorithme MCTS, et son adaptation au cas partiellement observable et stochastique, semblent être des voies prometteuses pour la résolution de problèmes de grande taille, notamment les DEC-POMDPs, mais également les TEAM-POMDPs. Dans notre travail, nous avons essayé, avec l'algorithme MELO, d'exploiter la structure des TEAM-POMDPs pour améliorer la qualité des politiques jointes que l'on calculerait avec un algorithme de résolution plus simple, comme MA-POS-UCT (qui n'est au final qu'un algorithme de résolution de POMDP). Les TEAM-POMDPs visent à s'attaquer à des problèmes de plus grande taille que les DEC-POMDPs : intuitivement, on ne considère pas une seule équipe avec x actions jointes, mais y équipes, avec chacune x actions jointes. Derrière cette apparente complexité, il demeure possible d'analyser le problème sous-jacent (est-il sous-additif ? Certaines équipes sont-elles clairement perdantes ?) pour améliorer à la fois la vitesse de convergence et la valeur des politiques jointes.

Il nous reste encore beaucoup de voies à explorer concernant les algorithmes BRD-POS-UCT et MELO. De très nombreuses recherches ont été menées sur l'algorithme MCTS [Browne et al., 2012], aussi certaines méthodes existantes peuvent certainement être appliquées ou modifiées pour apporter des solutions au problème de la formation d'équipes sous incertitude. Ici, nous avons essayé de modifier la stratégie de descente de l'arbre généralement employée dans tous les algorithmes basés sur MCTS, mais il se trouve que les résultats obtenus n'ont pas été concluants (après tout, UCB n'est pas utilisé par presque tous les algorithmes existants sans raisons).

Cinquième partie

Conclusion

Chapitre 9

Conclusion

« Expliquer toute la nature est une tâche trop ardue pour un seul homme ou une seule époque. Il est plus sage de faire peu en étant sûr de soi et laisser le reste à ceux qui viendront après, que présumer de tout sans être sûr de rien. »

Isaac Newton

9.1 Bilan

L'objectif de cette thèse était de proposer des moyens pour permettre à un ensemble d'agents coopératifs autonomes de se reformer, de retirer ou d'ajouter des agents à l'équipe opérationnelle afin de faire face à l'évolution de la tâche qui leur est confiée. Il s'agit d'un problème qui n'a été que peu, voire pas du tout, traité dans le domaine de la planification multi-agents sous incertitude, pourtant il existe de nombreuses applications réelles où l'équipe mobilisée n'est pas destinée à rester la même durant la totalité de sa mission.

Afin de nous attaquer à ce genre de scénarios, il nous a fallu définir formellement le problème de planification sur les actions des agents et sur la formation d'équipes : le problème TARS. Puis, nous sommes partis d'un modèle de décision et de planification multi-agents existant et bien étudié dans littérature, le DEC-POMDP, pour l'étendre aux cas ouverts, où la population des agents dans le système est susceptible d'être modifiée durant l'exécution de la tâche. Notre premier axe de contributions concerne nos modèles de décision. Nous avons introduit le modèle d'OPEN-DEC-POMDP, que nous avons par la suite restreint aux situations où les entrées et sorties des agents restent sous contrôle. Cela a donné le modèle de TEAM-POMDP, dont nous avons donné et étudié certaines propriétés structurelles. Étant un modèle dérivé, et équivalent, des DEC-POMDPs, les TEAM-POMDPs souffrent de la même complexité théorique, doublement exponentielle avec le nombre d'agents et l'horizon de planification.

Notre choix pour la résolution de nos modèles a ainsi consisté à nous tourner vers une méthode de résolution approximative mais ayant produit des résultats concluants dans d'autres domaines de l'intelligence artificielle : la recherche arborescente Monte-Carlo. Nos algorithmes constituent le second axe de nos contributions. D'abord, l'algorithme BRD-POS-UCT calcule les politiques de meilleures réponses de chaque agent jusqu'à converger vers un équilibre. Chaque meilleure réponse est construite avec l'algorithme POS-UCT, une version de MCTS que nous avons adaptée aux environnements partiellement observables et stochastiques. L'algorithme BRD-POS-UCT

n'est toutefois qu'un algorithme de résolution de DEC-POMDPs, que nous avons appliqué, en l'occurrence, au cas ouvert. Nous avons donc construit l'algorithme MELO, un vrai algorithme de résolution de TEAM-POMDPs, qui calcule quant à lui une politique jointe. Cet algorithme repose sur la mise à jour des classements Elo de chaque équipe dans chaque historique d'observations joint pour prédéterminer les équipes efficaces et les équipes peu prometteuses. Nos résultats montrent que l'utilisation des classements Elo permet à notre algorithme de générer des politiques au moins aussi bonnes (voire meilleures, suivant le scénario étudié) qu'un algorithme de type POS-UCT, plus simple, tout en ayant des temps de calculs aussi rapides.

Ces deux axes de contributions (modèles et algorithmes) permettent de résoudre, au moins approximativement, le problème TARS, et donc d'apporter une réponse positive à cette thèse.

