# Planification multiagent sous incertitude orientée interactions : modèle et algorithmes

Arnaud Canu

université de Caen
Basse-Normandie

**U.F.R.** : Sciences

**ÉCOLE DOCTORALE** : SIMEM

**THÈSE**

présentée par

Arnaud Canu

et soutenue

**le** 28 novembre 2011

en vue de l'obtention du

**DOCTORAT de l'UNIVERSITÉ de CAEN**

**spécialité : Informatique et applications**

***(Arrêté du 7 août 2006)***

# Planification multiagent sous incertitude orientée interactions : modèle et algorithmes.

**MEMBRES du JURY**

| | | | |
|---|---|---|---|
| Brahim Chaib-Draa | Professeur | Université Laval (Quebec) | *(rapporteur)* |
| Gérard Verfaillie | Directeur de Recherches | ONERA | *(rapporteur)* |
| Raja Chatila | Directeur de Recherches | CNRS, Paris | |
| Eva Cruck | Resp. Recherche Innovation | DGA-DS-MRIS | |
| Bruno Patin | Chef de Projet | Dassault Aviation | |
| Joël Morillon | Resp. Technologies Amont | Thales Optronic S.A. | |
| Abdel-Illah Mouaddib | Professeur | Université de Caen | *(directeur)* |

**Thèse DGA/CNRS, financée par la Direction Générale de l'Armement (DGA), soutenue par les groupes THALES Optronics S.A. et Dassault Aviation.**

THALES DGA DASSAULT AVIATION

Mis en page avec la classe thloria.

## Remerciements

Je remercie, pour commencer, Abdel-Illah Mouaddib pour ses conseils et son encadrement durant ces trois dernières années. J'ai particulièrement apprécié notre collaboration : tu as su me conseiller lorsque c'était nécessaire, me laisser travailler librement le reste du temps. S'il est vrai, au vu de tes responsabilités chronophages, qu'il fut parfois difficile de nous voir aussi régulièrement que nous l'aurions voulu, j'ai malgré tout apprécié cette liberté, qui m'a permis de mener mes travaux comme je l'entendais. J'espère que notre collaboration à venir sera tout aussi fructueuse.

Mes remerciements vont également à la DGA (merci à Eva Crück), qui a rendu possible la réalisation de cette thèse, ainsi qu'à Joël Morillon (de Thales) et à Bruno Patin (de Dassault). Nos discussions furent très intéressantes, et auront apporté à mes travaux une coloration différente de l'habituelle approche purement académique. J'ai, grâce à vous, découvert l'intérêt qu'il y a à travailler sur des problèmes plus concrets.

Je remercie, de même, mon jury d'avoir accepté ce rôle exigeant. Merci en particulier à Brahim Chaib-Draa et Gérard Verfaillie pour avoir accepté d'être mes rapporteurs : vos remarques ont ouvert certaines questions que je ne m'étais pas forcément posées.

D'un point de vue plus personnel, je remercie les gens avec qui j'ai passé ces trois dernières années. Je pense en particulier à Lamia et Boris, avec qui j'ai partagé un bureau, ainsi que Guillaume qui nous a rejoint un an plus tard et Nicolas, le « petit dernier », Thesard-Prospect pour quelques temps encore. J'ai vraiment eu plaisir à passer ces journées avec vous. De même, merci à Benoît pour ces interminables discussions sur Gtalk, tu as été un vecteur de procrastination incomparable. Merci enfin à Abir, Mathieu, Jean-Philippe, Laetitia, Grégory, et tous les autres. Tous, vous êtes pour moi des amis, bien plus que des collègues.

Merci pour finir à ceux qui me sont les plus proches, je pense ici à mes parents, à ma soeur et à mon frère, ainsi qu'à toute ma famille. C'est probablement grâce à vous plus qu'à quiconque que j'en suis là aujourd'hui. C'est à Amélie en particulier que je souhaite dédier cette thèse : tu as fait de moi le plus heureux des hommes en acceptant de devenir ma femme l'été dernier, tu as su me re-motiver lorsque ma motivation faiblissait, tu m'as donné la force nécessaire à la réussite du doctorat.

ps : merci à Simba et Isis, mes chats, qui auront su m'empêcher de dormir le matin, ne me laissant d'autre choix que d'aller travailler...

# Table des matières

# Introduction

Savoir prendre la bonne décision au bon moment est un problème auquel toute entité intelligente doit constamment faire face. Lorsque l'on conduit une voiture par exemple, la question de la meilleur action à effectuer se pose sans arrêt : dois-je doubler à gauche ou ralentir ? Vaut-il mieux passer par Paris ou contourner la ville ? Dès l'instant où on tente de donner une autonomie à une intelligence artificielle (qu'il s'agisse d'un robot ou simplement d'un logiciel), le problème de la prise de décision devient central. On peut par exemple considérer le cas des Mars Rovers, ces robots évoluant sur Mars afin d'en analyser la surface : en raison des contraintes techniques (la distance Terre-Mars rend tout pilotage du robot impossible), de telles applications nécessitent une indépendance totale des agents (robots ou logiciels) vis-à-vis de l'humain.

Ce terme d'intelligence artificielle, particulièrement large, recoupe plusieurs thématiques de recherche. Dans le livre *Artificial intelligence : a modern approach*, Russel et Norvig définissent l'intelligence artificielle comme l'étude d'agents qui reçoivent des perceptions relatives à leur environnement, et exécutent des actions. Certains agents se contentent de réagir à leur perceptions, tandis que d'autres suivent un raisonnement visant à déterminer la meilleure action possible. Ces agents peuvent, dans certains cas, apprendre afin d'améliorer leurs connaissances initiales [1].

Il existe d'autres définitions au terme d'intelligence artificielle. Certains rapprochent intelligence artificielle et robotique, via l'étude de problèmes propres à la perception par exemple (tels que la « compréhension » d'une image, en repérant les objets, les êtres humains, etc.). On pourrait également inclure les problèmes de langage (comprendre le langage parlé, écrit, communiquer avec l'humain), la question de la représentation des connaissances (que sais-je, que puis-je en déduire), etc. En ce qui nous concerne, nous nous baserons sur la définition donnée par Russel et Norvig et étudierons le problème de la prise de décision.

Au vu de cette définition, on peut identifier trois grands axes de recherche : observer l'environnement (perception), raisonner afin de déterminer quelle action exécuter (décision) et exécuter l'action choisie (contrôle). Nous nous contenterons, dans cette thèse, d'étudier le problème de la décision. Cette courte introduction vise à introduire le contexte scientifique de nos travaux, puis l'objectif de recherche que nous nous sommes fixés et finalement la solution envisagée.

1. « *We define AI as the study of agents that receive percepts from the environment and perform actions. Each such agent implements a function that maps percept sequences to actions, and we cover different ways to represent these functions, such as reactive agents, real-time planners, and decision-theoretic systems. We explain the role of learning as extending the reach of the designer into unknown environments (...)* » , [Russell et Norvig, 2009]

# 1 Contexte scientifique

Prendre une décision est fondamentalement simple : celui qui conduit sa voiture pourra, sans raison particulière, choisir de contourner Paris plutôt que de traverser la ville par exemple, il aura ainsi pris une décision. Le fait de prendre une *bonne* décision est nettement plus dur, et soulève de nombreuses questions :

- Qu'est-ce qu'une bonne ou une mauvaise décision ?
- Existe t'il toujours une bonne décision ?
- Comment choisir entre deux bonnes décisions ?
- Savoir reconnaître qu'une décision est bonne est-il suffisent pour choisir cette décision, parmi toutes les décisions possibles ?
- etc.

En intelligence artificielle, il est courant de *planifier* ses actions, afin de déterminer les bonnes (voir les meilleurs) décisions. Ainsi, si on se donne un objectif, une bonne décision sera une décision qui nous rapproche de l'accomplissement de cet objectif. On choisira donc toujours la décision la plus efficace, pour atteindre l'objectif fixé. On étudiera notamment l'efficacité *à long terme* de chaque décision possible (je vais faire ceci, puis cela, puis cela, et j'atteindrai finalement mon objectif). Le raisonnement classique est le suivant :

1. je perçois mon environnement,
2. j'énumère toutes mes possibilités d'action, au vu de ma situation actuelle,
3. pour chaque action possible, j'étudie à quel point elle me rapproche de mon objectif,
4. je décide d'une action à exécuter (la plus efficace, pour atteindre mon objectif),
5. j'exécute cette action et reprend à l'étape 1.

On parle donc bien de planification, puisqu'il s'agit de raisonner sur les différentes suites d'actions possibles afin d'atteindre l'objectif au plus vite. Ce type d'approche se retrouve dans de nombreuses applications concrètes, qu'il s'agisse de robotique (avec l'exemple classique des Mars Rovers, pour l'exploration de Mars) ou d'applications logicielles. Prenons un exemple de la vie courante : la réservation d'un voyage sur internet. En réservant ce voyage, les agences en ligne vont proposer un hôtel pour passer la nuit, un billet de train ou d'avion pour se rendre sur place, un service de taxi pour atteindre l'hôtel... Pourtant, aucun de ces choix n'est trivial : quel hôtel choisir, quel type de transport emprunter pour se rendre sur place, etc. ? Il s'agit là d'un problème relativement « simple » de planification, mais on peut envisager des situations nettement plus complexes, pour lesquelles les techniques actuelles sont insuffisantes.

Ainsi, dans cette thèse, nous avons choisi d'étudier une catégorie de problèmes pour lesquels il n'existe pas, actuellement, de méthode de décision suffisamment performante. Il nous a donc fallu identifier ces problèmes (qu'est-ce qui rend un problème de planification difficile ?), puis en extraire un ensemble de caractéristiques permettant de simplifier leur résolution, afin de proposer finalement une méthode de décision applicable à des problèmes réels. Dans cette optique, les

groupes Dassault Aviation et Thales TOSA ont fourni un cadre industriel à nos travaux de recherche, notamment via deux applications concrètes :

1. *Le problème de Thales* - il s'agit ici de rendre autonome un convoi (c'est-à-dire un ensemble de véhicules) devant se rendre à un endroit donné. Les véhicules, au sein de ce convoi, doivent respecter certaines contraintes. Ils devront par exemple éviter toute collision, respecter une formation particulière, etc. Notre objectif était donc de permettre à ces véhicules de se « piloter » automatiquement, sans intervention de l'humain, tout en respectant ces différentes contraintes. Il s'agit bien d'un problème de planification, puisqu'il faut étudier l'impact à long terme de chaque décision (typiquement, si un véhicule fait un écart pour éviter une collision, il faut s'assurer qu'il ne se mettra pas sur la trajectoire d'un autre véhicule, ce qui engendrerait une seconde collision après quelques instants).
2. *Le problème de Dassault* - il s'agit de contrôler des avions au sol, lorsqu'ils se déplacent sur un aéroport afin d'atteindre leur piste de décollage. Ces avions doivent, là encore, respecter certaines contraintes (tel avion doit décoller en premier, deux avions ne peuvent être trop proches l'un de l'autre, etc.). Il s'agit donc bien à nouveau d'un problème de planification, dans lequel chaque avion doit prévoir une trajectoire compatible avec l'ensemble des autres avions, tout en prenant en compte le risque qu'un événement extérieur vienne perturber le trafic (un orage par exemple, rendant certaines pistes impraticables).

Plusieurs verrous scientifiques rendent ces problèmes particulièrement complexes. Il y a, pour commencer, les problèmes propres à la robotique : les robots (qu'il s'agisse des véhicules au sein du convoi, des avions sur la piste ou d'une autre application) sont imparfaits :

- lorsqu'un robot exécute une action, celle-ci peut échouer ou produire un résultat légèrement différent de celui attendu (les roues d'un véhicule, par exemple, peuvent « patiner », maintenant le robot sur place alors que celui-ci avait prévu d'avancer),
- de même, un robot observant son environnement peut recevoir des perceptions erronées ou incomplètes. Une caméra, par exemple, ne donnera pas au robot une vision parfaite du monde dans lequel il évolue : certaines parties ne seront pas observées (ce qui est derrière un mur par exemple), et l'image de la caméra pourra être bruitée.

Il y a également le problème de la décision multiagent : dès l'instant où on manipule non-plus un agent, mais un groupe (par exemple, le convoi), il faut prendre en compte les décisions de chacun afin que l'un n'entre pas en conflit avec l'autre. Il faut également prendre en compte les capacités de calcul limitées des robots : il faut parfois réaliser certains calculs « offline », sur une machine puissante, avant d'embarquer le résultat au sein du robot (d'autant plus qu'on attend des réactions en « temps-réel » de la part du robot).

## 2 Le problème de la décision décentralisée sous incertitude

Nous avons finalement identifié une classe particulière de problèmes à traiter : il s'agit des problèmes de décision *décentralisée*, et *sous incertitude*. La décision sous incertitude représente

une branche à part entière du domaine général qu'est la planification. On s'intéresse ici aux problèmes pour lesquels tout n'est pas maîtrisable. On a vu comment les robots pouvaient commettre des erreurs, en exécutant leurs actions (roue qui glisse) : il faut alors prendre en compte cette possibilité, dans le processus de décision. Imaginons par exemple deux actions, ayant à priori le même effet, mais dont l'une des deux a un risque élevé d'échouer. Dans une telle situation, il sera préférable d'exécuter l'action ne pouvant pas échouer... Il s'agit donc bien de planifier ses actions, en prenant en compte l'incertitude quant à l'exécution de celles-ci. Il est en général plus difficile de décider sous incertitude, que dans un problème totalement maîtrisé : il y a en effet beaucoup plus de situations possibles à prendre en compte (chaque action ayant plusieurs résultats possibles, il faut tous les prendre en compte).

Il n'est pas toujours possible, dans ce genre de problème, de fixer un but unique à atteindre. Prenons l'exemple du problème de Thales (gestion de convoi) : les agents doivent se déplacer vers l'objectif, maintenir la formation voulue, éviter les collisions avec les autres véhicules... On a alors un ensemble de buts devant être atteints. Dans certains cas, il faudra atteindre la totalité des buts. Dans d'autres cas, il pourra y avoir des buts critiques (éviter les collisions) et d'autres moins importants (maintenir la formation). On choisit en général de représenter ces buts par un ensemble de récompenses et de coûts : chaque situation jugée « positive » apportera une récompense à l'agent, plus ou moins élevée selon le but atteint, et chaque situation négative (par exemple, une collision) sera coûteuse pour l'agent. On planifiera alors des actions permettant de maximiser les récompenses obtenues et de minimiser les coûts : le modèle des processus de décision markoviens, ou MDP, permet de représenter ce type de problèmes et de calculer facilement (complexité dans P) une décision optimale.

L'usage du modèle MDP implique toutefois le respect d'une hypothèse implicite : on considère que le problème traité est complètement observable. L'agent doit donc pouvoir percevoir la totalité de son environnement (on imaginera par exemple une « super-caméra », capable de tout voir et tout connaître), ce qui lui permet de prendre des décisions optimales au vu de la situation dans laquelle il se trouve. Pourtant, une telle hypothèse d'observabilité totale est rarement vérifiée. Dans la plupart des applications robotiques par exemple, l'agent doit se contenter des capteurs qu'il embarque (sonar, caméra, etc.) pour observer son environnement, dont il n'a alors qu'une vision partielle. On utilise alors le modèle POMDP (MDP partiellement observable) pour décrire le problème, ce qui permet à l'agent de raisonner à partir des informations partielles dont il dispose. Le temps nécessaire pour prendre une décision augmente alors considérablement.

On traite donc des problèmes dans lesquels on maximise les récompenses, tout en prenant en compte l'incertitude sur l'exécution des actions et l'observabilité partielle sur l'environnement. Il s'agit là d'un problème typique de décision sous incertitude. Ces problèmes sont complexes, mais la difficulté augmente encore lorsque l'on manipule non-plus un seul agent, mais un ensemble d'agents. En général, chaque agent est responsable du choix de ses actions : on parle alors d'exécution décentralisée, puisqu'il n'y a pas d'élément central pouvant piloter l'ensemble du groupe. Ces problèmes multiagents sont particulièrement difficiles, puisque chaque agent doit, avant de prendre une décision, émettre des suppositions quant aux actions qui seront choisies par

les autres agents (on ne connaît pas, à priori, leurs intentions). De plus, l'observabilité partielle implique en général que l'agent ne connaît pas avec certitude l'état des autres agents. Cela augmente donc la difficulté qu'il y a à émettre des suppositions sur leurs actions à venir. Ainsi, on représente en général ce type de problèmes via le modèle DEC-POMDP, pour lequel le temps nécessaire avant de prendre une décision est doublement exponentiel (notamment en le nombre d'agents). Il s'agit donc d'un modèle extrêmement complexe à manipuler.

# 3 Objectif de recherche

Nous nous sommes intéressé à la résolution de problèmes de décision décentralisée, sous incertitude et en environnement partiellement observable (DEC-POMDP). Comme nous le montrerons dans la suite de ce document, la plupart des approches existantes ne permettent pas de traiter ce type de problèmes, à moins de se limiter à certaines sous-catégories de problèmes bien particulières. Nous nous sommes fixés, pour la résolution de ces problèmes, plusieurs objectifs :

1. *Prendre en compte l'observabilité partielle* - nous l'avons expliqué précédemment, l'agent dépend de ses capteurs pour observer la situation dans laquelle il se trouve. Il y a donc des choses qu'il ignore, qu'il s'agisse de son environnement ou de l'état des autres agents. Il faudra donc employer une approche permettant de gérer cette observabilité partielle.

2. *Prendre en compte les autres agents* - l'agent évolue au milieu d'un groupe. Afin d'optimiser son comportement, il faudra non-seulement étudier l'impact de ses actions sur son état, mais également sur l'état des autres agents (va-t'on les aider, les pénaliser ?). De même, il faudra envisager les différentes actions pouvant être exécutées par les autres agents, afin de voir si l'un d'eux ne peut pas nous aider (y'a t'il une action qui ne peut être accomplie qu'avec l'aide d'un autre agent ?). L'exécution étant décentralisée, il faudra faire des prévisions sur les actions des autres agents, tout en intégrant la possibilité que ceux-ci prennent une décision que l'on n'avait pas prévue.

3. *S'assurer que l'on prend des bonnes décisions* - il ne suffit pas de prendre des décisions, il faut également que celles-ci soient bonnes, voir optimales. Ainsi, on tentera de maximiser l'espérance de gain, c'est-à-dire de maximiser les récompenses et de minimiser les coûts. On pourra par exemple comparer les décisions prises par l'agent avec les décisions qu'aurait pris un agent « idéal », bénéficiant d'une observabilité totale sur son environnement.

4. *Assurer une prise de décision suffisamment rapide* - nous avons pour objectif de pouvoir traiter des applications « réelles », telles que celles fournies par Thales et Dassault. Il faut donc pouvoir prendre des décisions rapides (le robot évoluant au sein d'un convoi par exemple, ne peut pas se permettre de s'arrêter quelques minutes avant chaque décision). Ainsi, l'approche choisie devra permettre l'exécution de tous les calculs « lourds » durant une étape préliminaire au déroulement de la mission, afin de rendre possible la prise de décision en temps-réel.

5. *Résoudre les problèmes de Thales et Dassault* - ces problèmes ayant guidé notre réflexion durant la réalisation de cette thèse, il sera intéressant de voir les résultats obtenus sur ceux-ci. Actuellement, aucune approche existante n'est à même de les résoudre : ils serviront donc de « validation » à nos travaux, de par leur complexité élevée.

Nous avons donc un objectif de recherche clairement établi : permettre une prise de décision décentralisée « de qualité » pour des problèmes multiagents, sous incertitude et en environnement partiellement observable. Il faudra de plus que les résultats soient suffisamment « ouverts » pour pouvoir s'appliquer aux problèmes complexes fournis par Thales et Dassault.

## 4 Solution proposée

Nous sommes partis d'une évidence : l'être humain, dans sa vie de tous les jours, doit constamment prendre des décisions (qu'il espère de qualité), alors qu'il n'a qu'une connaissance partielle du monde dans lequel il évolue et qu'il doit prendre en compte les autres humains avec qui il entre en interaction. Nous avons alors tenté de s'inspirer du comportement qu'a cet humain, afin de permettre la prise de décision pour nos agents.