9.2 Perspectives

Il y a plusieurs directions vers lesquelles se diriger après le travail décrit et développé dans cette thèse. Nous esquissons ici quelques pistes de réflexion pour de futurs travaux.

Résolution optimale. Tout d'abord, et c'est ce qui nous semble être le plus important, il est nécessaire de proposer des algorithmes de résolution optimale aux modèles d'OPEN-DEC-POMDP et de TEAM-POMDP. Les algorithmes BRD-POS-UCT et MELO sont des algorithmes approximatifs, sans réelles garanties théoriques de convergence. Étant donné qu'il est possible de représenter un TEAM-POMDP avec un DEC-POMDP, il semble qu'il soit donc également possible d'appliquer l'ensemble des nombreux algorithmes optimaux de DEC-POMDPs existants directement dans le cadre des systèmes ouverts. Il doit également être possible de développer de nouveaux algorithmes optimaux qui pourraient traiter efficacement certaines sous-classes de TEAM-POMDPs, sous-classes qu'il reste à exhiber.

Définitions étendues au cas séquentiel. Une étude plus approfondie du modèle de TEAM-POMDP peut être menée. Bien qu'il s'agisse d'un modèle semblable à un DEC-POMDP, il possède néanmoins certaines caractéristiques qui le différencient de ce modèle. En particulier, nous pensons qu'il est possible d'étendre les notions de monotonie, de sur-additivité et de convexité au cas séquentiel. Par exemple, si un TEAM-POMDP est monotone (c'est-à-dire que ses fonctions de récompense sont monotones par rapport aux équipes d'agents), alors étant donnée une politique jointe, la valeur de cette politique jointe est-elle également monotone par rapport aux équipes d'agents ? On pourrait s'inspirer des travaux de [Kumar et al., 2017] pour réaliser ce travail.

Nouveaux benchmarks. Il est possible de proposer plus d'expérimentations pour réellement évaluer les performances des algorithmes présentés dans cette thèse. Nous avons vu que, suivant le scénario sur lequel il est évalué, l'algorithme MELO ne produit pas toujours de meilleurs résultats qu'un algorithme plus simple, tel que MA-POS-UCT. Il faut donc multiplier les expérimentations en faisant varier les différents paramètres de chaque algorithme pour obtenir une étude empirique exhaustive, évaluant la pertinence de ces algorithmes. Il pourrait également être intéressant d'étudier de nouveaux problèmes afin de déterminer pour lesquels d'entre eux l'utilisation des classements Elo est la plus pertinente. Il faut pour cela introduire de nouveaux benchmarks, avec des nombres d'équipes importants, des agents hétérogènes ou encore des fonctions de coûts non-linéaires pénalisant la présence d'agents dans l'équipe opérationnelle.

Politiques séparables. L'algorithme BRD-POS-UCT produit des politiques jointes séparables, mais ce n'est pas le cas de l'algorithme MELO, qui se sert d'indices de pouvoir sur les équipes et non pas sur les agents. C'est un problème en soi car cela empêche le contrôle décentralisé de l'équipe d'agents : un agent ne peut pas extraire sa politique individuelle de l'arbre de recherche pour l'exécuter de façon indépendante. On peut envisager deux voies différentes afin d'obtenir des politiques jointes séparables. D'une part, on pourrait rendre la politique jointe calculée par l'algorithme MELO séparable une fois que l'algorithme a terminé l'étape de planification (en suivant, par exemple, les pistes données en section 6.5.4). D'autre part, on pourrait s'assurer que, durant la construction de l'arbre de recherche, la politique jointe est garantie d'être toujours séparable. Notons finalement que la différence de qualité entre une politique jointe séparable et une politique jointe non-séparable peut être colossale.

Signal global. Toujours dans l'optique de générer des politiques jointes séparables, il peut être intéressant de développer l'idée, brièvement évoquée dans la section 6.4.1, de signal global. Ce signal de coordination centralisé, qui serait une observation globale (que tous les agents recevraient), fonction de l'état du système, renseignerait suffisamment chacun des agents (ceux de l'équipe opérationnelle *et* ceux de la réserve) pour leur permettre de décider de faire, ou non, partie de l'équipe opérationnelle lors de la prochaine étape de décision. On pourrait ainsi calculer séparément, pour chaque agent, une politique individuelle de domaine, qui lui dirait comment agir lorsqu'il est présent dans l'équipe en fonction de ses observations (comme une politique de POMDP), et une politique individuelle d'entrée-sorties, qui lui dirait si oui on non il doit faire partie de l'équipe opérationnelle en fonction de l'état du signal global.

Autres indices de pouvoir. Dans cette thèse, nous avons décidé de classer les équipes en fonction de leur classement Elo, mais il existe d'autres indices de pouvoir, comme l'indice de Shapley-Shubik ou l'indice de Banzhaf, souvent utilisés en théorie de jeux (et, plus précisément, des jeux de vote). Il serait intéressant d'utiliser ces autres systèmes de notation, d'une part pour faire la comparaison avec notre approche, et d'autre part pour déterminer s'il est possible d'évaluer directement les agents plutôt que les équipes, dans le but de concevoir des politiques individuelles à chaque agent (et donc des politiques jointes séparables).

Et merci à toi, Anaïs.