Lorsque l'agent évolue dans son environnement, il commence par observer celui-ci. Il obtient donc des informations partielles quant à sa situation, mais aussi concernant les autres agents. Pour être exact, l'agent observe rarement la totalité de la population au sein de laquelle il évolue, mais plutôt une portion de cette population (agents que l'on nommera *voisins*). De même, un humain évoluant dans une foule par exemple n'observe que ses voisins (il ne s'intéresse pas à l'état ni aux actions du reste de la foule). Cet humain va alors, pour se déplacer, éviter d'entrer en collision avec ces voisins.

On peut appliquer ce comportement à notre méthode de décision : l'agent qui doit choisir une action à exécuter n'a pas forcément besoin de prendre en compte la totalité de la population au sein de laquelle il évolue. Il peut, au contraire, se contenter d'analyser l'impact de ses actions sur ses voisins. Cette notion de voisinage est à prendre au sens large : dans le cas du convoi par exemple, il s'agira bien d'un voisinage au sens géographique du terme, mais tout critère de voisinage pourra être acceptable (sur le problème de Dassault par exemple, deux avions seront « voisins » s'ils doivent décoller l'un après l'autre, peut importe leur position géographique). Il s'agira alors, pour déterminer le voisinage, d'étudier les *interactions* entre agents : un agent a-t'il une influence sur mon état ? Ai-je une influence sur le sien ? Si oui, nous sommes en interaction, ce qui en fait un de mes voisins.

Revenons à l'humain, qui se déplace au sein d'une foule. Celui-ci peut, potentiellement, entrer en interaction avec n'importe qui, et ce n'importe quand. Pourtant, durant son déplacement, l'humain n'entre en interaction qu'avec quelques autres personnes, et seulement à certains moments. De la même façon, il faudra prévoir qu'un agent puisse entrer en interaction avec n'importe quel autre agent impliqué dans le problème, et ce dans n'importe quel état, alors même qu'à l'exécution ces interactions seront certainement rares.

Il faudra donc, pour résumer, permettre à l'agent de planifier ses actions, tout en prenant en compte l'impact qu'auront les interactions sur l'intérêt de chaque action possible. Cette notion d'interaction sera centrale dans notre approche : plus le nombre d'interactions à prendre en compte au même moment sera élevé, et plus il sera dur de prendre une décision. Là encore, on peut se baser sur l'humain pour déterminer comment agir : lorsque celui-ci évolue au sein d'une foule, il ne considère que très peu d'interactions à la fois. Si l'humain avance par exemple, il n'a pas besoin de prendre en compte la présence d'une autre personne derrière lui. De même, nous minimiserons les interactions prises en compte par l'agent durant son processus de décision.

Nous proposerons donc un modèle, dérivé de l'approche DEC-POMDP, permettant d'intégrer explicitement les interactions entre agents. Nous proposerons également un ensemble d'algorithmes, pour le calcul d'une politique de comportement à partir de ce modèle. Nous verrons alors que cette approche permet de traiter des problèmes jusqu'à présent non-résolus (tout d'abord en augmentant la dimension de certains benchmarks classiques du domaine, puis en traitant une application industrielle fournie par Thales).

## 5 Organisation du document

Ce document s'articule autour de trois grande parties, que nous présentons ici brièvement.

**Partie I - État de l'art**

Cette première partie propose une analyse préliminaire à nos travaux de recherche. Ainsi, nous commencerons par y décrire les principaux éléments nécessaires à la bonne compréhension de nos travaux : nous rappellerons brièvement quelques généralités autour de la notion de planification, avant de présenter plus en détails le domaine qui nous intéresse, c'est-à-dire la planification sous incertitude via les MDPs. Nous en viendrons rapidement au modèle DEC-POMDP que nous avons choisi d'utiliser, ce qui nous permettra de présenter plus en détails les travaux existants relatifs à ce domaine d'étude. Ceux-ci étant particulièrement nombreux, cet état de l'art ne se veut pas exhaustif. Il s'agira plutôt d'une présentation des grandes tendances, afin de comprendre les avantages et inconvénients de chaque approche existante.

Nous présenterons principalement, dans cette partie, les travaux relatifs à l'usage des interactions dans la prise de décision, ceux-ci étant directement corrélés à nos recherches. Cela nous permettra de mettre en avant les limitations de ces travaux, principalement en ce qui concerne leur applicabilité. En effet, les travaux les plus prometteurs dans le domaine reposent tous sur des hypothèses fortes quant aux interactions possibles entre les agents. Une hypothèse souvent effectuée consiste par exemple à limiter ces interactions, en spécifiant que tel agent ne peut être en interaction qu'avec tel autre, ou seulement dans tel état, etc. Il est également courant de supposer une observabilité très forte sur le voisinage (voir totale), ce qui est rarement vérifié dans une application « réelle ». Nous en déduirons finalement que les approches existantes sont insuffisantes pour traiter des problèmes tels que ceux fournis par Thales et Dassault.

**Partie II - DyLIM : un modèle d'interactions locales et dynamiques**

Dans cette seconde partie, nous montrerons comment l'agent peut représenter ses connaissances sur le monde et sur son voisinage. Les approches existantes ne nous permettant pas de traiter les problèmes complexes fournis par Thales et Dassault, nous proposerons notre propre façon de modéliser l'environnement de l'agent. Ainsi, nous commencerons cette partie par une analyse des éléments nécessaires à une bonne représentation de l'environnement et des interactions entre agents. Nous en déduirons les caractéristiques d'un modèle « idéal », permettant de représenter tout problème de décision sous incertitude, tout en intégrant la notion d'interactions.

Après cette analyse, nous proposerons DyLIM (Dynamic Local Interaction Model), un modèle théorique correspondant à notre définition du modèle idéal. DyLIM propose une autre façon de modéliser un problème de type DEC-POMDP, en intégrant une description explicite des interactions, afin de faciliter le processus de résolution. Nous réaliserons tout d'abord une présentation formelle de ce modèle, de chacun de ses composants et de la façon dont on peut l'utiliser pour représenter un problème réel (nous verrons notamment son instanciation au problème de Thales). Nous verrons alors que ce modèle bénéficie d'une applicabilité forte, tout en permettant d'exploiter les interactions entre agents pour simplifier la représentation du problème.

**Partie III - Algorithmes de résolution**

Cette troisième et dernière partie traitera du cœur du problème : la prise de décision. Ainsi, nous y présenterons un ensemble d'algorithmes, permettant de planifier les actions de l'agent en se basant sur le modèle DyLIM introduit précédemment. Nous donnerons les algorithmes en question et proposerons plusieurs méthodes possibles, selon le type de problème traité. Nous verrons notamment que certains problèmes permettent une résolution optimale tandis que d'autres, plus complexes, nécessitent une résolution approchée.

Nous proposerons finalement une analyse de complexité de nos algorithmes, ainsi qu'un ensemble d'expérimentations visant à montrer la qualité des décisions effectuées par les agents. Nous verrons alors que ces algorithmes peuvent passer à l'échelle et résoudre de manière approchée des problèmes de taille « réelle », tout en produisant des comportements de bonne qualité. Nous montrerons finalement les résultats obtenus sur le problème de Thales, afin de valider l'applicabilité de notre approche.

# Première partie

# État de l'art

« *Réfléchis avec lenteur, mais exécute rapidement tes décisions.* »
Isocrate, vers 400 av. J.-C., qui avait tout compris aux MDPs...

# Introduction à la partie I

La notion d'intelligence artificielle regroupe de nombreux problèmes, qu'il s'agisse de robotique, d'analyse d'informations, de cognition, de problèmes de mémorisation... Les problèmes étudiés dans cette thèse s'inscrivent dans le domaine, plus réduit, de la prise de décision (et plus particulièrement dans le cadre multiagent). Nous allons, dans cet état de l'art, définir ces problèmes de façon formelle : pour cela, nous commencerons par introduire un certain nombre d'éléments classiques du domaine, puis nous introduirons progressivement des données supplémentaires pour arriver finalement à une description précise du cadre auquel nous nous sommes intéressés. Ce cadre de la décision multiagent étant particulièrement large, il existe plusieurs façons de nommer et représenter les éléments que nous allons être amenés à manipuler. Nous insisterons donc tout particulièrement sur le sens que nous donnons, dans cette thèse, aux différentes notions propres au domaine. Ainsi, cet état de l'art n'a pas pour vocation de répertorier la totalité des approches existantes, mais plutôt de fixer le cadre formel de nos travaux afin non seulement de mettre ceux-ci en contexte, mais aussi et surtout d'éviter toute mauvaise compréhension des éléments présentés.

Nous commencerons, dans le premier chapitre, par donner une définition des notions « de base » employées par la suite (principalement : agent, environnement et mission), ce qui nous permettra de comprendre en quoi le problème étudié s'inscrit dans le cadre général de l'intelligence artificielle. Une fois ces éléments introduits, nous présenterons la notion de planification, c'est-à-dire le choix non pas d'une action à effectuer, mais d'une séquence d'actions permettant d'accomplir une mission donnée. Nous terminerons par une présentation du domaine qui nous intéresse, à savoir la planification probabiliste (lorsque l'évolution du problème n'est pas totalement maîtrisée mais soumise à une part d'incertitude).

Le second chapitre est consacré aux différents formalismes possibles de planification probabiliste. Nous présenterons tout d'abord le modèle des processus de décision markoviens (MDP), et la façon dont on peut utiliser un MDP pour représenter le problème de décision d'un agent donné. Nous verrons en quoi ce modèle est parfaitement adapté aux types de problèmes que l'on traite, et la façon dont on peut se baser sur un MDP pour planifier les actions de l'agent. Nous nous intéresserons ensuite à une généralisation des MDPs aux situations où l'agent n'a qu'une connaissance partielle du monde dans lequel il évolue. Nous présenterons finalement le cas multiagent, là encore via une généralisation des MDPs. Nous verrons notamment pourquoi cette généralisation augmente la difficulté du problème à traiter.

Le troisième chapitre de cet état de l'art introduit le modèle des processus de décision markoviens partiellement observables et décentralisés (DEC-POMDP). Celui-ci offre la représentation la plus générique possible pour un MDP, puisqu'il permet de considérer à la fois une connaissance partielle du monde et une situation multiagent. Nous donnerons tout d'abord une description du modèle en lui-même, puis nous verrons qu'il existe plusieurs classes de problèmes représentables par ce modèle, selon que l'on admette ou non un certain nombre d'hypothèses. Nous verrons alors plusieurs modèles dérivés des DEC-POMDPs, tirant parti de ces hypothèses.

Nous verrons finalement, dans un dernier chapitre, comment planifier les actions des agents à partir d'un DEC-POMDP. Nous ferons un tour d'horizon des approches existantes, en mettant en avant les avantages et inconvénients de chacune. Nous montrerons alors qu'aucune approche réellement efficace n'existe pour le cas le plus général, lorsqu'aucune hypothèse n'est émise au sein du DEC-POMDP. Cette constatation sera le point de départ de nos travaux, présentés dans le reste de ce document.

# Chapitre 1

# Généralités autour de la planification

**Sommaire**



Avant de présenter la planification à proprement parler, il est nécessaire d'introduire un certain nombre d'éléments. Nous débuterons donc ce chapitre par une présentation des concepts de base de la planification, après quoi nous présenterons les deux domaines principaux que sont la planification déterministe et probabiliste.

## 1.1 Concept de planification

Nous considérons, dans cette thèse, la planification comme le fait pour un agent ou un ensemble d'agents de calculer un plan (une séquence d'actions) ou une politique de comportement permettant d'accomplir une mission donnée, au sein d'un environnement en particulier.

### 1.1.1 Notion d'agent

Il existe de nombreuses définitions pour le terme d'agent, et plusieurs synthèses en ont été réalisées [Beynier, 2006]. Russell et Norvig définissent un agent [Russell et Norvig, 2009] comme « simplement quelque chose qui agit »[2]. Cette définition est un bon point de départ mais s'avère

2. « An agent is just something that acts. »

insuffisante dès lors que l'on s'intéresse aux agents en informatique (ce qui est bien entendu notre cas). Russell et Norvig ajoutent d'ailleurs à leur définition qu'un agent informatique doit répondre à un certain nombre de critères [3], notamment être capable de prendre des décisions de manière autonome ou encore de percevoir son environnement et de s'adapter aux changements afin d'atteindre un objectif donné.

Ces notions d'agent autonome, d'environnement ou encore d'objectif sont présentes dans la définition de Wooldridge et Jennings [4] [Wooldridge et Jennings, 1995], qui est celle que nous retiendrons dans cette thèse.

**Définition 1 (Agent)** *Un agent est un système informatique situé dans un environnement et capable d'agir de manière autonome dans cet environnement afin d'atteindre les objectifs pour lesquels il a été conçu.*

Ainsi, un agent est considéré comme **situé** au sein de son environnement. Ce concept d'environnement sera détaillé par la suite : considérons pour l'instant qu'il s'agit tout simplement du monde dans lequel l'agent évolue. Le fait que l'agent soit situé et capable d'agir dans cet environnement implique un certain nombre de choses. Tout d'abord, l'agent doit être capable d'observer son environnement : lorsqu'un être humain se déplace, il voit des objets, entend des bruits, sent des odeurs, etc. De la même façon, l'agent est muni de **capteurs** : qu'il s'agisse d'une caméra montée sur un robot, ou de données analysées par un logiciel, les capteurs fournissent un ensemble de perceptions. Ce sont ces perceptions qui permettent à l'agent d'analyser la situation, afin de prendre la décision la plus adaptée.

Prendre une décision se manifeste systématiquement par l'exécution d'une action. Un robot pourra par exemple faire le choix de se déplacer, ou de saisir un objet, tandis qu'un logiciel pourra par exemple choisir d'afficher quelque chose à l'écran. Dans tous les cas, l'agent agit par le biais de ses **effecteurs** (roues, pinces, écran, etc.). On voit alors se dessiner le comportement de base de tout agent, que nous nommerons **boucle de vie**.

**Définition 2 (Boucle de vie)** *La boucle de vie de tout agent consiste en la répétition de trois éléments successifs : observer, raisonner, agir.*

Cette définition (voir figure 1.1) correspond au comportement d'un agent **autonome**, c'est à dire capable d'évoluer dans son environnement sans aucune intervention de l'humain. Un agent n'est pas nécessairement complètement autonome : l'agent peut être semi-autonome (piloté par l'humain lors de phases critiques), voir complètement piloté [Goodrich *et al.*, 2001]. À l'inverse, un agent peut apprendre de ses expériences passées : il sera alors capable d'améliorer progressi-

3. ≪ But computer agents are expected to have other attributes that distinguish them from mere programs, such as operating under autonomous control, perceiving environment, persisting over a prolonged period, adapting to change and being capable of tacking on another's goals. ≫

4. ≪ An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objectives. ≫

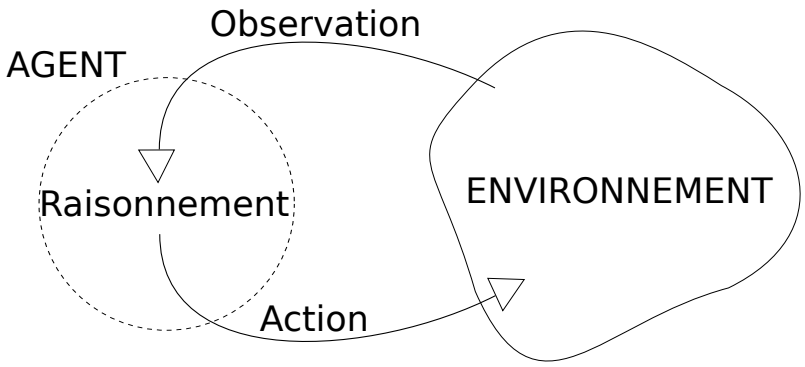


FIGURE 1.1 – Boucle de vie d'un Agent

vement son comportement. En ce qui nous concerne, nous nous intéresserons au cas d'un agent complètement autonome, mais qui ne possède pas la capacité d'apprendre.

La boucle de vie de l'agent comporte une phase de raisonnement : c'est cette phase qui doit permettre à l'agent de planifier ses actions, afin d'atteindre ses objectifs. Intervient ici la notion de **rationalité** : un agent rationnel est tel qu'il prendra les meilleurs décisions possibles. Il est évident qu'à un instant donné, certaines actions seront préférables à d'autres (un agent au bord d'un précipice évitera par exemple de continuer d'avancer) : l'agent rationnel sera à même d'identifier et choisir ces actions. Afin d'estimer l'intérêt d'une action, l'agent a besoin d'une **mesure de performance**, c'est à dire d'un critère qui estime si une action est bonne ou non : on peut alors introduire la notion de **comportement optimal**.

**Définition 3 (Comportement optimal)** *Un comportement est optimal si, à tout moment, l'agent choisit l'action qui maximise sa mesure de performance.*

Au delà de cette notion de rationalité, on peut également spécifier le type de raisonnement employé par l'agent : **réactif**, ou **pro-actif**. Un agent réactif sera tel que chaque observation entraînera une réaction de l'agent, sans nécessiter de temps de réflexion. Ce type d'agent, ayant un comportement plutôt « réflexe », sera particulièrement rapide à prendre des décisions mais aura plus de difficultés à atteindre un comportement optimal. À l'inverse, un agent pro-actif sera tel qu'il anticipera les événements à venir, afin d'agir non pas en réaction à ses observations mais en prévisions des observations futures. L'agent pro-actif aura alors un raisonnement à long terme de qualité, mais nécessitera parfois d'importants moments de réflexion. Finalement, il existe aussi des agents mixtes, adoptant un comportement réactif lorsque cela s'avère nécessaire (éviter un obstacle pour un robot) et pro-actif lorsque c'est possible (réfléchir au meilleur chemin à emprunter pour se rendre à un point donné).

### 1.1.2 Environnement et mission

On considère qu'un agent est situé dans un environnement donné : cet environnement représente le monde dans lequel l'agent évolue. Pour un robot, l'environnement sera souvent le monde extérieur, ou une portion de ce monde (une ville en particulier par exemple). Pour un logiciel, l'environnement pourra être un ordinateur, Internet, etc. Un logiciel de domotique par exemple aura pour environnement une maison qu'il observera via des thermomètres, caméras, et dans lequel il agira via le chauffage, la télévision, etc. Parfois, les capteurs dont dispose l'agent ne seront pas suffisants pour observer la totalité de l'environnement (un robot par exemple ne verra pas ce qui se passe de l'autre côté d'une porte), on parle alors **d'observabilité partielle**.

**Définition 4 (Observabilité)** *L'observabilité d'un agent décrit la portion de l'environnement qu'il est à même de percevoir par ses propres moyens. On parle d'observabilité partielle ou totale.*

Nous reviendrons sur la question de l'observabilité plus loin dans cet état de l'art. Pour l'instant, considérons que l'on parle d'un environnement complètement observable. Dans la plupart des cas, cela permettra à l'agent de connaître à tout moment la situation dans laquelle il se trouve. Il existe cependant des environnements **dynamiques**, c'est à dire tels que certains éléments peuvent évoluer indépendamment de toute action de l'agent : ainsi, l'agent pourra observer l'environnement, puis raisonner pour prendre une décision, mais réaliser au moment d'exécuter cette décision que l'environnement a évolué et que l'action choisie n'est plus optimale.

**Définition 5 (Environnement statique/dynamique)** *Un environnement est dit statique si il n'évolue pas tant que l'agent n'exécute aucune action. Sinon, il est dynamique.*

Qu'il soit ou non statique, l'environnement peut également s'avérer **stochastique** : cela signifie que les actions des agents n'ont pas toujours l'effet attendu. A l'opposé, un environnement **déterministe** sera tel que l'agent pourra prédire avec certitude l'effet de ses actions.