Bibliographie

[Agogino and Tumer, 2004] Agogino, A. K. and Tumer, K. (2004). Unifying Temporal and Structural Credit Assignment Problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS'04, pages 980–987, Washington, District of Columbia, USA. IEEE Computer Society.

[Agrawal and Varakantham, 2017] Agrawal, P. and Varakantham, P. (2017). Proactive and Reactive Coordination of Non-dedicated Agent Teams Operating in Uncertain Environments. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 28–34, Palo Alto, California, USA. AAAI Press.

[Agrawal et al., 2018] Agrawal, P., Varakantham, P., and Yeoh, W. (2018). Decentralized Planning for Non-Dedicated Agent Teams with Submodular Rewards in Uncertain Environments. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*, UAI'18, pages 958–967, Corvallis, Oregon, USA. AUAI Press.

[Åström, 1965] Åström, K. J. (1965). Optimal Control of Markov Processes with Incomplete State Information. *Journal of Mathematical Analysis and Applications*, 10(1) :174–205.

[Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3) :235–256.

[Auger, 2011] Auger, D. (2011). Multiple Tree for Partially Observable Monte-Carlo Tree Search. *Computing Research Repository*, abs/1102.1580.

[Banzhaf III, 1964] Banzhaf III, J. F. (1964). Weighted Voting Doesn't Work : A Mathematical Analysis. *Rutgers Law Review*, 19 :317.

[Barrett et al., 2016] Barrett, S., Rosenfeld, A., Kraus, S., and Stone, P. (2016). Making Friends on the Fly : Cooperating with New Teammates. *Artificial Intelligence*.

[Barrett and Stone, 2012] Barrett, S. and Stone, P. (2012). An Analysis Framework for Ad Hoc Teamwork Tasks. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 357–364, Richland, South Carolina, USA. International Foundation for Autonomous Agents and Multiagent Systems.

[Barrett et al., 2012] Barrett, S., Stone, P., Kraus, S., and Rosenfeld, A. (2012). Learning Teammate Models for Ad Hoc Teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*.

[Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6) :1554–1563.

[Bayes and Price, 1763] Bayes, M. and Price, M. (1763). An Essay towards Solving a Problem in the Doctrine of Chances. By the Late Rev. Mr. Bayes, F. R. S. Communicated by Mr. Price, in a Letter to John Canton, A. M. F. R. S. *Philosophical Transactions (1683-1775)*, 53 :370–418.

[Bell, 1982] Bell, D. E. (1982). Regret in Decision Making Under Uncertainty. *Operations Research*, 30(5) :961–981.

[Bellman, 1954] Bellman, R. (1954). The Theory of Dynamic Programming. Technical report, RAND Corporation, Santa Monica, California, USA.

[Bellman, 1956] Bellman, R. (1956). Dynamic Programming and the Smoothing Problem. *Management Science*, 3(1) :111–113.

[Bellman, 1984] Bellman, R. (1984). *Eye of the Hurricane : An Autobiography*. Series in Modern Applied Mathematics. World Scientific.

[Benda et al., 1986] Benda, M., Jagannathan, V., and Dodhiawala, R. (1986). On Optimal Cooperation of Knowledge Sources - An Empirical Investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, USA.

[Bernstein et al., 2000] Bernstein, D. S., Zilberstein, S., and Immerman, N. (2000). The Complexity of Decentralized Control of Markov Decision Processes. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 32–37. Morgan Kaufmann Publishers Inc.

[Bertsekas et al., 2005] Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control*, volume 1. Athena scientific Belmont, MA.

[Binmore, 2008] Binmore, K. (2008). *Rational Decisions*. Princeton University Press.

[Browne et al., 2012] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S., et al. (2012). A Survey of Monte Carlo Tree Search Methods. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 4, pages 1–43.

[Bylander, 1994] Bylander, T. (1994). The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1-2) :165–204.

[Calmet et al., 2004] Calmet, J., Daemi, A., Endsuleit, R., and Mie, T. (2004). A Liberal Approach to Openness in Societies of Agents. In Omicini, A., Petta, P., and Pitt, J., editors, *Engineering Societies in the Agents World IV*, pages 81–92, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Campbell and Wu, 2011] Campbell, A. and Wu, A. S. (2011). Multi-agent Role Allocation : Issues, Approaches, and Multiple Perspectives. *Autonomous Agents and Multi-Agent Systems*, 22(2) :317–355.

[Cassandra, 1998a] Cassandra, A. (1998a). A Survey of POMDP Applications. *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*.

[Cassandra et al., 1997] Cassandra, A., Littman, M. L., and Zhang, N. L. (1997). Incremental pruning : a Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 54–61. Morgan Kaufmann Publishers Inc.

[Cassandra, 1998b] Cassandra, A. R. (1998b). *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Providence, Rhode Island, USA. AAI9830418.

[Chandrasekaran et al., 2016] Chandrasekaran, M., Eck, A., Doshi, P., and Soh, L. (2016). Individual Planning in Open and Typed Agent Systems. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'16, pages 82–91, Arlington, Virginia, United States. AUAI Press.

[Cohen et al., 2017] Cohen, J., Dibangoye, J. S., and Mouaddib, A.-I. (2017). Open Decentralized POMDPs. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 977–984, New York, New York, USA. IEEE.

[Cohen and Mouaddib, 2018a] Cohen, J. and Mouaddib, A.-I. (2018a). Monte-Carlo Planning for Team Re-Formation Under Uncertainty : Model and Properties. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 458–465, New York, New York, USA. IEEE.

[Cohen and Mouaddib, 2018b] Cohen, J. and Mouaddib, A.-I. (2018b). Re-formation décentralisée d'équipes sous incertitude : modèle et propriétés structurelles. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA 2018)*, Nancy, France.

[Cohen and Mouaddib, 2019] Cohen, J. and Mouaddib, A.-I. (2019). Power Indices for Team Reformation Planning Under Uncertainty : Extended Abstract. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS'19, pages 458–465, New York, New York, USA. ACM.

[Cohen and Levesque, 1991] Cohen, P. R. and Levesque, H. J. (1991). Teamwork. *Noûs*, 25(4) :487–512.

[Curiel, 2013] Curiel, I. (2013). *Cooperative Game Theory and Applications : Cooperative Games Arising from Combinatorial Optimization Problems*. Theory and Decision Library C. Springer US.

[Devlin et al., 2014] Devlin, S., Yliniemi, L., Kudenko, D., and Tumer, K. (2014). Potential-based Difference Rewards for Multiagent Reinforcement Learning. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 165–172, Richland, South Carolina, USA. International Foundation for Autonomous Agents and Multiagent Systems.

[Dibangoye et al., 2016] Dibangoye, J., Amato, C., Buffet, O., and Charpillet, F. (2016). Optimally Solving Dec-POMDPs as Continuous-State MDPs. *Journal of Artificial Intelligence Research*, 55 :443–497.

[Eijk et al., 1999] Eijk, R. M. v., Boer, F. S. d., Hoek, W. v. d., and Meyer, J.-J. C. (1999). Open Multi-agent Systems : Agent Communication and Integration. In *6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL),*, ATAL '99, pages 218–232, Berlin, Heidelberg. Springer-Verlag.

[Elo, 1978] Elo, A. E. (1978). *The Rating of Chessplayers, Past and Present*, volume 3. Batsford London.

[Ferrari, 2017] Ferrari, F. V. (2017). *Cooperative POMDPs for Human-Robot Joint Activities*. PhD thesis, Normandie Université.

[Fishburn, 2013] Fishburn, P. C. (2013). *The Foundations of Expected Utility*, volume 31. Springer Science & Business Media.

[Gale, 1989] Gale, D. (1989). *The Theory Of Linear Economic Models*. Economics / mathematics. University of Chicago Press.

[Genter et al., 2011] Genter, K., Agmon, N., and Stone, P. (2011). Role-based Ad Hoc Teamwork. In *Proceedings of the 16th AAAI Conference on Plan, Activity, and Intent Recognition*, AAAIWS'11-16, pages 17–24. AAAI Press.

[Genter and Stone, 2014] Genter, K. and Stone, P. (2014). Orienting a Flock via Ad Hoc Teamwork. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '14, pages 1543–1544, Richland, South Carolina, USA. International Foundation for Autonomous Agents and Multiagent Systems.

[Genter and Stone, 2016] Genter, K. and Stone, P. (2016). Ad hoc Teamwork Behaviors for Influencing a Flock. *Acta Polytechnica*, 56 :18.

[Gerkey and Matarić, 2003a] Gerkey, B. P. and Matarić, M. J. (2003a). A Formal Framework for the Study of Task Allocation in Multi-robot Systems. *International Journal of Robotic Research - IJRR*.

[Gerkey and Matarić, 2003b] Gerkey, B. P. and Matarić, M. J. (2003b). On Role Allocation in RoboCup. In *Robot Soccer World Cup*, pages 43–53. Springer.

[Glickman, 1998] Glickman, M. E. (1998). The Glicko system. *Boston University*.

[Goldman and Zilberstein, 2004] Goldman, C. V. and Zilberstein, S. (2004). Decentralized Control of Cooperative Systems : Categorization and Complexity Analysis. *Journal of Artificial Intelligence Research*, 22(1) :143–174.

[Hansen et al., 2004] Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pages 709–715. AAAI Press.

[Hendrickx and Martin, 2017] Hendrickx, J. M. and Martin, S. (2017). Open Multi-agent Dystems : Gossiping with Random Arrivals and Departures. *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 763–768.

[Herbrich et al., 2007] Herbrich, R., Minka, T., and Graepel, T. (2007). TrueSkill$^{\text{TM}}$: a Bayesian skill rating system. In *Advances in Neural Information Processing Systems*, pages 569–576.

[Hoey et al., 2010] Hoey, J., Poupart, P., Bertoldi, A. v., Craig, T., Boutilier, C., and Mihailidis, A. (2010). Automated Handwashing Assistance for Persons with Dementia Using Video and a Partially Observable Markov Decision Process. *Computer Vision and Image Understanding*, 114(5) :503–519.

[Howard, 1960] Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. Technology Press of the Massachusetts Institute of Technology.