**Définition 6 (Déterminisme)** *Un environnement est dit stochastique, ou non-déterministe, lorsque l'exécution d'une action dans une situation donnée ne produit pas systématiquement le même résultat mais est soumis à une distribution de probabilités parmi un ensemble de possibilités. À l'inverse, un environnement dans lequel on connaît à l'avance le résultat est déterministe.*

Dans le cadre de cette thèse, nous considérerons l'environnement comme étant statique, mais stochastique et partiellement observable. De telles suppositions sont assez classiques lorsque l'on tente de décrire un problème « réel ». En effet, supposer l'environnement statique est en général acceptable dès lors que l'on contrôle l'ensemble des agents, tandis que supposer l'environnement déterministe impliquerait que l'agent dispose d'effecteurs parfaits (ce qui est rarement le cas : les roues d'un robot ont par exemple tendance à déraper). Au delà de ces trois critères intrinsèques à l'environnement (observabilité, staticité et stochasticité), on peut également caractériser celui-ci par la façon dont l'agent le perçoit : **discret**, ou **continu**. Pour des raisons de faisabilité, nous choisissons de traiter l'environnement comme étant discret.

**Définition 7 (Environnement discret)** *Un environnement est dit « discret » si il existe un nombre fini de perceptions et d'actions possibles. Sinon, il est continu.*

L'agent, situé dans son environnement, a une mission (ou un ensemble de buts) à accomplir. En général, accomplir sa mission signifie faire évoluer l'environnement vers l'état voulu. L'agent devra donc raisonner de façon à déterminer l'action (ou la séquence d'actions) lui permettant d'atteindre, le plus vite possible, cet état but. On parle alors de planification.

## 1.2 Cadre formel

La planification désigne le procédé par lequel l'agent va calculer l'action, ou la séquence d'actions, qui lui permettra d'atteindre son but. Nous nous contenterons ici de présenter les notions de base nécessaires à la compréhension de notre domaine d'étude : la planification stochastique (c'est-à-dire en environnement non-déterministe).

### 1.2.1 Notion d'état, d'action et de transition

Nous avons, dans la section précédente, défini les notions de base que sont l'agent et l'environnement. Nous avons également vu, intuitivement, que l'agent prend des décisions en fonction de la situation dans laquelle il se trouve et que ces décisions font évoluer la situation. Nous allons maintenant introduire ces notions de façon plus formelle.

#### État

L'agent, lorsqu'il évolue dans son environnement, a besoin d'une façon de représenter cet environnement afin de prendre ses décisions. Un humain entrain de conduire une voiture par exemple, se dira « je suis sur l'Autoroute 13, il y a une voiture à environ 10 mètres devant moi, je roule à 130 km/h... ». Ces informations ne constituent pas nécessairement l'ensemble de l'environnement, mais plutôt la représentation que s'en fait l'agent. Il existe plusieurs façons de représenter ces informations, mais la plus couramment utilisée est la notion d'état.

**Définition 8 (État)** *On appelle **état** l'ensemble des informations dont dispose un agent, à un instant donné.*

Traditionnellement, l'état décrit à la fois l'environnement et l'agent en lui même. Ainsi, si on prend à nouveau l'exemple de l'automobiliste, l'état décrira non seulement sa situation sur la route (vitesse, position, etc.), mais aussi la quantité d'essence restante, son niveau de fatigue, etc. Idéalement, l'état regroupe l'ensemble des informations dont l'agent peut avoir besoin pour prendre une décision.

D'un point de vue formel, il existe plusieurs façons de représenter un état. On peut imaginer par exemple un ensemble de variables (vitesse, position, essence). On peut également imaginer disposer d'un ensemble fini de situations $(s_1, s_2, \dots)$, tel que chaque $s_i$ représente un état donné, une telle représentation s'appliquant surtout à des problèmes non décomposables en variables.

#### Action

Nous avons expliqué que, lors de sa boucle de vie, l'agent prend des décisions en fonction de la situation actuelle. Littéralement, cela signifie qu'à tout état $s$, l'agent associe une action $a$ à exécuter. On aura donc, en général, un ensemble $(a_1, a_2, \dots)$ d'actions possibles (par exemple : accélérer, ralentir, tourner, etc.). On pourra de plus supposer que toute action est toujours applicable, ou au contraire que l'ensemble des actions applicables dépende de l'état actuel.

#### Transition

Lorsque l'agent exécute une action, il modifie en général son état. L'automobiliste qui accélère par exemple, va voir sa position changer ainsi que sa vitesse, son niveau de carburant, etc. Cette modification de l'état est appelée **transition**.

**Définition 9 (Transition)** *On appelle **transition** l'acte de passer d'un état $s$ à un état $s'$, via l'application d'une action $a$.*

Selon les cas, cette transition pourra se traduire par une modification de certaines variables (la vitesse passe de 110 à 130), ou par le passage d'une situation à une autre (on passe de $s_7$ à $s_3$). Les transitions en environnement stochastique impliquent de plus certaines particularités que nous détaillerons par la suite.

### 1.2.2 Utilité et agent rationnel

On a vu précédemment que l'agent était amené à raisonner, pour choisir le comportement à adopter au vu de son environnement (figure 1.1, page 15). Ce raisonnement implique que l'agent ait un objectif à atteindre, sans quoi n'importe quelle action serait acceptable, à tout moment. Une représentation simple de cet objectif consisterait à fournir à l'agent un but à atteindre (un état dans lequel l'agent doit se rendre par exemple). On peut également envisager une représentation plus « générale », en associant à chaque état :

- un gain, décrivant ce que l'agent gagne lorsqu'il arrive dans cet état (en général, une fonction $g$ associant à tout état $s$ un réel positif ou nul $g(s) \geq 0$),
- un coût, décrivant ce que l'agent perd en arrivant dans cet état (en général, une fonction $c$ associant à tout état $s$ un réel négatif ou nul $c(s) \leq 0$).

Cette représentation est plus générale, puisque l'on peut l'utiliser pour décrire un état but (avec $g(but) > 0$ et $\forall s \neq but, g(s) = 0$) ou des problèmes plus complexes dans lesquels plusieurs états sont intéressants, sans que l'on puisse déterminer un but en particulier. On peut alors introduire la notion d'utilité.

**Définition 10 (Utilité)** *Une **fonction d'utilité** $U$ est une fonction qui, à tout état $s$, associe un réel $U(s)$ représentant le degré de satisfaction de l'agent lorsqu'il arrive dans cet état.*

Ainsi, la fonction d'utilité permettra d'exprimer des préférences sur les états. Pour cela, on posera simplement :

$$\forall s, U(s) = g(s) - c(s)$$

Cette fonction se présente donc comme une agrégation des fonctions de gain et de coût. Nous avions vu précédemment que la rationalité d'un agent était définie par sa capacité à prendre des décisions optimales, selon un critère de performance donné. Comment, alors, mettre au point un agent rationnel selon ce concept d'utilité ?

Lorsque l'agent exécute une action, il transite vers un nouvel état $s$, d'utilité $U(s)$. Il exécutera ensuite une nouvelle action, transitant vers un état $s'$ d'utilité $U(s')$, etc. Une approche naïve, dans ce cas, serait d'exécuter à chaque fois l'action menant vers l'état d'utilité maximale (nous verrons par la suite, plus formellement, comment intégrer dans ce raisonnement l'aspect stochastique de l'environnement). On parle ici de raisonnement « à horizon 1 », puisque l'on se contente de maximiser l'utilité à court terme. Cette stratégie n'est toutefois pas forcément judicieuse : imaginons que l'on puisse transiter vers deux états, $s_1$ ou $s_2$. Posons $U(s_1) = 10$ et $U(s_2) = 3$. Dans ce cas, transiter vers l'état $s_1$ semble plus intéressant. Imaginons maintenant que, depuis l'état $s_1$, on ne puisse atteindre qu'un état $s'_1$ d'utilité $U(s'_1) = -1000$, tandis que l'état $s_2$ mène à un état $s'_2$ d'utilité $U(s'_2) = 50$. Il serait alors plus intéressant de passer par $s_2$, pour atteindre ensuite $s'_2$ : on raisonne ici à horizon 2. On peut, de même, raisonner à horizon 3, 4, etc.

On décide donc de l'horizon sur lequel on va raisonner, puis on choisit un comportement (c'est-à-dire, pour chaque état, une action à appliquer). On introduit alors la notion de **valeur d'un état** : pour un comportement donné, la valeur $V$ d'un état $s$ correspond au cumul, sur l'horizon choisi, des utilités de $s$ et des états suivants. Ainsi, si :

1. on connaît l'horizon (c-à-d le nombre de pas de temps) sur lequel s'exécute le problème,
2. on parvient à calculer, pour tout comportement possible, la valeur de chaque état,

alors, on peut déterminer le comportement idéal, maximisant cette fonction de valeur. Un agent sera donc rationnel si il peut calculer ce comportement idéal. On en déduit une définition formelle du concept de planification :

**Définition 11 (Planification)** *Le problème de planification consiste à calculer un **plan**, c'est-à-dire une fonction qui associe à tout état une action à exécuter. Ce problème de planification se résout via un **processus de décision**. Le calcul d'une fonction de valeur est un exemple de processus de décision.*

Ainsi, on pourra calculer un plan tel que l'agent choisisse, en tout état, l'action qui maximise l'utilité à long terme. Ce plan aura la forme d'une fonction qui, à tout état $s$, associe l'action $a$ à exécuter.

## 1.3 Maximisation de la valeur espérée

Nous avons présenté, dans la partie précédente, les concepts d'utilité et de planification. Voyons maintenant comment calculer une fonction de valeurs en environnement stochastique. On parlera ici de maximisation de la **valeur espérée**.

### 1.3.1 Agir sous incertitude

Un agent évoluant en environnement stochastique sera soumis à une difficulté : l'impact d'une action est incertain. Littéralement, cela signifie qu'appliquer une action $a$, dans un état $s$, ne se traduira pas systématiquement par une transition vers un nouvel état $s'$. On aura plutôt un ensemble de transitions possibles vers les états de $S = \{s'_1, s'_2, \ldots\}$ et on notera $P(s'_i|s,a)$ la probabilité d'arriver dans l'état $s'_i$ après avoir appliqué $a$ dans l'état $s$. Prenons l'exemple d'un robot : celui-ci pourra appliquer l'action `avancer` et, dans la majorité des cas, cette action lui permettra de se déplacer vers l'avant ($P = 0.9$). Le risque existe pourtant que ses roues dérapent, et qu'il ne bouge pas de sa position actuelle ($P = 0.1$). Une propriété importante des environnements stochastiques est que l'on a, pour tout état $s$ et action $a$ :

$$\sum_{s'_i \in S} P(s'_i|s,a) = 1$$

Andreï Andreïevitch Markov (1856-1922) a introduit la **propriété de Markov**. Un système respectant cette propriété sera tel que la probabilité de transiter vers un nouvel état ne dépend pas des états précédents, mais uniquement de l'état actuel. Littéralement, cela signifie qu'à tout instant $t$, la probabilité de passer de l'état $s_t$ à l'état $s_{t+1}$ ne dépendra pas des états $s_i, 0 \leq i < t$. On peut donc formaliser cette propriété de la façon suivante :

$$P(s_{t+1} = s'|s_0,s_1,\ldots,s_t) = P(s_{t+1}|s_t)$$

Cette propriété a donné naissance à la notion de **chaîne de Markov**, qui permet de représenter un problème respectant la propriété de Markov.

**Définition 12 (Chaîne de Markov)** *Une chaîne de Markov est un ensemble d'états $S$, muni d'une fonction de transition $T$ telle que $\forall(s,s') \in S^2, T(s,s') = P(s'|s)$ donne la probabilité de passer d'un état $s$ à un état $s'$ (indépendamment des états antérieurs à $s$).*

Ainsi, une chaîne de Markov permet de représenter un problème dans lequel les probabilités de transition ne dépendent que de l'état actuel. On peut représenter une chaîne de Markov via un graphe, dans lequel les noeuds représenteront les états, et les arêtes représenteront les transitions. On indiquera alors, sur chaque arête, la probabilité de la transition correspondante. La figure 1.2 est un exemple de chaîne de Markov pour un problème à quatre états.

Cette représentation est cependant insuffisante pour un problème de planification sous incertitude. En effet, résoudre un problème de planification implique de raisonner sur les actions,

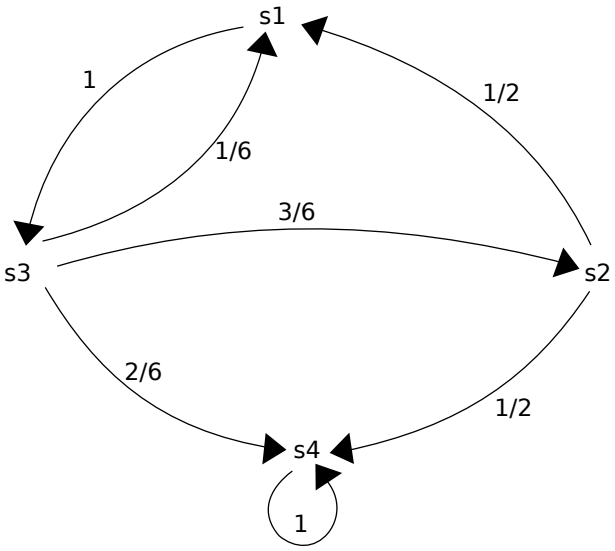


FIGURE 1.2 – Exemple de chaîne de Markov

alors qu'une chaîne de Markov décrit des probabilités de transition indépendantes de toute action. On peut cependant étendre cette représentation afin de réintroduire la notion d'action. La figure 1.3 représente un problème (incomplet) de planification sous incertitude : les losanges sont des états, les ronds sont des actions et les arêtes représentent les transitions.

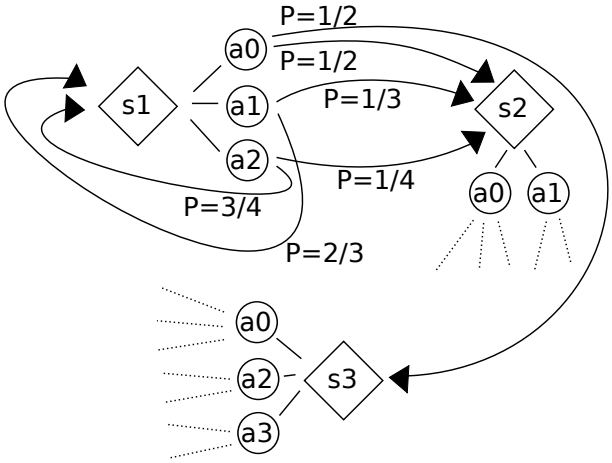


FIGURE 1.3 – Exemple de chaîne de Markov avec actions

On peut voir sur cette figure que dans l'état $s_1$ par exemple, l'agent a le choix entre trois actions ($a_0$, $a_1$ ou $a_2$). Selon l'action choisie, les probabilités de transition vont changer.

### 1.3.2 Valeur espérée d'un comportement

Nous avons, jusqu'à présent, parlé de « comportement ». On introduit maintenant la notion de **politique**, permettant de formaliser un comportement donné.

**Définition 13 (Politique de comportement)** *Une politique de comportement $\pi$ est une fonction qui, à tout état $s$, associe une action $a$. Ainsi, suivre une politique $\pi$ lorsque l'on se trouve dans un état $s$ signifie exécuter l'action $\pi(s)$.*

On a introduit, précédemment, la notion de valeur d'une politique. Calculer cette fonction de valeur en environnement stochastique n'est pas trivial : puisque l'on ne connaît pas, avec certitude, le résultat des actions exécutées, comment prendre en compte l'utilité des états futurs ? On estime la valeur d'une politique $\pi$ donné. Notons $V_i^\pi(s)$ la valeur de l'état $s$, selon la politique $\pi$, calculée à horizon $i$. À horizon 1, on pose tout simplement :

$$V_1^\pi(s) = U(s)$$

À horizon 2, on prend en compte l'utilité de $s$, plus l'utilité de l'état $s'$ dans lequel l'agent arrivera après exécution de l'action $\pi(s)$ (cette action étant donc « choisie » par la politique dont on calcule la valeur). L'environnement étant stochastique, l'exécution de $\pi(s)$ dans l'état $s$ peut mener à plusieurs états $s'$ possibles : on suppose que l'on connaît les probabilités $P(s'|s,\pi(s))$. On peut alors écrire :

$$V_2^\pi(s) = U(s) + \sum_{s'} P(s'|s,\pi(s)).U(s')$$

Poursuivons ce raisonnement : à horizon 3, il faudra prendre en compte l'utilité de $s$, des états $s'$, et des états $s''$ (états dans lesquels on arrive après deux pas de temps, en appliquant la politique $\pi(s')$). On écrira donc :

$$V_3^\pi(s) = U(s) + \sum_{s'} P(s'|s,\pi(s)).\left(U(s') + \sum_{s''} P(s''|s',\pi(s')).U(s'')\right)$$

On pourrait poursuivre ce raisonnement à horizon 4, 5, etc. On constate toutefois une chose : dans $V_3^\pi(s)$, le bloc $\left(U(s') + \sum_{s''} P(s''|s',\pi(s')).U(s'')\right)$ est en fait l'équation de $V_2^\pi(s')$. On peut donc réécrire :

$$V_3^\pi(s) = U(s) + \sum_{s'} P(s'|s,\pi(s)).V_2^\pi(s')$$

De la même manière, on peut écrire $V_2^\pi(s) = U(s) + \sum_{s'} P(s'|s,\pi(s)).V_1^\pi(s')$. Cette notation se généralise, et on note la valeur de l'état $s$, à horizon $i$ (toujours selon $\pi$) :

$$V_i^\pi(s) = U(s) + \sum_{s'} P(s'|s,\pi(s)).V_{i-1}^\pi(s')$$

On dispose donc d'une équation, permettant de déterminer pour tout état la valeur d'une politique donnée, à horizon $i$.

### 1.3.3 Notion de politique optimale

Le but d'un agent est de déterminer quelle politique appliquer. Pour cela, on va utiliser la notion de valeur espérée, introduite précédemment. On introduit ici une notion intermédiaire, dite de $Q$-valeur.

**Définition 14 ($Q$-valeur)** *La $Q$-valeur $Q^{\pi}(s,a)$ désigne la valeur de l'état $s$, lorsque l'on y exécute l'action $a$ puis que l'on applique la politique $\pi$ dans les états suivants. On parle notamment de $Q$-valeur maximale, lorsque la politique adoptée dans les états suivants est supposée optimale (la meilleure politique possible).*

L'équation de calcul d'une $Q$-valeur peut être facilement déduite des équations précédentes. Si l'on suppose, pour les états suivants, l'exécution d'une politique $\pi$ dont la valeur est estimée à horizon $i$, on note :

$$Q^{\pi}(s,a) = U(s) + \sum_{s'} P(s'|s,a).V_i^{\pi}(s')$$

Cette équation permet de déduire la « qualité » d'une action, pour un état en particulier. Le comportement optimal consistera donc simplement à toujours choisir l'action de $Q$-valeur maximale. On définit ainsi la notion de **politique optimale**.

**Définition 15 (Politique optimale)** *La politique optimale $\pi^*$ est celle qui maximise la valeur espérée, en tout état. Ainsi, pour tout état $s$, on a $Q^{\pi^*}(s,\pi^*(s)) = \max_a Q^{\pi^*}(s,a)$. On peut donc définir la politique optimale avec, pour tout état $s$, $\pi^*(s) = \operatorname{argmax}_a Q^{\pi^*}(s,a)$.*

> **Rappel :** l'opérateur $\operatorname{argmax}_{x\in X} f(x)$ renvoie un élément $x$ qui maximise $f$ (contrairement à l'opérateur classique max, qui renvoie le résultat maximal de $f$). Par exemple, avec l'ensemble $X = \{1,2,3\}$ et $f(x) = 5 - x$, on a $\max_{x\in X} f(x) = f(1) = 4$, tandis que $\operatorname{argmax}_{x\in X} f(x) = 1$. Une autre façon de définir cet opérateur serait d'écrire :
>
> $$\operatorname*{argmax}_{x\in X} f(x) = x_i \ tq. \ \left(x_i \in X \ et \ f(x_i) = \max_{x\in X} f(x)\right)$$
>
> Il convient, pour être rigoureux, de noter que l'opérateur argmax renvoie un ensemble d'éléments (car il peut y avoir plusieurs éléments $x_i, x_j, \ldots$ de valeurs égales, c'est à dire tels que $f(x_i) = f(x_j) = \max_{x\in X} f(x)$). Afin de simplifier l'écriture, on supposera alors que argmax renvoie toujours le premier de ces éléments.