[Hsu et al., 2007] Hsu, D., Lee, W. S., and Rong, N. (2007). What Makes Some POMDP Problems Easy to Approximate ? In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 689–696, USA. Curran Associates Inc.

[Huynh et al., 2006] Huynh, T. D., Jennings, N. R., and Shadbolt, N. R. (2006). An Integrated Trust and Reputation Model for Open Multi-agent Systems. *Autonomous Agents and Multi-Agent Systems*, 13(2) :119–154.

[Kaelbling et al., 1998] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artificial intelligence*, 101(1-2) :99–134.

[Katehakis and Veinott, 1987] Katehakis, M. N. and Veinott, A. F. (1987). The multi-armed bandit problem : Decomposition and computation. *Mathematics of Operations Research*, 12(2) :262–268.

[Katt et al., 2017] Katt, S., Oliehoek, F. A., and Amato, C. (2017). Learning in pomdps with monte carlo tree search. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1819–1827. JMLR.org.

[Kendall, 1953] Kendall, D. G. (1953). Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 24(3) :338–354.

[Kitano and Tadokoro, 2001] Kitano, H. and Tadokoro, S. (2001). RoboCup Rescue : A Grand Challenge for Multiagent and Intelligent Systems. *AI Magazine*, 22 :39–52.

[Kitano et al., 1997] Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., and Asada, M. (1997). The RoboCup Synthetic Agent Challenge 97. In *Robot Soccer World Cup*, pages 62–73. Springer.

[Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg. Springer-Verlag.

[Kuhn, 1955] Kuhn, H. W. (1955). The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2(1-2) :83–97.

[Kumar et al., 2017] Kumar, R. R., Varakantham, P., and Kumar, A. (2017). Decentralized Planning in Stochastic Environments with Submodular Rewards. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 3021–3028. AAAI Press.

[Kushmerick et al., 1995] Kushmerick, N., Hanks, S., and Weld, D. S. (1995). An Algorithm for Probabilistic Planning. *Artificial Intelligence*, 76(1-2) :239–286.

[Littman et al., 1995a] Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995a). Efficient Dynamic-programming Updates in Partially Observable Markov Decision Processes. Technical report, Brown University, Providence, Rhode Islande, USA.

[Littman et al., 1995b] Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995b). On the Complexity of Solving Markov Decision Problems. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 394–402, San Francisco, California, USA. Morgan Kaufmann Publishers Inc.

[Loomes and Sugden, 1982] Loomes, G. and Sugden, R. (1982). Regret Theory : An Alternative Theory of Rational Choice Under Uncertainty. *The Economic Journal*, 92(368) :805–824.

[Madani et al., 1999] Madani, O., Hanks, S., and Condon, A. (1999). On the Undecidability of Probabilistic Planning and Infinite-horizon Partially Observable Markov Decision Problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 541–548, Menlo Park, California, USA. American Association for Artificial Intelligence.

[Markov, 1906] Markov, A. A. (1906). Extension of the Law of Large Numbers to Dependant Quantities. *Izvestia, Fiziko-matematicheskomu otdeleniyu, Kazan University*, 15 :135–156.

[Metropolis and Ulam, 1949] Metropolis, N. and Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247) :335–341.

[Morgenstern and Von Neumann, 1953] Morgenstern, O. and Von Neumann, J. (1953). *Theory of Games and Economic Behavior*. Princeton University press.

[Nair et al., 2002] Nair, R., Tambe, M., and Marsella, S. (2002). Team Formation for Reformation. In *Proceedings of the AAAI spring symposium on intelligent distributed and embedded systems*, pages 52–56.

[Nair et al., 2003a] Nair, R., Tambe, M., and Marsella, S. (2003a). Role allocation and reallocation in multiagent teams : Towards a practical analysis. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS'03, pages 552–559, New York, New York, USA. ACM.

[Nair et al., 2003b] Nair, R., Tambe, M., and Marsella, S. (2003b). Team Formation for Reformation in Multiagent Domains Like RoboCupRescue. In Kaminka, G. A., Lima, P. U., and Rojas, R., editors, *RoboCup 2002 : Robot Soccer World Cup VI*, pages 150–161, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Nair et al., 2003c] Nair, R., Tambe, M., Yokoo, M., Pynadath, D., and Marsella, S. (2003c). Taming Decentralized POMDPs : Towards Efficient Policy Computation for Multiagent Settings. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, pages 705–711, San Francisco, California, USA. Morgan Kaufmann Publishers Inc.

[Nair et al., 2005] Nair, R., Varakantham, P., Tambe, M., and Yokoo, M. (2005). Networked Distributed POMDPs : A Synthesis of Distributed Constraint Optimization and POMDPs. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1*, AAAI'05, pages 133–139. AAAI Press.

[Nisan et al., 2007] Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V. (2007). *Algorithmic Game Theory*. Cambridge University Press, New York, New York, USA.

[Nodine and Unruh, 1997] Nodine, M. H. and Unruh, A. (1997). Facilitating Open Communication in Agent Systems : The InfoSleuth Infrastructure. In *Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, ATAL '97, pages 281–295, Berlin, Heidelberg. Springer-Verlag.