On peut donc, en s'appuyant sur les $Q$-valeurs, construire la politique optimale. Une application de ce concept (dit de **maximisation des valeurs espérées**) est proposée dans la figure 1.4. Celle-ci décrit un exemple simple, à 5 états et 2 actions, dans lequel les états $s_4$ et $s_5$ sont finaux. On cherche à calculer la valeur de l'état $s_1$, à horizon 3.

La valeur des états $s_4$ et $s_5$ à horizon 1 est triviale (il s'agit de leur utilité). On calcule donc la valeur de $s_2$ et $s_3$ à horizon 2 : on en déduit que les 2 actions possibles sont de même

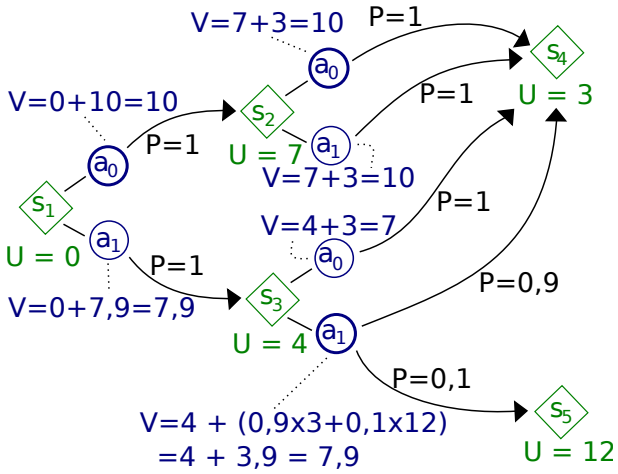


FIGURE 1.4 – Exemple de valeurs espérées

intérêt en $s_2$ (on choisit donc, arbitrairement, $a_0$) mais que l'action $a_1$ est plus intéressante en $s_3$ ($Q(s_3,a_1) = 7{,}9$ et $Q(s_3,a_0) = 7$). On calcule finalement la valeur de $s_1$, pour lequel l'action $a_0$ s'avère la plus intéressante. On dispose alors d'une politique optimale, au vu des probabilités de transition : en $s_1$ par exemple, l'agent exécutera l'action $a_0$ dont la $Q$-valeur ($Q(s_1,a_0) = 10$) est supérieure à celle de l'action $a_1$ ($Q(s_1,a_1) = 7{,}9$).

## Conclusion

Nous avons exposé le problème de la planification, dans lequel un agent (situé dans un environnement) doit calculer une politique de comportement optimale afin d'optimiser son espérance de gain. Dans le cadre de nos travaux, nous nous limitons aux problèmes en environnement stochastique respectant la propriété de Markov. Le prochain chapitre présente le modèle des processus de décision markoviens (MDP), qui est parfaitement adapté à la représentation et à la résolution de tels problèmes.

# Chapitre 2

# Planification sous incertitude

**Sommaire**



Nous avons introduit, dans le chapitre précédent, les concepts de planification en environnement stochastique. Le modèle des processus de décision markoviens, ou MDP [5], est un modèle parfaitement adapté à la représentation et à la résolution de tels problèmes, dès lors que la propriété de Markov est respectée. Dans ce chapitre, nous présentons le modèle MDP ainsi que ses extensions aux problèmes partiellement observables et multiagents.

## 2.1 Processus de décision markovien

Nous présentons dans cette section le modèle MDP, adapté à la représentation des problèmes de décision en environnement stochastique respectant la propriété de Markov (les transitions ne dépendent que de l'état actuel et non des états précédents). Ce modèle propose une description simple de l'ensemble des états, actions, transitions et récompenses associées et permet de calculer rapidement une politique optimale.

5. MDP : de l'anglais « Markovian Decision Process ». Une erreur assez répandue consiste à écrire « processus décisionnel de Markov », au lieu de « processus de décision markovien », en traduction de « Markov Decision Process ». Cette seconde écriture est erronée, les MDP n'ayant pas été introduits par Markov mais étant simplement qualifiés de « markoviens » car vérifiant l'hypothèse du même nom. Les premiers articles traitant de MDP [Bellman, 1957] utilisaient d'ailleurs bien l'écriture « Markovian Decision Process ».

### 2.1.1 Le modèle MDP

On commence par introduire le modèle MDP et donner une description rapide des éléments qui le composent. Une description détaillée sera donnée par la suite.

**Définition 16 (Processus de décision markovien (MDP))** *Un MDP est défini par un quadruplet $\langle S,A,T,R\rangle$ tel que :*

- $S = \{s_1,s_2,\ldots,s_{|S|}\}$ *est un ensemble fini d'états dans lesquels l'agent peut se trouver,*
- $A = \{a_1,a_2,\ldots,a_{|A|}\}$ *est un ensemble fini d'actions exécutables par l'agent,*
- $T : S\times A\times S \to [0,1]$ *est la* ***fonction de transition****, qui décrit la dynamique du système : $T(s,a,s')$ donne la probabilité de passer de l'état $s$ à l'état $s'$, après avoir appliqué $a$,*
- $R : S\times A\times S \to \mathbb{R}$ *est la* ***fonction de récompense*** *: cette fonction associe une récompense (qui peut être négative) à toute transition $(s,a,s')$.*

*Résoudre un MDP consiste alors à calculer une politique $\pi : S \to A$.*

#### Les états et actions

On définit un ensemble fini $S$ décrivant l'ensemble des états dans lesquels l'agent peut se trouver (voir le chapitre 1). On définit également un ensemble $A$ d'actions exécutables par l'agent. Par souci de simplicité, on supposera que les actions sont applicables en tout état. Le modèle MDP est un modèle discret. Ainsi, on ne travaille pas sur des durées, mais sur des pas de temps (un pas correspond à l'exécution d'une action, puis à l'observation de l'état résultant). Chaque action s'exécute donc en exactement 1 pas de temps (l'exécution ne peut pas s'étendre sur plusieurs pas de temps). La figure 2.1 présente le schéma standard d'exécution d'un MDP.

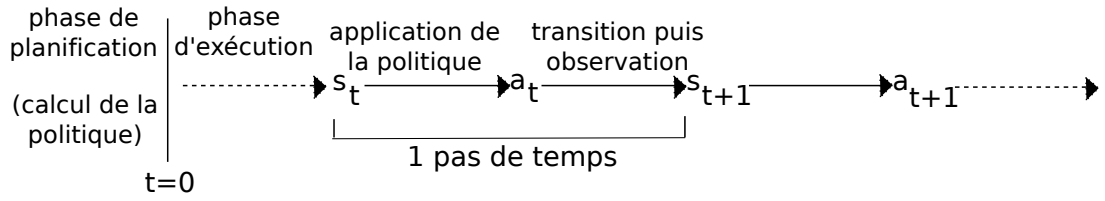


FIGURE 2.1 – Schéma d'exécution d'un MDP

#### La fonction de transition

La fonction de transition décrit la dynamique du système. Elle donne donc, pour tout état $s$ et action $a$, la probabilité de transiter vers un nouvel état $s'$. On considère dans le modèle standard qu'il n'existe pas d'état « final », ce qui signifie que $T$ peut s'appliquer en tout état $s$ :

$$\forall s \in S, \forall a \in A, \left( \exists s' \in S \ tq.\ T(s,a,s') > 0 \ et \ \sum_{s'\in S} T(s,a,s') = 1 \right)$$

Il est important de garder à l'esprit que cette fonction est une **représentation** du problème, ce qui veut dire que les probabilités données ne sont pas nécessairement exactes. On peut par

exemple imaginer une transition dont la probabilité réelle est de 0,897 mais que l'on aurait approchée à 0,9. La perte de qualité de la politique sera alors proportionnelle aux erreurs commises dans la description du problème.

#### La fonction de récompense

La fonction de récompense décrit l'intérêt pour l'agent d'effectuer une transition donnée. Elle associe donc, à toute transition $(s,a,s')$, une valeur réelle. Cette valeur pourra être positive ou négative, selon que la transition soit considérée comme bénéfique pour l'agent, ou à éviter. Les récompenses pourront correspondre à des éléments concrets (par exemple, un agent taxi aura une récompense calculée en fonction de la durée de ses trajets), ou à des éléments abstraits (un agent recevant par exemple une récompense de -1000 si il tombe dans un trou).

La forme standard pour une récompense est une fonction $R(s,a,s')$. On peut cependant envisager d'utiliser une fonction $R(s,a)$, et passer d'une forme à l'autre via la formule $R(s,a) = \sum_{s' \in S} [T(s,a,s').R(s,a,s')]$. D'autres formes moins classiques peuvent être utilisées ($R(s,s')$, $R(a)$ ou $R(s')$ par exemple), en fonction du problème traité. Cette fonction est donc une généralisation de la notion d'utilité, définie dans le chapitre précédent (la restriction de $R(s,a,s')$ à $R(s')$ donne une fonction équivalente à l'utilité $U(s')$ d'un état).

### 2.1.2 Résolution d'un MDP

L'objectif réel d'un MDP n'est pas la représentation en elle-même, mais plutôt son traitement afin de calculer une politique de comportement $\pi : S \rightarrow A$ optimale. Avant de calculer cette politique, il faut décider d'un critère d'évaluation, que l'on cherchera à optimiser.

#### Critère d'évaluation d'une politique

Nous avons vu, précédemment, la notion de **fonction de valeur**. Une fonction de valeur $V : S \rightarrow \mathbb{R}$ associe, pour rappel, une valeur numérique à tout état. Il existe une équation, centrale dans le domaine des MDPs, permettant de calculer, pour une politique $\pi$ donnée, la fonction de valeur $V^\pi$ associée : il s'agit de l'**équation de Bellman** [Bellman, 1957].

**Définition 17 (Equation de Bellman)** *L'équation de Bellman permet de calculer la fonction de valeur $V$ associée à une politique $\pi$ donnée, avec $0 \leq \gamma < 1$ :*

$$\forall s \in S, V^\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in S} T(s,\pi(s),s').V^\pi(s')$$

Cette fonction est donc telle que la valeur d'un état corresponde à sa récompense immédiate, plus l'espérance de gain sur les états suivants. On calcule ici la valeur à **horizon infini** (on a introduit, dans le chapitre précédent, la notion d'horizon pour décrire le nombre de pas de temps sur lequel on raisonne. Ici, on adopte un raisonnement optimal, dans le sens où on raisonne au « plus long terme » possible). On utilise alors le facteur d'atténuation $\gamma$, qui permet de

donner plus d'importance aux récompenses à court terme (sans ce facteur, la valeur serait potentiellement infinie pour tout état). Plus $\gamma$ se rapproche de 1, plus on raisonne à long terme mais plus $V$ est long à calculer. Au contraire, un $\gamma$ proche de 0 implique un raisonnement à court terme mais une fonction $V$ rapide à calculer. En général, on choisit un $\gamma$ très proche de 1.

Ainsi, en estimant la valeur d'une politique via l'équation de Bellman, on peut choisir la politique optimale $\pi^*$ qui a pour valeur $V^*$. L'équation donnée ici correspond au cas à « horizon infini ». L'horizon est une valeur représentant le nombre de pas de temps considérés pour le calcul de $V$ : un horizon de 15 par exemple signifie qu'on considère la récompense immédiate, ainsi que les récompenses espérées sur les 14 pas de temps suivants. Cela ne veut pas nécessairement dire que le problème va s'arrêter après ces 15 pas de temps, mais seulement qu'on raisonne 15 « coups » à l'avance. Dans le cas d'un horizon fini, on peut poser $\gamma = 1$.

Un horizon infini implique que l'on raisonne sur une infinité de pas de temps : en réalité, on ne raisonne pas vraiment sur une infinité, mais sur un nombre de pas suffisamment élevés pour que tout pas supplémentaire ait un impact négligeable sur la fonction de valeur. En effet, l'équation de Bellman est telle que l'on considère 1 fois la récompense immédiate, $\gamma$ fois la récompense après un pas de temps, ..., $\gamma^i$ fois la récompense après $i$ pas de temps. Étant donné la valeur de $\gamma$, comprise entre 0 et 1, $\gamma^i$ tend vers 0 quand $i$ tend vers l'infini.

### Algorithmes de résolution

Il existe plusieurs algorithmes permettant de calculer une fonction de valeur optimale et la politique associée. L'algorithme **Value-Iteration** [Bellman, 1957] est un des plus couramment utilisés (algorithme 1).

**Algorithme 1** : Algorithme de Value-Iteration.

**Entrées** : le MDP $\langle S,A,T,R\rangle$, une borne $\epsilon$ et le coefficient d'atténuation $\gamma$
**Sorties** : politique $\pi$
**pour chaque** $s \in S$ **faire** $V(s) \leftarrow \max_{a\in A} R(s,a)$;
**répéter**
  $\Delta \leftarrow 0$;
  **pour chaque** $s \in S$ **faire**
    **pour chaque** $a \in A$ **faire** $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s'\in S} T(s,a,s').V(s')$;
    $V'(s) \leftarrow \max_{a\in A} Q(s,a)$;
    $\pi(s) \leftarrow \text{argmax}_{a\in A} Q(s,a)$;
    $\Delta \leftarrow max(|V(s) - V'(s)|,\Delta)$;
  $V \leftarrow V'$;
**jusqu'à** $\Delta < \frac{\epsilon(1-\gamma)}{2\gamma}$ ;
**retourner** $\pi$;

Cet algorithme repose sur une équation dérivée de l'équation de Bellman :

$$\forall s \in S, V^*_{t+1}(s) = \max_{a\in A} \left( R(s,a) + \gamma \sum_{s'\in S} T(s,a,s').V^*_t(s') \right)$$

L'idée est donc de construire récursivement la fonction $V^*$. On initialise $V_0$ avec des valeurs au choix (une initialisation proche des vrais valeurs permet un processus de calcul plus rapide : initialiser à $\forall s \in S, V_0(s) = \max_{a \in A} R(s,a)$ est en général une bonne solution). On construit ensuite $V_1$ à partir de $V_0$, $V_2$ à partir de $V_1$, etc. On s'arrête lorsque la fonction de valeur n'évolue plus entre deux itérations. Il a été démontré que cette méthode permet de converger vers l'unique fonction de valeur optimale [Bertsekas, 1995]. On peut alors extraire une politique optimale $\pi^*$ à partir de $V^*$ (plusieurs politiques optimales de même qualité sont possibles). En réalité, la fonction de valeur que l'on a calculée est optimale à $\epsilon$ prêt. On peut donc garantir un écart maximum de $\epsilon$ entre les valeurs de $V^*$ et les valeurs optimales. En général, on choisit $\epsilon < 0{,}1$ afin d'obtenir une bonne précision (bien entendu, plus $\epsilon$ est petit et plus $V^*$ sera longue à calculer). Si $\epsilon$ est suffisamment petit, $\pi^*$ est optimale : l'erreur faite sur $V^*$ n'est plus suffisante pour modifier la politique.

On pourrait également citer l'algorithme de **Policy Iteration** [Howard, 1960]. L'idée (algorithme 2) est de calculer une politique initiale, puis sa valeur, et de modifier la politique jusqu'à en trouver une de valeur $V^*$. Cette approche s'oppose à celle utilisée dans Value Iteration, où on construit progressivement $V^*$ afin d'en extraire $\pi^*$.

**Algorithme 2** : Algorithme de Policy-Iteration.

**Entrées** : le MDP $\langle S,A,T,R \rangle$ et le coefficient d'atténuation $\gamma$
**Sorties** : politique $\pi$
**pour chaque** $s \in S$ **faire**
  $\pi(s) \leftarrow \text{argmax}_{a \in A} R(s,a)$;
  $V(s) \leftarrow \max_{a \in A} R(s,a)$;
**répéter**
  **répéter**
    $\Delta \leftarrow 0$;
    **pour chaque** $s \in S$ **faire**
      $V'(s) \leftarrow R(s,\pi(s)) + \gamma \sum_{s' \in S} T(s,\pi(s),s').V(s')$;
      $\Delta \leftarrow max(|V(s) - V'(s)|,\Delta)$;
    $V \leftarrow V'$;
  **jusqu'à** $\Delta < \frac{\epsilon(1-\gamma)}{2\gamma}$ ;
  **pour chaque** $s \in S$ **faire**
    **si** $\exists a \in A$ *tq.* $R(s,a) + \gamma \sum_{s' \in S} T(s,a,s').V^{\pi}(s') > V^{\pi}(s)$ **alors**
      $\pi'(s) \leftarrow a$;
    **sinon**
      $\pi'(s) \leftarrow \pi(s)$;
  $\pi^{old} \leftarrow \pi$;
  $\pi \leftarrow \pi'$;
**jusqu'à** $\forall s \in S, \pi(s) = \pi^{old}(s)$ *puis* **retourner** $\pi$ ;

L'algorithme de Policy-Iteration permet de résoudre un MDP en moins d'itérations qu'un Value-Iteration, mais chaque itération est plus longue à effectuer. Des études ont été menées sur la comparaison de ces deux approches [Littman, 1996], mais la question de savoir quel algorithme

est le meilleur, entre PI ou VI, demeure sans réponse. Selon la structure du problème étudié, l'un ou l'autre de ces deux algorithmes sera le plus rapide. Dans la majorité des cas, c'est l'algorithme de Value-Iteration qui est utilisé, mais ce choix est probablement du à sa facilité d'implémentation plus qu'à une raison scientifique.

D'autres algorithmes ont été proposés, exploitant la structure particulière de certains types de problèmes afin d'en accélérer le processus de résolution [Boutilier *et al.*, 1995]. On peut par exemple citer le cas des MDP orientés but, dans lesquels on connaît les états de départ et d'arrivée de l'agent, ce qui permet d'orienter la résolution et de gagner considérablement en temps. On peut également citer les MDPs factorisés (FMDP [Boutilier *et al.*, 1999]), dont la représentation compacte permet de traiter directement des « paquets » d'états (plutôt que de traiter les états un à un). L'approche MDP a de plus été étendue à des cas plus généraux, dans lesquels on considère que l'agent peut n'avoir qu'une observabilité partielle de son environnement par exemple, ou impliquant non plus un, mais plusieurs agents. Les deux prochaines sections présentent ces deux extensions du modèle MDP.

## 2.2 Le cas partiellement observable

Nous avons vu, dans le chapitre 1, qu'un agent pouvait n'avoir qu'une observabilité partielle de son environnement. Pourtant, le modèle MDP implique que l'agent sache à tout instant dans quel état il se trouve, afin d'appliquer sa politique. Nous allons maintenant voir comment ce modèle peut être généralisé aux problèmes partiellement observables.

### 2.2.1 Processus de décision markovien partiellement observable

Afin d'étendre le modèle MDP aux problèmes partiellement observables, il est nécessaire de donner un cadre formel à la notion d'observabilité. Pour cela, on considère qu'après chaque action, l'agent ne peut pas observer l'état dans lequel il arrive mais qu'il reçoit une **observation** au sujet de cet état. Ainsi, un agent qui agit en environnement partiellement observable alternera des phases d'observation et des phases de décision (voir figure 2.2).

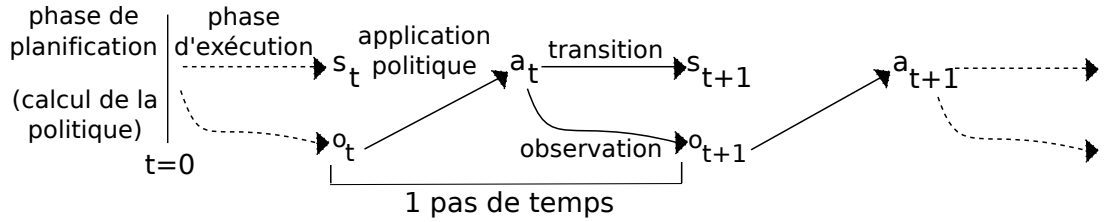


FIGURE 2.2 – Schéma d'exécution d'un POMDP

Le modèle des processus de décision markoviens partiellement observables (ou POMDP [6] [Cassandra *et al.*, 1994]) généralise les MDPs aux environnement partiellement observables.

6. POMDP : de l'anglais « Partially Observable Markovian Decision Process ».

**Définition 18 (Processus de décision markovien partiellement observable)** *Un processus de décision markovien partiellement observable (POMDP) est défini, similairement à un MDP, par un n-uplet $\langle S,A,T,R,\Omega,O\rangle$ tel que :*

- *$S$ est un ensemble fini d'états,*
- *$A$ est un ensemble fini d'actions,*
- *$T : S \times A \times S \to [0,1]$ est une fonction de transition,*
- *$R : S \times A \times S \to \mathbb{R}$ est une fonction de récompense,*
- *$\Omega = \{o_1, \ldots, o_{|\Omega|}\}$ est un ensemble fini d'observations,*
- *$O : S \times A \times S \times \Omega \to [0,1]$ est une fonction d'observation.*

Les éléments $S$, $A$, $T$ et $R$ sont identiques à ceux d'un MDP. Viennent s'ajouter l'ensemble $\Omega$ et la fonction d'observation $O$. La notation $\Omega$ désigne l'ensemble des observations qu'un agent peut recevoir durant l'exécution du problème : on suppose cet ensemble fini. On peut, par exemple, imaginer un agent qui évolue dans un bâtiment : il ne connaîtra pas sa position exacte (à la façon d'un GPS), mais recevra après chaque déplacement une observation (y'a t'il un mur devant, derrière, sur les côtés ? Y'a t'il une porte, un couloir, une salle). La **fonction d'observation** $O$ quant à elle, donne la probabilité $O(s,a,s',o) = P(o|s,a,s')$ de recevoir l'observation $o$ après avoir exécuté la transition $(s,a,s')$.

Ainsi, lors de l'exécution d'un POMDP, l'agent ne connaît pas l'état dans lequel il se trouve mais uniquement l'ensemble des actions précédemment exécutées et des observations reçues. Ces connaissances dont dispose l'agent sont appelées **historique**.

**Définition 19 (Historique)** *À l'instant $t$, on note $h_t$ l'historique d'un agent. Cet historique est un n-uplet $h_t = (o_0, a_0, o_1, a_1, \ldots, a_{t-1}, o_t)$ tel que $o_i$ désigne l'observation reçue à l'instant $i$ et $a_i$ désigne l'action exécutée après cette observation.*

L'historique est une connaissance suffisante pour calculer et exécuter une politique optimale. Maintenir un historique peut cependant s'avérer problématique. En effet, certains problèmes s'exécutent sur un grand nombre de pas de temps : cela implique de conserver des historiques de grand taille ($h_t$ avec $t$ très grand). Calculer une politique pour un POMDP implique de calculer l'action optimale pour chaque historique possible. Hors le nombre d'historiques possibles à l'instant $t$ est, dans le pire cas, en $O(|\Omega|!^t)$. La taille de la politique augmente donc exponentiellement en la durée du problème.

### 2.2.2 Résolution d'un POMDP

Il apparaît, au vu de la section précédente, que résoudre un POMDP en utilisant la notion d'historique n'est possible que pour des problèmes s'exécutant sur un petit nombre de pas de temps. La notion d'**état de croyance** a été introduite [Astrom, 1965] afin de palier à cette difficulté.

**Définition 20 (État de croyance)** *Un état de croyance*[7] *$b$ est une distribution de probabilités sur les états du problème. À l'instant $t$, $b_t(s_i)$ donne la probabilité que l'agent soit dans l'état $s_i$.*

Il a été démontré que, pour tout historique, on peut calculer un état de croyance équivalent. Ainsi, on ne raisonnera plus sur des n-uplets de taille potentiellement infinie, mais sur des distributions de probabilités de taille fixe (en tout instant $t$, $|b_t| = |S|$). De plus, une politique utilisant des états de croyance est plus simple à représenter :

- on associe, à toute action $a$ un ensemble de $k$ « alpha-vecteurs » $\{\alpha_{(a,1)},\alpha_{(a,2)},\ldots,\alpha_{(a,k)}\}$,
- un alpha-vecteur $\alpha$ est un $|S|$-uplet $(v_1,\ldots,v_{|S|})$ où $v_i$ est une valeur attribuée à l'état $s_i$,
- on définit le produit scalaire entre un état de croyance $b = (b(s_1),\ldots,b(s_{|S|}))$ et un alpha-vecteur $\alpha = (\alpha(s_1),\ldots,\alpha(s_{|S|}))$ par l'opérateur $\alpha^T b = \sum_{s\in S} b(s).\alpha(s)$,
- on pose alors $\pi(b) = \underset{a\in A}{\text{argmax}} \max_{\alpha_{(a,i)}} \alpha_{(a,i)}^T b$.

La figure 2.3 est un exemple d'utilisation des alpha vecteurs, dans un problème à deux états $s_1$ et $s_2$. Ici, on dispose de deux actions $a_1$ (associée à deux alpha-vecteurs) et $a_2$ (associée à un seul alpha-vecteur). Ainsi, pour l'état de croyance $b$, on appliquera l'action $a_1$.

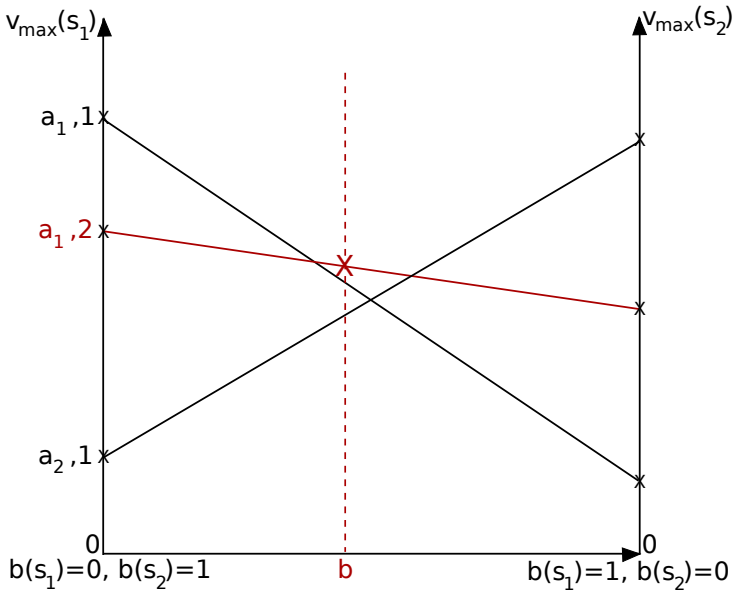


FIGURE 2.3 – Exemple d'utilisation des alpha-vecteurs

Ainsi, on peut avoir une infinité d'états de croyance possibles, mais on se contente pour représenter la politique de mémoriser un ensemble fini d'alpha-vecteurs. Le calcul de l'action à exécuter sera alors effectué à la volée, au fur et à mesure de l'exécution (selon ces alpha-vecteurs). Nous ne détaillerons pas dans cet état de l'art les différents algorithmes permettant de calculer les alpha-vecteurs, cet aspect étant relativement éloigné des travaux que nous traitons ici. Nous pouvons donc, à tout moment, choisir l'action optimale au vu de l'état de croyance $b$. Cet état de croyance est assez simple à calculer. On suppose que l'on dispose d'un état de croyance $b_0$

7. L'état de croyance est traditionnellement noté $b$ en référence au nom anglais « Belief state ».

décrivant les connaissances initiales de l'agent. Après chaque action $a$ suivie d'une observation $o$, on peut **mettre à jour** l'état de croyance $b_t$.

$$\forall s' \in S, b_{t+1}^{a,o}(s') = \frac{\sum_s b_t(s) \cdot T(s,a,s') \cdot O(s,a,s',o)}{\sum_{s''} \sum_s b_t(s) \cdot T(s,a,s'') \cdot O(s,a,s'',o)}$$

Le passage à un modèle partiellement observable n'est pas sans conséquence : à horizon fini, la complexité de résolution passe de P-complet, pour un MDP, à PSPACE-complet [8] pour un POMDP [Papadimitriou et Tsitsiklis, 1987]. À horizon infini, la résolution exacte est un problème indécidable [Madani *et al.*, 1999], mais une solution approchée peut être trouvée, via une approche heuristique [Cassandra, 1998, Aberdeen, 2003]. Ce modèle offre ainsi une première possibilité d'extension des MDPs, mais d'autres sont possibles, notamment au cas multiagent.

## 2.3 Problème multiagent

Nous n'avons, jusqu'à présent, traité que le cas monoagent. Pourtant, on peut imaginer un problème impliquant un ensemble d'agents. Prenons l'exemple d'un groupe de robots ayant pour tâche le rangement d'un entrepôt : on peut imaginer que certaines caisses seront trop lourdes pour être transportées par un agent seul. Les agents devront donc prendre en compte les autres agents impliqués dans le problème, avant de choisir l'action à exécuter.

### 2.3.1 Contrôle décentralisé : MDP multiagent

Nous étudions désormais le cadre des problèmes multiagents. Nous considérons alors les problèmes monoagents comme des MDP multiagents particuliers (impliquant un seul agent).

#### Le modèle

Le modèle des processus de décision markoviens multiagents (ou MMDP [9]) a été introduit par Boutilier [Boutilier, 1996] afin d'étendre le modèle MDP aux problèmes multiagents.

**Définition 21 (processus de décision markovien multiagent)** *Un MMDP est défini par un quadruplet $\langle I,S,A,T,R\rangle$ tel que :*

- *$I$ désigne le nombre d'agents impliqués dans le problème,*
- *$S$ est un ensemble fini d'états joints,*
- *$A$ est un ensemble fini d'actions jointes,*
- *$T : S \times A \times S \to [0,1]$ est une fonction de transition jointe,*
- *$R : S \times A \times S \to \mathbb{R}$ est une fonction de récompense jointe.*

*Résoudre un MMDP signifie donc calculer, pour chaque agent $i$, une politique $\pi_i$.*

8. PSPACE : nécessite un espace mémoire polynomial en la taille du problème (classe englobante de NP).
9. MMDP : de l'anglais « Multiagent Markovian Decision Process »

On suppose, dans un MMDP, que les agents sont collaboratifs. Cela signifie que le groupe œuvre à l'accomplissement d'un (ou des) objectif commun (contrairement à des agents égoïstes qui auraient planifié leurs actions afin d'atteindre des objectifs personnels). Les agents sont de plus supposés « synchronisés ». Le schéma est le suivant : le groupe est dans un état donné, les $I$ agents exécutent en même temps une action (l'ensemble des agents doit agir à chaque pas de temps), suite à quoi le groupe transite vers un nouvel état. Ici, $S$ contient un ensemble d'états « joints » : cela signifie que chaque état $s \in S$ peut s'écrire $s = (s_1,s_2,\ldots,s_I)$ avec $s_i$ la portion de $s$ décrivant l'état individuel de l'agent $i$. Imaginons par exemple un problème de navigation :

1. un premier agent se trouve aux coordonnées $(x_1,y_1)$,
2. un second agent se trouve aux coordonnées $(x_2,y_2)$,
3. on a alors $s_1 = (x_1,y_1)$ et $s_2 = (x_2,y_2)$,
4. l'état joint est $s = (s_1,s_2)$.

On défini, de la même façon, l'ensemble d'actions jointes $a \in A$ tel que $a = (a_1,a_2,\ldots,a_I)$. Cette décomposition permet alors de poser $S = S_1 \times S_2 \times \cdots \times S_I$ avec $S_i$ l'ensemble des états dans lesquels l'agent $i$ peut se trouver (de même pour $A$ et un ensemble de $A_i$). En réalité, cette décomposition n'est pas toujours possible : prenons l'exemple d'une opération chirurgicale. Chaque agent travail sur le patient, les actions étant l'ensemble des actes médicaux. Ici, l'état joint décrira la situation du patient : il est alors difficile de le « répartir » entre les agents.

**Définition 22 (Distributivité d'un problème)** *Un problème de décision multiagent est dit* ***distribuable*** *si on peut séparer l'espace d'états $S$ en un produit d'espaces $S_i$, un par agent (on considérera que les actions sont toujours distribuables).*

Les fonctions de transition et de récompense sont également jointes. Ainsi, la fonction $T$ donne la probabilité, lorsque les $I$ agents ont agit, que le groupe passe de l'état joint $s$ à l'état joint $s'$. Si on prend à nouveau l'exemple des robots ayant pour tâche de ranger un entrepôt, l'utilisation d'une fonction de transition jointe permet de représenter les « actions de groupe ». Si au contraire chaque agent avait eu son propre MDP, on n'aurait (par exemple) pas pu représenter le fait que deux agents puissent travailler de concert afin de soulever une caisse lourde. De même, la fonction $R$ associe une récompense aux transitions jointes effectuées par le groupe.

### Résolution d'un MMDP

On peut remarquer que la structure du MMDP est très proche de celle d'un MDP. On peut donc appliquer directement les algorithmes de résolution de MDPs, tels que le Value-Iteration (algorithme 1, section 2.1.2). On obtiendra alors une politique optimale $\pi^*$ qui, à tout état joint $s$, associe une action jointe $a$. Obtenir un comportement optimal n'est possible que par cette approche jointe. Prenons à nouveau l'exemple d'un problème de navigation :

- on suppose l'action « tourner à gauche » optimale pour un agent seul,
- on imagine ensuite un deuxième agent, qui se trouve sur le chemin du premier,

– l'action ≪ tourner à gauche ≫ exécutée par l'agent 1 va créer une collision avec l'agent 2,
– l'action optimale pour l'agent 1 n'est donc pas nécessairement de tourner à gauche, mais peut être de commencer par s'éloigner un peu de l'agent 2 avant de tourner.

La politique optimale doit donc prendre en compte l'ensemble des $I$ agents impliqués dans le problème. Il faut alors faire attention à la complexité de la résolution : le temps nécessaire pour effectuer une passe de l'algorithme de Value-Iteration est en $O(|S|^2.|A|)$. Hors, avec $n$ le nombre d'états dans lesquels un agent donné peut se trouver, on a $|S| = n^I$. Ainsi, l'algorithme VI est polynomial en le nombre d'états, mais ce nombre est exponentiel en le nombre d'agents.

#### Notion de contrôle décentralisé

La notion de centralisation/décentralisation est essentielle à la compréhension des problèmes multiagents. Cette notion intervient aussi bien durant la planification que durant l'exécution. Le modèle MMDP, tel que nous l'avons présenté précédemment, suppose qu'un élément central calcul une politique optimale jointe. On parle ici de planification centralisée. On pourrait également imaginer une planification décentralisée, dans laquelle chaque agent calculerait sa propre politique. La planification décentralisée n'implique pas nécessairement que les agents agissent indépendamment les uns des autres : un agent peut très bien calculer sa propre politique, tout en prenant en compte l'existence des autres agents.

De la même manière, l'exécution de la politique peut être centralisée (un élément central donne des ordres à l'ensemble des agents), ou décentralisée (chaque agent prend ses propres décisions, au vu de la politique jointe). Il est en général peu réaliste d'envisager une exécution centralisée, au vu des coûts en communication. L'exécution décentralisée nécessite cependant que chaque agent $i$ dispose de sa propre politique $\pi_i$. On a vu précédemment comment calculer la politique jointe optimale $\pi^*$ : cette politique associe, à chaque état joint $s$, une action jointe $a$. On peut donc représenter la politique $\pi^*$ comme un n-uplet :

$$\forall s \in S, \pi^*(s) = (\pi_1^*(s),\pi_2^*(s),\ldots,\pi_I^*(s))$$

Ainsi, en distribuant à chaque agent sa politique $\pi_i^*$, on pourra mettre en place une exécution décentralisée. On suppose ici qu'à tout moment, l'agent peut observer l'état joint : nous verrons par la suite que cette hypothèse n'est pas toujours vraie. Dans tous les cas, le rôle de la communication est prépondérant : un problème dans lequel la communication est gratuite et illimitée sera considéré comme centralisé. Cette hypothèse est cependant rarement vérifiée : dans le cas général, on considère que toute communication est indésirable.

### 2.3.2 Complexité des problèmes multiagent partiellement observables

Nous avons vu comment étendre un MDP au cas partiellement observable, ainsi qu'au cas multiagent. Voyons maintenant comment réunir ces deux propriétés.

**Problématique**

Le modèle MMDP permet la représentation de problèmes complètement observables : cette hypothèse d'observabilité complète est obligatoire. Nous avons montré précédemment (via le cas de l'agent devant tourner à gauche) en quoi cette hypothèse est nécessaire pour la prise d'une décision optimale. Il n'est pourtant pas toujours possible d'observer la totalité de l'environnement. Prenons l'exemple de plusieurs agents dans une pièce, l'un d'eux voulant sortir :

- l'agent ne sait pas si la porte est fermée à clef ou non, et ne le saura pas temps qu'il n'aura pas essayé de sortir : il y a observabilité partielle sur l'environnement,
- l'agent ne sait pas si un des autres agents présents dispose de la clef pour ouvrir la porte : il y a observabilité partielle sur les autres agents.

De tels problèmes sont plutôt courants. En réalité, il est rarement possible de mettre au point un agent omniscient. Comment, dans ces conditions, appliquer la politique précédemment calculée ? Il s'agit là de la limite du modèle MMDP.

**Conséquences de l'observabilité partielle en environnement multiagent**

En environnement partiellement observable, la composante multiagent est une source importante de complexité. L'agent devant considérer l'état joint pour prendre une décision, il ne faut plus seulement établir une croyance sur l'état de l'agent, mais aussi sur les états de l'ensemble des autres agents. Un tel état de croyance est alors beaucoup plus coûteux à mémoriser (la taille de $S$ augmente exponentiellement en le nombre d'agents), mais aussi plus complexe à calculer (rien ne dit que les observations reçues permettront d'estimer l'état des autres agents).

Un second problème se pose dans ce type de situation : on a vu que la mise à jour de l'état de croyance repose sur l'observation reçue et sur l'action effectuée (ici, l'action jointe), il est donc nécessaire de connaître cette action jointe. Hors, pour savoir quelles actions les autres agents ont exécuté, il faut connaître leurs états de croyance respectifs, mais l'agent ne dispose pas de cette information... La seule solution est donc de maintenir une distribution de probabilités sur l'ensemble des états de croyance possibles pour les autres agents. Cet ensemble étant infini, la distribution de probabilités sera également de taille infinie ! Il est donc impossible de maintenir ce type de connaissance en mémoire. Ainsi, même si il existe des techniques d'approximation, on considère en général qu'il est impossible de travailler avec des états de croyance lorsque l'on tente de résoudre un problème multiagent et partiellement observable. On en revient donc au problème initial : il faut travailler avec les historiques $h_{t,i} = (o_0,a_0,\ldots,a_{t-1},o_t)$ de chaque agent $i$. Le problème est alors que le nombre d'historiques possibles est potentiellement très gros.

## Conclusion

Nous avons présenté le modèle MDP pour la représentation de problèmes de décision en environnement stochastique respectant la propriété de Markov. Nous avons ensuite vu comment ce modèle pouvait s'étendre aux problèmes multiagents ou partiellement observables. Nous

présentons dans le chapitre suivant le modèle DEC-POMDP, une extension du modèle MDP à de tels problèmes. Nous verrons notamment les différentes approches possibles pour la résolution de ce type de problème, ainsi que les limitations et avantages de chacune.

# Chapitre 3

# Les modèles de DEC-POMDPs

**Sommaire**



Le modèle des processus de décision markoviens partiellement observables et décentralisés (notés DEC-POMDP [10]) permet une représentation simple et expressive des problèmes de décision multiagents en environnement partiellement observable. Dans ce chapitre, nous commencerons par décrire le modèle en lui-même et la façon dont on peut l'utiliser pour calculer une politique jointe, après quoi nous présenterons les différentes classes de DEC-POMDP possibles et les modèles associés. La question des différents algorithmes de résolution ne sera pas traitée dans ce chapitre mais laissée pour le suivant.