[Novozhilov et al., 2006] Novozhilov, A. S., Karev, G. P., and Koonin, E. V. (2006). Biological Applications of the Theory of Birth-and-death Processes. *Briefings in bioinformatics*, 7(1) :70–85.

[Oliehoek and Amato, 2016] Oliehoek, F. A. and Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition.

[Oliehoek et al., 2013] Oliehoek, F. A., Spaan, M. T. J., Amato, C., and Whiteson, S. (2013). Incremental Clustering and Expansion for Faster Optimal Planning in Decentralized POMDPs. *Journal of Artificial Intelligence Research*, 46(1) :449–509.

[Oliehoek et al., 2008] Oliehoek, F. A., Spaan, M. T. J., and Vlassis, N. (2008). Optimal and Approximate Q-value Functions for Decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32(1) :289–353.

[Pageaud et al., 2017] Pageaud, S., Deslandres, V., Lehoux, V., and Hassas, S. (2017). Co-Construction of Adaptive Public Policies using SmartGov. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1328–1335, New York, New York, USA. IEEE.

[Papadimitriou and Tsitsiklis, 1987] Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of operations research*, 12(3) :441–450.

[PDMIA, 2008] PDMIA, G. (2008). *Processus décisionnels de Markov en intelligence artificielle (Édité par Olivier Buffet et Olivier Sigaud)*. Lavoisier - Hermes Science Publications.

[Pineau et al., 2006] Pineau, J., Gordon, G., and Thrun, S. (2006). Anytime Point-based Approximations for Large POMDPs. *Journal of Artificial Intelligence Research*, 27(1) :335–380.

[Planck Collaboration et al., 2016] Planck Collaboration, Ade, P. A. R., Aghanim, N., Arnaud, M., Ashdown, M., Aumont, J., Baccigalupi, C., Banday, A. J., Barreiro, R. B., Bartlett, J. G.,

Bartolo, N., Battaner, E., Battye, R., Benabed, K., Benoît, A., Benoit-Lévy, A., Bernard, J.-P., Bersanelli, M., Bielewicz, P., Bock, J. J., Bonaldi, A., Bonavera, L., Bond, J. R., Borrill, J., Bouchet, F. R., Boulanger, F., Bucher, M., Burigana, C., Butler, R. C., Calabrese, E., Cardoso, J.-F., Catalano, A., Challinor, A., Chamballu, A., Chary, R.-R., Chiang, H. C., Chluba, J., Christensen, P. R., Church, S., Clements, D. L., Colombi, S., Colombo, L. P. L., Combet, C., Coulais, A., Crill, B. P., Curto, A., Cuttaia, F., Danese, L., Davies, R. D., Davis, R. J., de Bernardis, P., de Rosa, A., de Zotti, G., Delabrouille, J., Désert, F.-X., Di Valentino, E., Dickinson, C., Diego, J. M., Dolag, K., Dole, H., Donzelli, S., Doré, O., Douspis, M., Ducout, A., Dunkley, J., Dupac, X., Efstathiou, G., Elsner, F., Enßlin, T. A., Eriksen, H. K., Farhang, M., Fergusson, J., Finelli, F., Forni, O., Frailis, M., Fraisse, A. A., Franceschi, E., Frejsel, A., Galeotta, S., Galli, S., Ganga, K., Gauthier, C., Gerbino, M., Ghosh, T., Giard, M., Giraud-Héraud, Y., Giusarma, E., Gjerløw, E., González-Nuevo, J., Górski, K. M., Gratton, S., Gregorio, A., Gruppuso, A., Gudmundsson, J. E., Hamann, J., Hansen, F. K., Hanson, D., Harrison, D. L., Helou, G., Henrot-Versillé, S., Hernández-Monteagudo, C., Herranz, D., Hildebrandt, S. R., Hivon, E., Hobson, M., Holmes, W. A., Hornstrup, A., Hovest, W., Huang, Z., Huffenberger, K. M., Hurier, G., Jaffe, A. H., Jaffe, T. R., Jones, W. C., Juvela, M., Keihänen, E., Keskitalo, R., Kisner, T. S., Kneissl, R., Knoche, J., Knox, L., Kunz, M., Kurki-Suonio, H., Lagache, G., Lähteenmäki, A., Lamarre, J.-M., Lasenby, A., Lattanzi, M., Lawrence, C. R., Leahy, J. P., Leonardi, R., Lesgourgues, J., Levrier, F., Lewis, A., Liguori, M., Lilje, P. B., Linden-Vørnle, M., López-Caniego, M., Lubin, P. M., Macías-Pérez, J. F., Maggio, G., Maino, D., Mandolesi, N., Mangilli, A., Marchini, A., Maris, M., Martin, P. G., Martinelli, M., Martínez-González, E., Masi, S., Matarrese, S., McGehee, P., Meinhold, P. R., Melchiorri, A., Melin, J.-B., Mendes, L., Mennella, A., Migliaccio, M., Millea, M., Mitra, S., Miville-Deschênes, M.-A., Moneti, A., Montier, L., Morgante, G., Mortlock, D., Moss, A., Munshi, D., Murphy, J. A., Naselsky, P., Nati, F., Natoli, P., Netterfield, C. B., Nørgaard-Nielsen, H. U., Noviello, F., Novikov, D., Novikov, I., Oxborrow, C. A., Paci, F., Pagano, L., Pajot, F., Paladini, R., Paoletti, D., Partridge, B., Pasian, F., Patanchon, G., Pearson, T. J., Perdereau, O., Perotto, L., Perrotta, F., Pettorino, V., Piacentini, F., Piat, M., Pierpaoli, E., Pietrobon, D., Plaszczynski, S., Pointecouteau, E., Polenta, G., Popa, L., Pratt, G. W., Prézeau, G., Prunet, S., Puget, J.-L., Rachen, J. P., Reach, W. T., Rebolo, R., Reinecke, M., Remazeilles, M., Renault, C., Renzi, A., Ristorcelli, I., Rocha, G., Rosset, C., Rossetti, M., Roudier, G., Rouillé d´Orfeuil, B., Rowan-Robinson, M., Rubiño-Martín, J. A., Rusholme, B., Said, N., Salvatelli, V., Salvati, L., Sandri, M., Santos, D., Savelainen, M., Savini, G., Scott, D., Seiffert, M. D., Serra, P., Shellard, E. P. S., Spencer, L. D., Spinelli, M., Stolyarov, V., Stompor, R., Sudiwala, R., Sunyaev, R., Sutton, D., Suur-Uski, A.-S., Sygnet, J.-F., Tauber, J. A., Terenzi, L., Toffolatti, L., Tomasi, M., Tristram, M., Trombetti, T., Tucci, M., Tuovinen, J., Türler, M., Umana, G., Valenziano, L., Valiviita, J., Van Tent, F., Vielva, P., Villa, F., Wade, L. A., Wandelt, B. D., Wehus, I. K., White, M., White, S. D. M., Wilkinson, A., Yvon, D., Zacchei, A., and Zonca, A. (2016). Planck 2015 results - XIII. Cosmological parameters. *Astronomy & Astrophysics*, 594 :A13.

[Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, New York, USA, 1st edition.

[Pynadath and Tambe, 2002] Pynadath, D. V. and Tambe, M. (2002). Multiagent teamwork : Analyzing the optimality and complexity of key theories and models. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 2*, pages 873–880. ACM.

[Ray and Vohra, 2015] Ray, D. and Vohra, R. (2015). Coalition Formation. In *Handbook of*

Game Theory with Economic Applications, volume 4, pages 239–326. Elsevier.

[Reis et al., 2000] Reis, L. P., Lau, N., and Oliveira, E. C. (2000). Situation based strategic positioning for coordinating a team of homogeneous agents. In *Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, pages 175–197. Springer.

[Ross et al., 2008] Ross, S., Pineau, J., Paquet, S., and Chaib-draa, B. (2008). Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32(1) :663–704.

[Russell and Norvig, 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence : A Modern Approach*. Prentice Hall Press, Upper Saddle River, New Jersey, USA, 3rd edition.

[Sandholm et al., 1999] Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2) :209–238.

[Scerri et al., 2005] Scerri, P., Farinelli, A., Okamoto, S., and Tambe, M. (2005). Allocating Tasks in Extreme Teams. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS'05, pages 727–734, New York, New York, USA. ACM.

[Seuken and Zilberstein, 2007a] Seuken, S. and Zilberstein, S. (2007a). Improved Memory-bounded Dynamic Programming for Decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, UAI'07, pages 344–351, Arlington, Virginia, United States. AUAI Press.

[Seuken and Zilberstein, 2007b] Seuken, S. and Zilberstein, S. (2007b). Memory-bounded Dynamic Programming for DEC-POMDPs. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2009–2015, San Francisco, California, USA. Morgan Kaufmann Publishers Inc.

[Shani et al., 2013] Shani, G., Pineau, J., and Kaplow, R. (2013). A Survey of Point-based POMDP Solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1) :1–51.

[Shapley, 1971] Shapley, L. (1971). Cores of Convex Games. *International Journal of Game Theory*, 1(1) :11–26.

[Shapley and Shubik, 1954] Shapley, L. and Shubik, M. (1954). A Method for Evaluating the Distribution of Power in a Committee System. *The American Political Science Review*, 48(3) :787–792.

[Shehory, 2001] Shehory, O. (2001). Software Architecture Attributes of Multi-agent Systems. In *First International Workshop, AOSE 2000 on Agent-oriented Software Engineering*, pages 77–90, Secaucus, New Jersey, USA. Springer-Verlag New York, Inc.

[Shoham and Leyton-Brown, 2008] Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent Systems : Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, New York, USA.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587) :484–489.

[Silver and Veness, 2010] Silver, D. and Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, pages 2164–2172, USA. Curran Associates Inc.

[Smallwood and Sondik, 1973] Smallwood, R. D. and Sondik, E. J. (1973). The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *Operations research*, 21(5) :1071–1088.

[Smith and Simmons, 2004] Smith, T. and Simmons, R. (2004). Heuristic Search Value Iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI'04, pages 520–527, Arlington, Virginia, United States. AUAI Press.