## 3.1 Le modèle classique

Le modèle standard de DEC-POMDP permet la représentation d'un problème dans le cas le plus général. Dans cette section, nous présentons ce modèle standard, tandis que les différents modèles dérivés seront introduits dans les sections suivantes.

10. DEC-POMDP : de l'anglais « Decentralized - Partially Observable Markovian Decision Process ».

### 3.1.1 Description formelle

Le modèle DEC-POMDP [Bernstein *et al.*, 2000] permet la représentation de problèmes de décision de type MDP, lorsque ceux-ci sont à la fois multiagent et en environnement partiellement observable. On peut alors considérer les modèles présentés précédemment comme des DEC-POMDPs particuliers. Un MMDP par exemple est un DEC-POMDP tel que l'observabilité sur l'état joint est toujours complète, tandis qu'un POMDP est un DEC-POMDP à un agent.

**Définition 23 (DEC-POMDP)** *Un processus de décision markovien partiellement observable et décentralisé (ou DEC-POMDP) est défini par un n-uplet* $\langle I,S,A,T,R,\Omega,O\rangle$ *tel que :*

- $I$ *désigne le nombre d'agents impliqués dans le problème,*
- $S$ *est un ensemble d'états joints potentiellement décomposable en* $S = S_1 \times \cdots \times S_I$ *avec* $S_i$ *l'ensemble des états individuels de l'agent* $i$*,*
- $A = A_1 \times \cdots \times A_I$ *est un ensemble d'actions jointes, avec* $A_i$ *l'ensemble des actions applicables par l'agent* $i$*,*
- $T : S \times A \times S \to [0,1]$ *est une fonction de transition jointe,*
- $R : S \times A \times S \to \mathbb{R}$ *est une fonction de récompense jointe,*
- $\Omega = \Omega_1 \times \cdots \times \Omega_I$ *est un ensemble d'observations jointes avec* $\Omega_i$ *les observations que l'agent* $i$ *peut effectuer au sujet de l'environnement,*
- $O : S \times A \times S \times \Omega \to [0,1]$ *est une fonction d'observation jointe.*

*Résoudre un DEC-POMDP signifie calculer une politique jointe* $\pi$ *décomposable en* $I$ *politiques* $\pi_1,\ldots,\pi_I$ *telles que* $\pi_i$ *soit la politique individuelle de l'agent* $i$*.*

Ces éléments sont similaires à ceux que l'on a introduit précédemment. Ainsi, la fonction de transition donnera la probabilité pour le groupe d'agents de passer d'un état joint à un autre via une action jointe, et la fonction de récompense donnera l'intérêt associé à chaque transition (via une valeur potentiellement négative). La fonction d'observation quant à elle donnera la probabilité que le groupe reçoive une observation jointe donnée, après avoir effectué une transition.

L'usage d'un DEC-POMDP suppose en général un calcul centralisé de la politique, suivi d'une exécution décentralisée. Afin que cette exécution décentralisée soit possible, il faut pouvoir diviser chaque action jointe en un ensemble d'action individuelles, et de même pour les observations. Ainsi, l'agent recevra une observation au sujet de l'état joint, appliquera sa politique et exécutera l'action individuelle ainsi déterminée. Un tel modèle permet alors la représentation de la plupart des problèmes de décision « réels ».

Dans le cas général, l'ensemble des états $S$ n'est pas nécessairement divisible en un produit $S_1 \times \cdots \times S_I$ (voir la remarque similaire au sujet des MMDPs en section 2.3.1). Lorsque cette division est possible, on peut alors se poser la question de l'observabilité dont dispose chaque agent. Si chacun dispose, individuellement, d'une observabilité complète au sujet de son état personnel (sans pour autant pouvoir observer complètement les états individuels des autres agents), on parle alors de DEC-MDP. Une telle condition n'apporte pas de gain conséquent en termes de complexité (relativement à un DEC-POMDP), les fonctions de transition, récompense

et observation étant toujours définies de manière jointe, mais permet certains raffinements algorithmiques (nous ne détaillerons pas ces points pour l'instant). Si les agents sont de plus capables de communiquer de façon gratuite et illimitée, ils peuvent alors s'échanger leurs observations et disposer ainsi d'une observabilité complète sur l'état joint : le DEC-MDP devient alors un MMDP. Dans le cas général cependant, ces hypothèses sont rarement vérifiées.

### 3.1.2 Résolution d'un DEC-POMDP

Nous avons expliqué, dans le chapitre précédent, qu'il est impossible d'utiliser des états de croyance pour de tels problèmes. On doit donc baser la politique de l'agent sur son historique d'observations/actions.

#### Les différentes représentations possibles pour la politique d'un agent

On choisi en général de représenter la politique de l'agent par un **arbre** (figure 3.1) dans lequel les nœuds représentent les actions que l'agent doit effectuer et où chaque branche représente une observation possible suite à l'application de cette action.

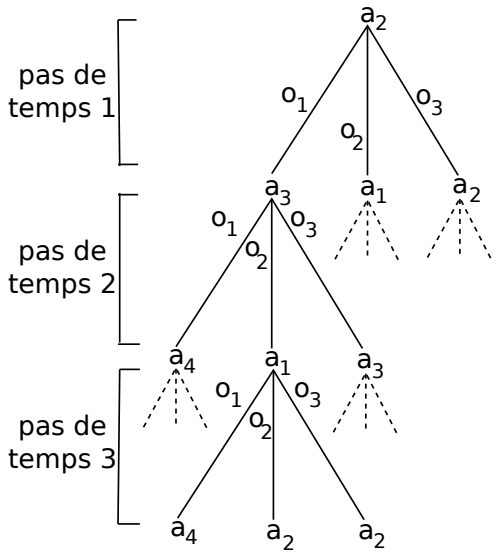


FIGURE 3.1 – Arbre de politique d'un agent

Ce genre d'arbre propose une représentation compacte de l'ensemble des historiques possibles, pour une politique et un horizon donnés. Sur l'exemple donné en figure 3.1, on peut imaginer que l'agent ait réalisé l'historique $h = (a_2,o_1,a_3,o_2)$. Selon son arbre de politique, il devra alors exécuter l'action $a_1$. On peut donc décrire la politique (potentiellement optimale) d'un agent à horizon fini. La politique jointe sera alors un ensemble d'arbres, un par agent. Il y a toutefois un problème, lorsque l'on représente les politiques des agents via des arbres : il est impossible de représenter ainsi une politique à horizon infini (car cela impliquerait d'utiliser des arbres de profondeur infinie, malgré que les ensemble $S$ et $A$ demeurent de taille finie). Il existe un second

type de représentation, permettant de palier ce problème : les **automates déterministes à états finis** (figure 3.2).

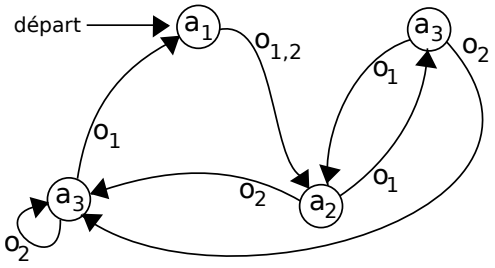


FIGURE 3.2 – Automate associé à la politique d'un agent

Là encore, les nœuds représentent les actions à effectuer et les arêtes représentent les observations que l'agent peut recevoir après avoir agi. Dans notre exemple, l'agent commence par exécuter l'action $a_1$, puis $a_2$ et $a_3$. Le choix de la quatrième action par contre dépendra de l'historique de l'agent, puisqu'il pourra se trouver dans le nœud en bas à gauche, ou dans celui en haut à droite. Cette représentation pose parfois un problème : il n'est en général pas possible de représenter une politique optimale. En effet, une politique optimale associerait, à tout historique $h$, une action $\pi(h)$. Une telle approche nécessiterait l'usage d'automates infinis. On ne peut donc représenter qu'une politique approximée : plus on ajoutera de nœuds à l'automate, et plus la politique sera proche de l'optimal. Là encore, la politique jointe sera représentée par un ensemble d'automates, un par agent. Ainsi, le choix du mode de représentation pour la politique de l'agent dépendra du problème que l'on tente de résoudre : des arbres pour les problèmes à horizon fini, ou des automates pour les problèmes à horizon infini.

### Calcul de la politique jointe

Tout comme pour un MMDP, il est impossible d'obtenir un comportement de groupe optimal si on calcule séparément la politique de chaque agent. Il faut donc calculer, de manière centralisée, une politique jointe. Il n'existe cependant, pour l'instant, aucun algorithme capable de calculer directement une politique jointe optimale pour un DEC-POMDP, à la façon du Value-Iteration pour un MDP. La seule approche est donc d'énumérer toutes les politiques jointes possibles, puis de calculer la valeur de chacune d'elles afin de conserver la meilleure. Afin de calculer la valeur d'une politique, on utilise en général une équation dérivée de l'équation de Bellman. Posons $a(\pi)$ l'action (jointe) renvoyée par la politique jointe $\pi$ (dans un arbre, c'est l'action au sommet et dans un automate, c'est l'action associée au nœud actuel). Posons ensuite $\pi \rightarrow o$ la politique jointe issue de l'observation jointe $o$ (dans un arbre, c'est le sous-arbre relié au sommet par la branche $o$ et dans un automate, c'est le même automate dans lequel on a changé de nœud en passant par l'arête $o$). On définit alors la valeur $V$ d'un état $s$ selon une politique $\pi$ :

$$V^{\pi}(s) = R(s,a(\pi)) + \gamma \sum_{s' \in S} \left( T(s,a(\pi),s').\sum_{o \in \Omega} O(s,a(\pi),s',o).V^{\pi \rightarrow o}(s') \right)$$

Cette équation permet donc de calculer l'espérance de gain pour une politique jointe donnée, en intégrant l'observabilité partielle. Cette opération n'est toutefois pas triviale, la complexité du calcul de $V(s)$ pour une politique donnée à horizon $h$ étant en $O(|S|^h.|\Omega|^h)$. De plus, en général, la taille de $S$ augmente exponentiellement en le nombre d'agents. Le coût pour estimer la valeur d'une politique donnée est donc doublement exponentiel, en l'horizon et en le nombre d'agents (dans le cas d'un automate, l'exponentielle est en la taille de l'automate).

Le calcul de la valeur d'une politique n'est pas la seule source de complexité. Posons $A_i$ les actions (individuelles) d'un agent $i$ et $\Omega_i$ ses observations (là encore, individuelles). On calcule le nombre de politiques possibles (dans le pire cas) pour cet agent, à horizon $h$ :

1. À horizon 0, on a $|A_i|$ politiques possibles (car une politique à horizon 0 est représentée par une simple action).
2. À horizon 1, il y a $|\Omega_i|$ branches et il faut choisir une action dans $A_i$ pour chaque branche. On a donc $|A_i|^{|\Omega_i|}$ choix possibles. On a alors $|A_i| \times |A_i|^{|\Omega_i|}$ politiques possibles à horizon 1 (car pour chaque politique à horizon 0, il y a $|A_i|^{|\Omega_i|}$ choix possible).
3. À horizon 2, il y a $|\Omega_i|^2$ branches (nombre de séquences de taille 2 sur $|\Omega_i|$ éléments). Ainsi, en appliquant le même raisonnement que précédemment, il y a $|A_i|^{|\Omega_i|^2}$ choix possibles, donc $|A_i| \times |A_i|^{|\Omega_i|} \times |A_i|^{|\Omega_i|^2}$ politiques possibles.
4. On peut ainsi généraliser : à horizon $k$, il y a $|\Omega_i|^k$ branches d'où $|A_i|^{|\Omega_i|^k}$ choix. On a donc $|A_i| \times |A_i|^{|\Omega_i|} \times |A_i|^{|\Omega_i|^2} \times \cdots \times |A_i|^{|\Omega_i|^k}$ politiques possibles.
5. Ainsi, à horizon $h$, on a $|A_i|^{\sum_{k=0}^{h} |\Omega_i|^k} = |A_i|^{\frac{|\Omega_i|^{h+1}-1}{|\Omega_i|-1}}$ politiques possibles.

On a alors un nombre de politiques jointes, dans le pire cas, en $O(\prod_{i=1}^{I} |A_i|^{\frac{|\Omega_i|^{h+1}-1}{|\Omega_i|-1}})$. Il existe différentes techniques d'élagage permettant de diminuer le nombre de politiques à générer mais, dans le pire cas, on retombe toujours sur cette complexité. On pose $\alpha = \max_{1 \leq i \leq I} |A_i|$ (et de même $\omega = \max_{1 \leq i \leq I} |\Omega_i|$). Comme il est nécessaire de tester la valeur de chaque politique jointe pour déterminer la meilleure, la complexité totale est en :

$$O\left(|S|^h.|\Omega|^h.\prod_{i=1}^{I} |A_i|^{\frac{|\Omega_i|^{h+1}-1}{|\Omega_i|-1}}\right) = O\left(|S|^h.|\Omega|^h.\alpha^{I.\omega^{h+1}}\right)$$

Au final, $h$ étant constant, la complexité de résolution d'un DEC-POMDP est de classe NEXP-complet [11]. Ainsi, calculer une politique optimale pour un DEC-POMDP de taille « raisonnable [12] » est, dans le cas général, considéré comme très difficile. Nous reviendrons sur la question des différents types d'algorithme de résolution dans le chapitre suivant.

11. NEXP : problème non déterministe, dont une solution peut être vérifiée en temps EXP, c'est à dire en $O(2^{p(n)})$ avec $p(n)$ un polynôme en la taille du problème.
12. DEC-POMDP de taille raisonnable : avec au moins 4 agents.

## 3.2 Classes de DEC-POMDP

Nous avons vu, dans la section précédente, en quoi la résolution d'un DEC-POMDP dans le cas général est considérée comme très dur (hormis pour des problèmes « jouets », avec 2 ou 3 agents, peu d'actions possibles, etc.). Plusieurs hypothèses ont cependant été proposées qui permettent, lorsqu'elle sont vérifiées, de diminuer (parfois considérablement) la complexité du problème à résoudre. Dans cette section, nous présentons l'ensemble des hypothèses permettant de caractériser la difficulté d'un problème. Nous verrons alors, dans la section suivante, comment le modèle DEC-POMDP peut être modifié selon que l'on vérifie ou non chacune d'elles.

### 3.2.1 Hypothèses d'indépendance et impact sur la complexité

Nous avons, jusqu'à présent, supposé une corrélation totale entre agents. On pourrait cependant imaginer que les agents soient (au moins partiellement) indépendants les uns des autres.

#### Notion de problème décomposable

Nous avons vu précédemment que les états joints pouvaient être, ou non, décomposables entre agents. On peut, par exemple, prendre un problème de navigation : un ensemble d'agents doivent se rendre à un point donné, sans se cogner entre eux. L'état joint, dans un tel problème, décrira les positions de l'ensemble des agents. Un tel état peut alors se décomposer en $s = (s_1, \ldots, s_I)$ avec $s_i$ l'état de l'agent $i$, tel que l'état d'un agent décrive sa position.

Un problème est **décomposable** lorsque l'ensemble des états, actions et observations sont décomposables entre agents. Dès l'instant où on dispose d'un problème décomposable, on peut s'intéresser à la distributivité des fonctions de transition et de récompense : est-il toujours nécessaire de les exprimer sur l'ensemble des $(s,a,s')$ avec $s$ et $s'$ des états joints (et $a$ une action jointe), ou peut-on les exprimer comme une composée de fonctions sur des actions et états individuels ?

#### Dépendances sur les transitions et récompenses

La fonction de transition d'un DEC-POMDP se défini, dans le cas général, sur $S \times A \times S$. Il existe toutefois certains problèmes pour lesquels on peut poser l'hypothèse d'une indépendance sur les transitions [Becker *et al.*, 2003, Becker *et al.*, 2004b].

**Définition 24 (Transitions indépendantes)** *On considère qu'un DEC-POMDP vérifie l'hypothèse d'indépendance sur les transitions si pour tous états joints $s$ et $s'$ et toute action jointe $a$ on peut écrire :*

$$T(s,a,s') = \prod_{i=1}^{I} T_i(s_i,a_i,s'_i)$$

Ainsi, lorsqu'un problème vérifie cette hypothèse, chaque agent dispose d'une fonction de transition individuelle. On peut alors prédire les transitions d'un agent donné en se basant uniquement sur ses actions (sans avoir à considérer les actions des autres agents). Cette hypothèse est systématiquement nécessaire pour réduire la complexité des DEC-POMDPs, mais non suffisante (voir figure 3.4). Il existe également des problèmes pour lesquels la fonction de récompense est décomposable entre agents.

**Définition 25 (Récompenses indépendantes)** *Un DEC-POMDP pour lequel on vérifie l'hypothèse d'indépendance sur les récompenses implique, pour tous états $s$ et $s'$ et toute action $a$ :*

$$R(s,a,s') = \sum_{i=1}^{I} R_i(s_i,a_i,s'_i)$$

Ainsi, la récompense jointe sera égale à la somme des récompenses individuelles, chaque agent apportant un gain (ou une pénalité) au groupe via ses actions. On peut également envisager des cas plus complexes, où les récompenses ne seront pas complètement indépendantes mais divisées en groupes. On aura alors un ensemble de groupes $G = \{g_1, \ldots, g_{|G|}\}$ tels que chaque $g_i$ contienne un ensemble d'agents (un agent pouvant appartenir à plusieurs groupes en même temps). On aura alors $R(s,a,s') = \sum_{g \in G} R_g(s_g,a_g,s'_g)$. Nous verrons par la suite que cette hypothèse seule n'a aucun impact sur la complexité du problème [Allen, 2009] mais qu'elle permet, lorsqu'elle est vérifiée conjointement à une indépendance sur les transitions, de casser la complexité combinatoire d'un DEC-POMDP (figure 3.4).

### 3.2.2 Influence de l'observabilité

La question de l'observabilité a une influence forte sur la complexité d'un DEC-POMDP, que les transitions et récompenses impliquent ou non de la dépendance. Supposons à nouveau que le problème soit décomposable entre agents : on peut alors envisager une indépendance sur les observations.

#### Observabilité disjointe

Dans le cas général, la fonction d'observation dépend des transitions jointes. Ainsi, un agent ne recevra pas la même observation selon que les autres agents aient exécutés telle ou telle action. Il existe cependant des problèmes pour lesquels on peut décomposer cette fonction d'observation.

**Définition 26 (Observations indépendantes)** *On dit d'un DEC-POMDP qu'il vérifie l'hypothèse d'indépendance sur les observations si, pour tous états joints $s$ et $s'$, toute action jointe $a$ et toute observation jointe $o$, on a :*

$$O(s,a,s',o) = \prod_{i=1}^{I} O_i(s_i,a_i,s'_i,o_i)$$

Ainsi, lorsqu'un problème vérifie cette hypothèse, les agents reçoivent des observations relatives à leur état individuel, en fonction de leurs actions et sans que les autres agents n'aient la moindre influence. Cela n'implique pas nécessairement une indépendance totale entre agents. Prenons l'exemple classique d'un réseau de capteurs [Nair *et al.*, 2005] : on imagine un bâtiment dans lequel plusieurs capteurs ont été disposés. Deux intrus se déplacent dans le bâtiment et on cherche à les localiser. Pour cela, il faut qu'au moins deux capteurs le détectent en même temps. La figure 3.3 donne un exemple d'application de ce problème.

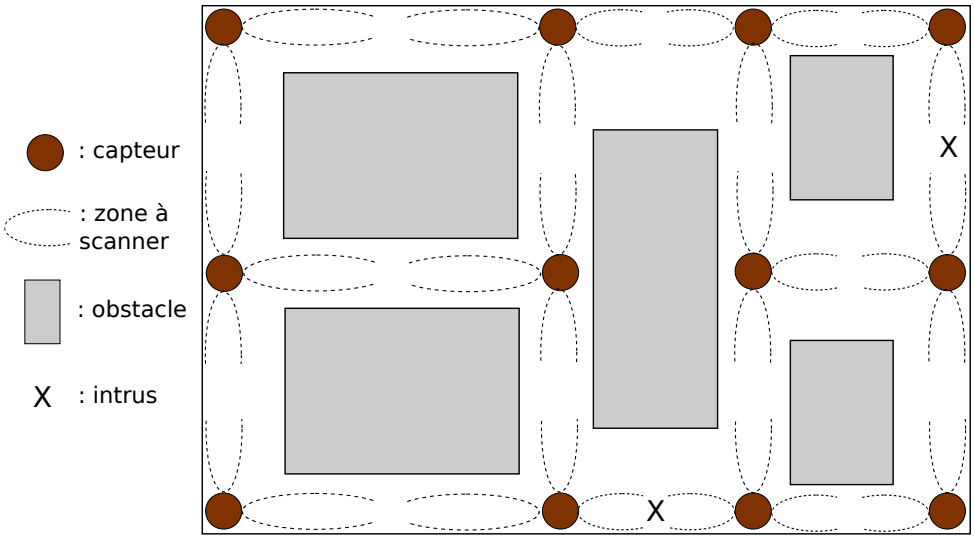


FIGURE 3.3 – Problème du réseau de capteurs

Ici, on représente chaque capteur par un agent dont les cinq actions possibles sont : scanner au nord, à l'est, à l'ouest ou au sud et ne rien faire. L'état d'un agent est un triplet ($pos1$,$pos2$,$detect$) tel que $pos1$ donne la position du premier intrus, $pos2$ la position du second intrus et où $detect = vrai$ si on a détecté un intrus après avoir scanné, $faux$ si on a rien détecté. Un agent qui scanne une position où se trouve un intrus aura une probabilité forte de le détecter et une probabilité faible d'avoir un faux négatif (de la même façon, en scannant une zone où aucun intrus ne se trouve, on a une probabilité forte de ne rien détecter et une probabilité faible d'avoir un faux positif). On considère que les intrus se déplacent à chaque pas de temps, dans une direction aléatoire. Scanner a un coût, contrairement à l'action « ne rien faire ». Finalement, lorsque deux agents détectent en même temps un intrus au même endroit, ils reçoivent une récompense.

Ici, la fonction de transition est indépendante (le fait de détecter ou non l'intrus ne dépend pas des autres agents) ainsi que la fonction d'observation (la probabilité d'observer l'intrus ne dépend que de notre action). Par contre, la fonction de récompense est partiellement dépendante (définie sur un ensemble de groupes : la récompense de chaque agent dépend des agents voisins, puisque l'on ne reçoit une récompense que si deux agents détectent l'intrus en même temps). Ici, la complexité est toujours NEXP-C, mais en la taille du plus grand groupe d'agents et non en $I$ (nombre d'agents impliqués dans le problème).

#### Renforcement de l'observabilité via la communication

Que l'observabilité soit (ou non) disjointe, la quantité d'informations observées aura une influence directe sur la complexité du problème. En effet, plus l'agent connaît de choses avec certitude, plus son état de croyance est petit (car la distribution de probabilités s'applique sur un plus petit nombre d'éléments). Dans le cas le plus extrême, l'agent peut observer la totalité de l'environnement : le DEC-POMDP se réduit alors à un MMDP et peut donc être résolu beaucoup plus facilement. Les agents peuvent donc décider d'échanger leurs observations afin d'augmenter leurs connaissances respectives [Goldman et Zilberstein, 2003].

Prenons l'exemple d'un groupe d'agents évoluant dans un bâtiment : chaque agent pourra observer la pièce dans laquelle il se trouve, mais pas le reste du bâtiment. Si ces agents peuvent communiquer, via une radio par exemple, ils peuvent s'échanger leurs observations et ainsi obtenir une connaissance sur un ensemble de pièces. Cette communication est toutefois rarement gratuite, ni illimitée. On a par exemple des problèmes dans lesquels communiquer consomme de l'énergie et entraîne donc un coût. On a également des problèmes où la bande passante disponible est limitée (avec des radios par exemple, deux agents ne peuvent pas communiquer en même temps sur le même canal).

Ainsi, lorsque la communication est possible, les agents doivent raisonner sur le coût de la communication, relativement au gain en information, afin de prendre ou non la décision de communiquer. On notera entre autres le cas extrême des DEC-POMDPs décomposables, lorsque l'agent dispose d'une observabilité totale sur son état individuel : si on ajoute une communication gratuite et illimitée à de tels problèmes, on considère que les agents disposent alors d'une observabilité totale (d'où un passage au MMDP). Cette hypothèse est toutefois peu réaliste : il est courant lorsque l'on traite des problèmes de robotique, au vu des coûts importants en énergie, que la communication soit complètement interdite.

### 3.2.3 Classification des DEC-POMDPs

Il est possible, en se basant sur de telles hypothèses, d'établir une classification des DEC-POMDPs. Nous verrons ainsi comment, selon le problème traité, on peut diminuer la complexité de NEXP-complet dans le pire cas jusqu'à P-complet dans le meilleur des cas. Cette analyse repose en partie sur le travail réalisé dans [Goldman et Zilberstein, 2004].

#### Hypothèses exotiques

Traditionnellement, un DEC-POMDP permet l'optimisation d'un problème dont chaque état rapporte une récompense plus ou moins élevée. Il existe toutefois des problèmes particuliers [Goldman et Zilberstein, 2005] dans lesquels un état unique (que l'on nommera « but ») apporte une récompense (les autres états étant associés à une récompense de 0). On pose de plus un certain nombre d'hypothèses : chaque action a un coup, l'état but est unique et atteindre le but entraîne l'arrêt de l'exécution du problème. Ce type de problème sera en général plus simple

à résoudre, puisque l'on sait comment orienter la résolution. Cette hypothèse, jointe à une indépendance sur les transitions et observations, réduit la complexité au domaine NP-complet.

Une autre hypothèse permet de réduire la complexité des DEC-POMDPs : il s'agit de la coordination basée tâche. Dans un problème vérifiant cette hypothèse, les agents sont mutuellement indépendants. Ils partagent cependant un ensemble de tâches dont la réalisation peut influer sur leurs fonctions de transition, de récompense et d'observation. Il s'agit donc d'une situation intermédiaire, plus simple à résoudre qu'un DEC-POMDP standard (car la coordination se fait seulement sur l'attribution des tâches) mais également moins expressive (nous verrons par la suite que l'on ne peut représenter que certains types de problèmes par cette approche).

#### Taxinomie des DEC-POMDPs

La figure 3.4 propose une taxinomie des différentes classes de DEC-POMDP, selon les hypothèses vérifiées. Bien entendu, cette figure ne regroupe pas l'ensemble des modèles existants, ceux-ci étant particulièrement nombreux. Elle ne contient de plus que les liens les plus classiques entre les différentes classes, afin de ne pas surcharger la visibilité. Elle permet cependant de voir comment, selon que l'on vérifie telle ou telle hypothèse, on peut casser la complexité de résolution.

## 3.3 Modèles dérivés

Plusieurs modèles ont été proposés, dérivés du modèle DEC-POMDP et permettant de représenter des classes de problème particulières (selon les hypothèses que l'on admet). Dans cette section, nous présentons certains de ces modèles en précisant les avantages et inconvénients de chacun. Nous n'effectuerons pas une énumération exhaustive des modèles existants (ceux-ci étant particulièrement nombreux), mais plutôt une présentation des grandes tendances dans le domaine illustrée par les modèles les plus utilisés au sein de la communauté. Nous avons de plus fait le choix de répartir ces modèles en deux catégories : les problèmes « orientés indépendance », et les problèmes « orientés interaction ». Dans le premier cas, on pose dès le départ les hypothèses d'indépendance (tel agent est en interaction avec tel autre, sur les récompenses et non les transitions, etc.) puis on exploite ces hypothèses pour structurer la résolution du problème. Dans le second cas, on ne pose aucune hypothèse à priori, mais on considère que les interactions évoluent durant l'exécution du problème et on base la politique de l'agent sur cette évolution.

### 3.3.1 Modèles basés indépendance (interactions statiques)

Les modèles basés indépendance ont pour point commun de poser dès le départ une structure d'interaction (qui n'évoluera plus), puis d'utiliser cette structure durant la résolution du problème. Une telle approche permet alors une résolution plus efficace mais ne peut s'appliquer qu'à certains problèmes (l'exemple du réseau de capteurs, donné page 46, est ici particulièrement adapté : les agents étant immobiles, les interactions n'évoluent pas).

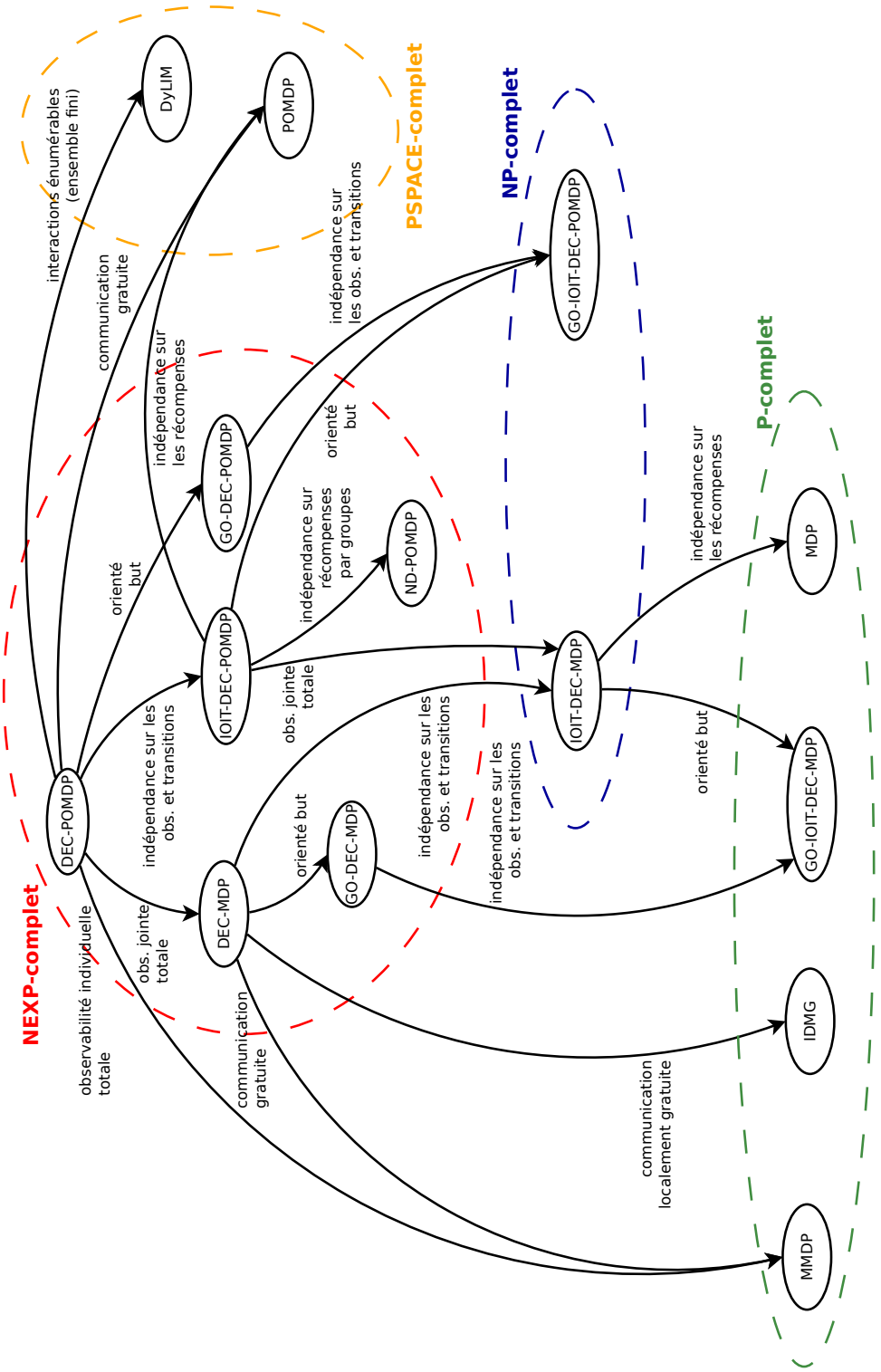


FIGURE 3.4 – Taxinomie des DEC-POMDPs

### DEC-POMDPs factorisés

Le modèle des DEC-POMDPs factorisés [Guestrin *et al.*, 2002, Kok *et al.*, 2005, Oliehoek *et al.*, 2008] permet une représentation compacte du problème, en fournissant une description précise des dépendances entre agents. L'idée, pour la représentation factorisée d'un DEC-POMDP, est la suivante :

- l'ensemble des états joints est représenté par un ensemble de variables, tel qu'un état sera égal à une valuation de ces variables (associer une valeur à chacune). Ainsi, on n'énumère plus l'ensemble des états joints possibles mais seulement l'ensemble des variables et des valeurs qu'elles peuvent prendre.
- Une transition consiste en la modification de la valeur d'une ou plusieurs variables, en fonction de l'action exécutée et des valeurs précédentes. On peut par exemple prendre l'exemple d'un groupe de véhicules autonomes : si on associe (entre autres) à chaque agent une variable représentant le niveau de carburant, l'action « avancer » diminuera la valeur de cette variable.
- On associe une récompense et une fonction d'observation à chaque transition. La fonction de récompense est alors décomposée entre variables (chaque changement de valeur apporte une récompense, la récompense totale étant égale à la somme de ces récompenses locales).

La figure 3.5 représente un DEC-POMDP factorisé. Sur cette figure, les variables d'état sont représentées par des losanges et les actions des agents sont représentées par des ronds. On voit alors, par exemple, que l'agent 0 reçoit une observation en fonction des variables 1 et 2. On voit également que la valeur de la variable 2 évolue en fonction des agents 0 et 1 uniquement.

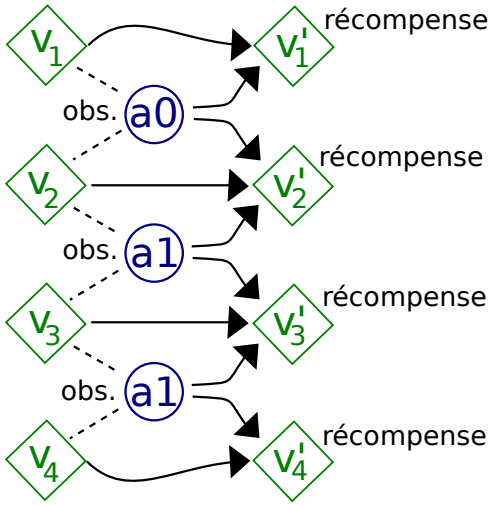


FIGURE 3.5 – DEC-POMDP factorisé

On a donc, avec ce type de représentation, une localité de l'interaction. Cette approche offre de bonnes performances (Oliehoek et al. ont mené des expériences montrant des temps de calcul divisés par 100), mais nécessite des hypothèses fortes : le problème doit être factorisable, et les interactions ne doivent pas évoluer dans le temps.

**Network Distributed POMDPs**

Le modèle ND-POMDP [13] [Nair *et al.*, 2005, Varakantham *et al.*, 2007, Kumar et Zilberstein, 2009] a été introduit pour la représentation des problèmes respectant trois hypothèses : indépendance sur les transitions, indépendance sur les observations et indépendance par groupes sur les récompenses. On définit alors un ND-POMDP par un n-uplet $\langle I,G,S,A,T,R,\Omega,O\rangle$ où :

- $I$ est le nombre d'agents impliqués dans le problème,
- $G = \{g_1,\ldots,g_{|G|}\}$ est un ensemble de groupes d'agents tels que tout agent appartient à au moins un groupe $g_i \in G$ (un agent peut appartenir à plusieurs groupes),
- $S = S_u \times S_1 \times S_2 \times \cdots \times S_I$ décrit l'ensemble des états joints avec $S_i$ les états individuels de l'agent $i$ et $S_u$ les états « incontrôlables » (un état incontrôlable est un état dont l'évolution ne dépend pas des actions des agents) : un état $s$ sera donc toujours égal à un n-uplet $s = (s_u,s_1,\ldots,s_I)$,
- $A = A_1 \times \cdots \times A_I$ décrit l'ensemble des actions jointes ($A_i$ les actions de l'agent $i$),
- $T : S \times A \times S \to [0,1]$ est une fonction de transition jointe telle que toute transition peut s'écrire $T(s,a,s') = T_u(s_u,s'_u) \times \prod_{i=1}^{I} T_i(s_i,a_i,s'_i)$ : la fonction $T$ se divise donc en une fonction $T_u$ et un ensemble de fonctions individuelles $T_i$,
- $R : S \times A \to \mathbb{R}$ est une fonction de récompense jointe avec $R(s,a) = \sum_{g\in G} R_g((s_u,s_g),a_g)$ où $s_g = (s_i | i \in g)$ décrit l'état $s$ réduit aux agents présents dans le groupe $g$ : la fonction $R$ se divise donc en un ensemble de fonctions de récompense locales, une par groupe d'agents,
- $\Omega = \Omega_1 \times \cdots \times \Omega_I$ est un ensemble d'observations jointes,
- $O : S \times A \times S \times \Omega \to [0,1]$ est une fonction d'observation jointe que l'on définit par l'égalité $O(s,a,s',o) = \prod_{i=1}^{I} O_i((s_u,s_i),a_i,(s'_u,s'_i),o_i)$ : cette fonction se divise donc en un ensemble de fonctions d'observation individuelles $O_i$.

Le modèle ND-POMDP permet de séparer le problème joint en un ensemble de problèmes individuels, en divisant les fonctions de transition et d'observation. Chaque agent $i$ aura alors un état $(s_u,s_i)$ sur lequel appliquer ces fonctions. La coordination devra se faire sur les récompenses, jointes par groupe. Au vu de ces propriétés, les algorithmes existants permettent de résoudre un ND-POMDP comme un ensemble de DEC-POMDPs (un par groupe d'agents), puis d'appliquer des méthodes de fusion de politiques afin d'obtenir une politique globale à partir des politiques locales à chaque groupe d'agents. On peut alors traiter des problèmes impliquant de nombreux agents, mais on reste limité par la taille des groupes (2 ou 3 agents par groupe au maximum). Dans le cas extrême ou chaque groupe contient exactement 1 agent, la complexité en temps retombe dans la classe NP-complet.

### 3.3.2 Modèles basés interaction (interactions dynamiques)

À l'inverse des modèles basés indépendance, les modèles basés interaction ne supposent aucune structure d'interaction au préalable. L'idée est plutôt de considérer que les interactions

13. ND-POMDP : de l'anglais « Network Distributed POMDP ».

vont évoluer durant l'exécution du problème : chaque action entraînera non seulement une modification de l'état, mais également une modification de la structure d'interaction.

### Distributed POMDPs with Coordination Locales

Le modèle DPCL [14] [Varakantham *et al.*, 2009] permet la représentation de problèmes pour lesquels la coordination entre agents se fait uniquement via une allocation de tâches. On définit un DPCL par un tuple $\langle I,S,A,T,R,\Omega,O\rangle$ dans lequel :

- $I$ donne le nombre d'agents,
- $S = S_g \times S_1 \times \cdots \times S_I$ est un ensemble d'états joints avec $S_i$ les états individuels de l'agent $i$. $S_g = E \times S_t$ est un ensemble « d'états d'avancement » globaux, tels que $E = \{e_1, \ldots, e_h\}$ est un ensemble de pas de temps et $S_t$ est un ensemble de tâches pouvant être dans l'état « terminé », ou « en cours ». À tout moment, l'état d'un agent sera alors un couple $(s_g,s_i) \in S_g \times S_i$ tel que $s_g$ décrive l'avancement des tâches (partagées par l'ensemble des agents) et $s_i$ décrive la situation personnelle de l'agent.
- $A = A_1 \times \cdots \times A_I$ est un ensemble d'actions jointes ($A_i$ les actions de l'agent $i$),
- pour tout agent $i$, $T_i : (S_g \times S_i) \times A_i \times (S_g \times S_i) \to [0,1]$ est une fonction de transition individuelle. La plupart du temps, on peut poser $T = \prod_{i=1}^{I} T_i$ (on verra plus loin qu'il existe des cas particuliers où cette décomposition est fausse),
- de la même façon, $R_i : (S_g \times S_i) \times A_i \times (S_g \times S_i) \to \mathbb{R}$ est la fonction de récompense individuelle de l'agent $i$ telle que, la plupart du temps, on pose $R = \sum_{i=1}^{I} R_i$ (là encore, il arrive que cette décomposition soit fausse),
- $\Omega = \Omega_1 \times \cdots \times \Omega_I$ est un ensemble d'observations jointes,
- $O_i : (S_g \times S_i) \times A_i \times (S_g \times S_i) \times \Omega_i \to [0,1]$ est la fonction d'observation individuelle de l'agent (on suppose une indépendance sur les observations, d'où $O = \prod_{i=1}^{I} O_i$).