[Spaan, 2012] Spaan, M. T. (2012). Partially observable Markov decision processes. In *Reinforcement Learning*, pages 387–414. Springer.

[Stone et al., 2010] Stone, P., Kaminka, G. A., Kraus, S., and Rosenschein, J. S. (2010). Ad Hoc Autonomous Agent Teams : Collaboration Without Pre-coordination. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 1504–1509. AAAI Press.

[Stone and Veloso, 1999] Stone, P. and Veloso, M. (1999). Task Decomposition, Dynamic Role Assignment, and Low-bandwidth communication for Real-time Strategic Teamwork. *Artificial Intelligence*, 110(2) :241–273.

[Szer et al., 2005] Szer, D., Charpillet, F., and Zilberstein, S. (2005). MAA* : A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pages 576–583, Arlington, Virginia, United States. AUAI Press.

[Tambe, 1997] Tambe, M. (1997). Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, 7 :83–124.

[Wachowskis, 1999] Wachowskis (1999). The Matrix. Warner Bros. (United States).

[Wolpert et al., 1999] Wolpert, D. H., Tumer, K., and Frank, J. (1999). Using Collective Intelligence to Route Internet Traffic. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pages 952–958, Cambridge, Massachusetts, USA. MIT Press.

[Wooldridge, 2009] Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition.

[Wooldridge, 2011] Wooldridge, M. (2011). Computational Aspects of Cooperative Game Theory. In O'Shea, J., Nguyen, N. T., Crockett, K., Howlett, R. J., and Jain, L. C., editors, *Agent and Multi-Agent Systems : Technologies and Applications*, pages 1–1, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Wooldridge et al., 1999] Wooldridge, M., Jennings, N. R., and Kinny, D. (1999). A Methodology for Agent-oriented Analysis and Design. In *Proceedings of the third annual conference on Autonomous Agents*, pages 69–76. ACM.

Formation dynamique d'équipes dans les DEC-POMDPs ouverts à base de méthodes Monte-Carlo

Résumé. Cette thèse traite du problème où une équipe d'agents coopératifs et autonomes, évoluant dans un environnement stochastique partiellement observable, et œuvrant à la résolution d'une tâche complexe, doit modifier dynamiquement sa composition durant l'exécution de la tâche afin de s'adapter à l'évolution de celle-ci. Il s'agit d'un problème qui n'a été que peu étudié dans le domaine de la planification multi-agents. Pourtant, il existe de nombreuses situations où l'équipe d'agent mobilisée est amenée à changer au fil de l'exécution de la tâche. Nous nous intéressons plus particulièrement au cas où les agents peuvent décider d'eux-même de quitter ou de rejoindre l'équipe opérationnelle. Certaines fois, utiliser peu d'agents peut être bénéfique si les coûts induits par l'utilisation des agents sont trop prohibitifs. Inversement, il peut parfois être utile de faire appel à plus d'agents si la situation empire et que les compétences de certains agents se révèlent être de précieux atouts. Afin de proposer un modèle de décision qui permette de représenter ces situations, nous nous basons sur les processus décisionnels de Markov décentralisés et partiellement observables, un modèle standard utilisé dans le cadre de la planification multi-agents sous incertitude. Nous étendons ce modèle afin de permettre aux agents d'entrer et sortir du système. On parle alors de système ouvert. Nous présentons également deux algorithmes de résolution basés sur les populaires méthodes de recherche arborescente Monte-Carlo. Le premier de ces algorithmes nous permet de construire des politiques jointes séparables via des calculs de meilleures réponses successives, tandis que le second construit des politiques jointes non séparables en évaluant les équipes dans chaque situation via un système de classement Elo. Nous évaluons nos méthodes sur de nouveaux jeux de tests qui permettent de mettre en lumière les caractéristiques des systèmes ouverts.

Mots-clés : planification sous-incertitude, DEC-POMDPs, systèmes multi-agents, systèmes ouverts

Dynamic team formation in open DEC-POMDPs with Monte-Carlo methods

Abstract. This thesis addresses the problem where a team of cooperative and autonomous agents, working in a stochastic and partially observable environment towards solving a complex task, needs toe dynamically modify its structure during the process execution, so as to adapt to the evolution of the task. It is a problem that has been seldom studied in the field of multi-agent planning. However, there are many situations where the team of agents is likely to evolve over time. We are particularly interested in the case where the agents can decide for themselves to leave or join the operational team. Sometimes, using few agents can be for the greater good. Conversely, it can sometimes be useful to call on more agents if the situation gets worse and the skills of some agents turn out to be valuable assets. In order to propose a decision model that can represent those situations, we base upon the decentralized and partially observable Markov decision processes, the standard model for planning under uncertainty in decentralized multi-agent settings. We extend this model to allow agents to enter and exit the system. This is what is called agent openness. We then present two planning algorithms based on the popular Monte-Carlo Tree Search methods. The first algorithm builds separable joint policies by computing series of best responses individual policies, while the second algorithm builds non-separable joint policies by ranking the teams in each situation via an Elo rating system. We evaluate our methods on new benchmarks that allow to highlight some interesting features of open systems.

Keywords : planning under uncertainty, DEC-POMDPs, multi-agent systems, open systems