La plupart du temps, ce modèle permet de supposer une indépendance totale entre agents. Il arrive cependant qu'en certains états, les agents doivent se coordonner. L'observabilité étant complètement disjointe, la coordination ne peut se faire que sur les états globaux (car on ne sait rien des états individuels des autres agents). Cette coordination peut alors se faire dans deux types de situations :

1. lorsque deux agents doivent travailler de concert pour terminer une tâche (points de coordination synchrones [15]), par exemple pour déplacer un objet trop lourd pour être porté par un agent seul,

2. lorsqu'un un agent termine une tâche et que cette action aura une influence future sur les autres agents (points de coordination future [16]), par exemple lorsqu'un agent ferme une porte, empêchant les autres agents de passer.

14. DPCL : de l'anglais « Distributed POMDPs with Coordination Locales ».
15. dans l'article : STCL, de l'anglais « Same-Time Coordination Locales ».
16. dans l'article : FTCL, de l'anglais « Futur-Time Coordination Locales ».

Ce modèle est donc parfaitement adapté à la représentation de problèmes basés tâche, mais s'avère limité dès que l'on veut représenter des problèmes plus complexes. Un exemple très simple est celui d'un groupe d'agents évoluant dans un espace clos, avec risque de collision. Chaque mouvement des agents pourra entraîner une collision avec les agents voisins, mais ce type d'interaction ne peut pas être représenté par l'exécution d'une tâche (à moins de disposer d'un ensemble infini de tâches, ce qui n'est pas le cas ici). Ce modèle s'avère donc performant lorsque l'on tente de résoudre des problèmes qui s'y prêtent, mais cette performance est présente au prix d'un manque d'applicabilité. D'autres approches basées tâches [Nair et Tambe, 2005, Ghavamzadeh *et al.*, 2006, Witwicki et Durfee, 2010] ou événements [Becker *et al.*, 2004a, Mostafa et Lesser, 2011] ont été proposées, toutes souffrant de ce manque d'applicabilité.

### Interaction Driven Markov Games

Le modèle IDMG [17] [Spaan et Melo, 2008] permet la représentation de problèmes dans lesquels les agents disposent d'une communication locale gratuite et illimitée (on définit une distance en dessous de laquelle les agents peuvent communiquer entre eux) et d'une observabilité locale complète (les agents ont une observabilité totale sur leur état individuel, mais aucune observabilité sur les états des autres agents). En contrepartie, ce modèle permet la manipulation d'interactions complexes (plus complexes qu'un ensemble de tâches à réaliser). Lorsque ces hypothèses sont vérifiées, on définit un IDMG par un n-uplet $\langle I, M_1, \ldots, M_I, \{M_m\}\rangle$ dans lequel :

- $I$ est un ensemble d'agents,
- $M_1, \ldots, M_I$ sont des problèmes individuels tels que $M_i$ décrit le problème de l'agent $i$, indépendamment des autres agents ($M_i$ est un jeu markovien à un seul agent, un modèle similaire aux MDPs),
- $\{M_m\}$ est un ensemble de problèmes multiagents, un par « zone d'interaction » (ici, chaque problème est un jeu markovien à $N$ agents, un modèle similaire aux MMDPs). La figure 3.6 donne un exemple applicatif pour ce modèle.

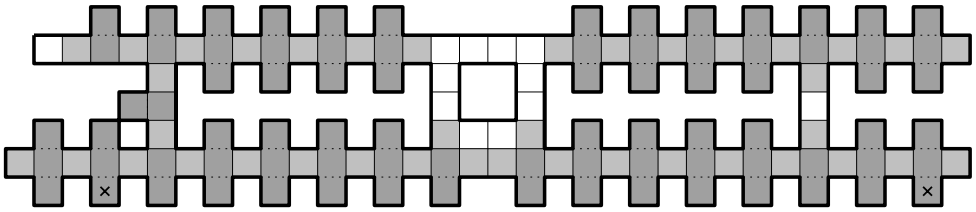

FIGURE 3.6 – Exemple d'application pour le modèle IDMG

Dans ce problème, deux agents doivent se rendre sur les cases indiquées par des croix, sans entrer en collision. On identifie un ensemble de zones d'interaction (une zone d'interaction est un ensemble d'états contigus) : cela signifie que, lorsque les deux agents sont en même temps dans cette zone, ils peuvent avoir un impact l'un sur l'autre. À l'inverse, on considère que deux agents

17. IDMG : de l'anglais « Interaction Driven Markov Games ».

n'étant pas dans la même zone d'interaction n'ont aucun impact l'un sur l'autre. Les agents ont alors une observabilité complète, mais uniquement sur leur état individuel. Ils se basent donc sur leur MDP pour agir. Lorsque les agents entrent dans une même zone d'interaction, ils communiquent afin d'acquérir une observabilité jointe complète. Ils peuvent alors utiliser leur MMDP pour agir. Les tests réalisés par Spaan et al. ont montré que, si les zones d'interaction sont suffisamment étendues, la qualité du comportement joint est quasi-optimale. La figure 3.7 montre un exemple de découpage en zones de taille réaliste.

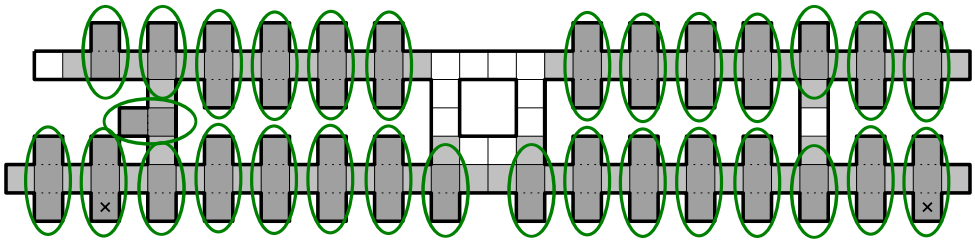

FIGURE 3.7 – Exemple de zones d'interaction

Ce découpage en 30 zones est cependant trop fin pour permettre des politiques de bonne qualité (puisque les agents sont obligés d'attendre le dernier moment pour se coordonner). Les tests réalisés par Spaan et al. ont montré qu'une politique jointe optimale sur ce problème aura une valeur espérée de 5,84 (lorsque l'on donne une observabilité jointe totale aux agents) et qu'une politique au pire cas (lorsque les agents n'ont aucune observabilité sur les autres agents) aura une valeur de -3,16. La politique résultante de ce découpage a une valeur espérée de -2,21 (ce qui représente seulement 9,5% de la valeur optimale). Afin d'augmenter cette valeur, il est nécessaire d'augmenter la taille des zones d'interaction. La figure 3.8 montre le découpage obtenu lorsque l'on veut atteindre quasiment 100% de la valeur optimale.

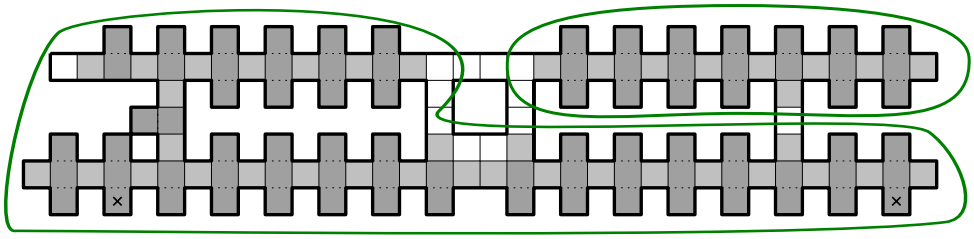

FIGURE 3.8 – Zones d'interaction étendues

Les résultats sont alors très bon, avec une politique ayant une valeur espérée de 5,81. Ce découpage en 2 zones est cependant peu réaliste, puisqu'il implique que les agents soient capables de communiquer de façon gratuite et illimitée dans la quasi totalité de leur environnement. Le modèle IDMG offre donc une approche puissante pour la résolution des DEC-POMDPs, lorsque les hypothèses de communication locale gratuite et d'observabilité locale complète sont vérifiées, mais sa mise en œuvre révèle des besoins peu réalistes, notamment en ce qui concerne la taille des zones d'interaction.

### Decentralized MDPs with sparse interactions

Le modèle DEC-SIMDP [18] [Melo et Veloso, 2011] est une extension récente du modèle IDMG. L'idée est donc de représenter séparément le problème individuel et les différents problèmes d'interaction, par un n-uplet $(\{M_k,\ k = 1,\ldots,I\},\{(S_i^z,M_i^z),\ i = 1,\ldots,N\})$ dans lequel :

- chaque $M_k$ est un MDP, décrivant le problème individuel de l'agent $k$,
- chaque $S_i^z$ est un ensemble d'états joints, correspondant à la zone d'interaction $i$ (le terme « zone d'interaction » ayant ici la même signification que pour le modèle IDMG),
- chaque $M_i^z$ est un MMDP, décrivant le problème associé à la zone $S_i^z$ (une zone implique $j \leq I$ agents, et le MMDP associé est défini sur l'ensemble des états joints impliquant ces $j$ agents, même les états joints hors de la zone).

Ainsi, lorsque l'agent est dans la même zone qu'un de ses voisins, il peut l'observer (observabilité locale complète) et suivre le MMDP associé. À l'inverse, lorsqu'il n'y a pas interaction, on applique le MDP de l'agent.

Le modèle IDMG ne permet de traiter les interactions que localement (lorsque les voisins ne sont plus dans la zone d'interaction, on les oublie complètement). Cela peut poser problème, dans le sens où on les observe parfois au dernier moment, rendant impossible toute coordination efficace. Ici, à l'inverse, on essaye de raisonner « à long terme » sur les interactions, plutôt que via une résolution locale. Pour rendre ce raisonnement possible, on garde un état de croyance sur les états des agents hors de la zone d'interaction.

Cette méthode a fourni de bons résultats (politiques de qualité similaire à celles obtenues via le modèle IDMG, tout en nécessitant des temps de calcul moindres), mais nécessite la vérification des mêmes hypothèses. Il faut donc disposer d'une observabilité locale complète, la localité correspondant ici à l'ensemble des états associés à la zone d'interaction dans laquelle se situe l'agent. Cette observabilité peut être matérialisée au travers d'une communication gratuite et illimitée par exemple, et s'applique parfois sur des zones d'interaction de très grande taille. Ainsi, si on traite des problèmes complexes (avec, par exemple, des agents évoluant dans un espace ouvert et étant amenés à se croiser n'importe où et avec n'importe qui), on fini par n'avoir plus qu'une seule zone d'interaction, recouvrant l'ensemble des états et des agents. L'hypothèse d'observabilité complète devient alors peu réaliste.

### Bilan sur les principaux modèles

Le tableau 3.1 résume les qualités et inconvénients des différents modèles présentés dans les page précédentes (DEC-POMDP, ND-POMDP, MMDP, IDMG/DEC-SIMDP, DPCL et DEC-POMDP-Factorisé), auxquels vient s'ajouter notre modèle, que nous présenterons dans la suite de ce document : DyLIM. Cette comparaison se fait sur un ensemble de critères d'applicabilité d'une part (observabilité nécessaire, besoins en communication, dépendances gérées entre agents et structures d'interaction reconnues), de qualité d'autre part (passage à l'échelle).

18. DEC-SIMDP : de l'anglais « Decentralized MDPs with Sparse Interactions ».

| Modèles / paramètres | Observabilité nécessaire | Besoins en communication | Dépendances gérées | Structure d'interaction | Passage à l'échelle |
|---|---|---|---|---|---|
| DEC-POMDP | partielle | non | toutes | aucune | non |
| ND-POMDP | partielle | non | récompenses par groupe | statique | moyen |
| MMDP | totale | non | toutes | aucune | oui |
| IDMG, DEC-SIMDP | locale | localement gratuite | récompenses, transitions | dynamique | oui |
| Factored DEC-POMDP | partielle | non | toutes | statique | moyen |
| DPCL | partielle | non | allocation de tâches | dynamique | oui |
| DyLIM | partielle | non | toutes | dynamique | oui |

TABLE 3.1 – Analyse comparative des principaux modèles de type « DEC-POMDP ».

## Conclusion

Nous avons présenté, dans ce chapitre, le modèle DEC-POMDP pour la représentation de problèmes de planification multiagent en environnement stochastique (respectant la propriété de Markov) et partiellement observable. Nous avons notamment vu comment, en posant un certain nombre d'hypothèses, on peut utiliser des modèles dérivés permettant de réduire la complexité d'un DEC-POMDP, passant ainsi de NEXP-complet à P-complet (dans le meilleur des cas).

Les modèles reposant sur des interactions complexes et dynamiques (tels que IDMG, ou DEC-SIMDP) sont probablement les plus intéressants, pour passer à l'échelle en termes de nombre d'agents sans limiter l'applicabilité. On suppose, dans ces deux approches, que l'agent dispose d'une observabilité locale complète. Il suffit donc que l'agent diffuse cette information dans son voisinage pour que chacun dispose d'une observabilité totale sur l'état joint. On peut ainsi traiter chaque « zone d'interaction » comme un problème multiagent complètement observable.

Que se passe t'il, lorsque même l'observabilité locale de l'agent est partielle ? C'est par exemple le cas dans un grand nombre d'applications robotiques, où l'agent se localise relativement aux obstacles avoisinants. Dans ce type de situation, chaque « zone d'interaction » devra être traitée comme un problème multiagent partiellement observable. Résoudre ce type de problème est alors très difficile (NEXP-Complet), même lorsque peu d'agents sont impliqués.

On ne peut donc pas se contenter de ces modèles pour traiter les problèmes impliquant une dépendance entre agents sur les transitions, récompenses et observations, dans lesquels l'observabilité locale est partielle et où les interactions mises en jeu sont plus poussées qu'une « simple » allocation de tâches. C'est sur ce constat que se basent nos travaux, via la proposition d'un modèle permettant de traiter de tels problèmes. Cet état de l'art s'achèvera sur le chapitre suivant, introduisant les différents algorithmes de résolution pour les modèles présentés ici.

# Chapitre 4

# Les différentes familles d'algorithmes de résolution

**Sommaire**



Le chapitre précédent a permit l'introduction du modèle DEC-POMDP et d'un ensemble de modèles dérivés. Nous allons maintenant présenter les algorithmes disponibles pour la résolution de problèmes exprimés via ces modèles : là encore, nous ne fournirons pas une énumération exhaustive des algorithmes existants, mais plutôt une description des différentes approches possibles. Nous verrons notamment comment exploiter les hypothèses propres à chaque modèle pour simplifier la résolution. Nous insisterons en particulier sur l'usage des interactions lors de la résolution du problème, cet aspect étant central dans nos travaux.

## 4.1 Approches classiques

Cette section propose une introductions aux algorithmes « standards » de résolution de DEC-POMDP. Ces approches, bien qu'inapplicables à la résolution de problèmes réels (car étant destinées à la résolution du modèle standard), constituent une base théorique intéressante pour la résolution de ce type de problème.

### 4.1.1 Résolution exacte à horizon fini

Les approches à horizon fini sont en général les plus efficaces pour la résolution de DEC-POMDPs. On considère donc que l'on dispose d'un DEC-POMDP pour lequel on doit calculer une politique jointe à horizon $h$ donné. On choisit la représentation en arbre pour la politique d'un agent (et donc un ensemble d'arbres pour la politique jointe). Il est impossible de calculer directement une politique jointe pour un DEC-POMDP, comme on l'aurait fait en résolvant un MDP. L'approche la plus évidente pour trouver cette politique jointe consiste alors à énumérer l'ensemble des politiques possibles, puis calculer la valeur espérée de chaque politique, afin de garder la meilleure (**approche naïve**). Nous avons cependant vu dans le chapitre 3 qu'il était extrêmement difficile de procéder ainsi, le nombre total de politiques jointes étant en $O\left(\alpha^{I.\omega^{h+1}}\right)$, avec $\alpha = \max_{1\leq i\leq I} |A_i|$ et $\omega = \max_{1\leq i\leq I} |\Omega_i|$. Dans cette section, nous présenterons les deux familles d'algorithmes les plus courants pour la résolution exacte (par programmation dynamique, puis par heuristique).

#### Programmation dynamique

La fonction de valeurs d'une politique (jointe ou non) est construite récursivement. On peut, en se basant sur cette structure, déduire le principe de Bellman « toute solution optimale est construite à partir de sous-solutions optimales ». Ainsi, si on prend une politique optimale à horizon $h$, chaque sous politique (= chaque branche de l'arbre) sera une politique optimale à horizon $h-1$. C'est en se basant sur ce constat qu'ont été introduits les algorithmes de résolution par **programmation dynamique** [Hansen *et al.*, 2004]. L'idée, pour résoudre le problème à horizon $h$, est de construire la politique jointe de façon incrémentale.

1. On construit l'ensemble des politiques jointes possibles de taille 1 (une politique de taille 1 est un nœud, ne contenant qu'une simple action).
2. On supprime les politiques jointes de taille 1 dominées (une politique est dominée si il existe une autre politique qui, pour tout état de départ, est meilleure).
3. On construit l'ensemble des politiques jointes de taille 2 (des arbres possédant 2 niveaux), en n'autorisant au niveau 1 que les politiques non supprimées à l'étape précédente.
4. On supprime les politiques de taille 2 dominées.
5. On répète le processus :
   - construction des politiques de taille $i$ (en se basant sur les politiques de taille $(i-1)$),
   - suppression des politiques de taille $i$ dominées.
6. On arrête lorsque l'on a construit l'ensemble des politiques de taille $h$.
7. Dans la politique jointe optimale $\pi$, chaque $\pi_i$ fait nécessairement partie des politiques à horizon $h$ calculées pour l'agent $i$ : on diminue donc considérablement l'espace de recherche.

Cette approche permet donc de réduire le nombre de politiques jointes à tester. Cependant, dans le pire cas (si aucune politique jointe n'est jamais dominée), on testera exactement le

même nombre de politiques que par l'approche naïve, d'où une complexité au pire cas identique. Plusieurs optimisations ont depuis été proposées, notamment en se basant sur la probabilité de recevoir une observation plutôt qu'une autre : nous ne rentrerons pas ici dans ces détails.

#### Programmation par heuristique

Une seconde famille d'algorithmes a été proposée permettant d'éviter l'énumération de l'ensemble des politiques jointes possibles : il s'agit des approches par **recherche heuristique basée A***. Ici, on va développer l'arbre de haut en bas. L'idée est de se baser sur une heuristique pour guider la recherche, en évitant certains sous-espaces de solutions potentielles.

1. On choisit une heuristique de résolution du problème pour approximer la fonction de valeurs pour toute action (remarque : l'heuristique doit être admissible, cela signifie qu'elle doit calculer une fonction de valeurs optimiste, donc supérieure ou égale aux vraies valeurs).
2. On teste les actions possibles au sommet de l'arbre (niveau 0) : afin de déterminer la politique optimale, on se base sur les valeurs heuristiques pour les niveaux 1 et suivants :
   - pour chaque action possible au niveau 0, on teste les actions au niveau 1, là encore en se basant sur les valeurs heuristiques pour les niveaux 2 et suivants,
   - de même, pour chaque action au niveau 1, on teste les actions au niveau 2, etc.
3. Lorsque l'on teste les actions à un niveau donné, on garde en mémoire la valeur de la meilleure action et on ne développe pas les actions dont la valeur heuristique est inférieure.
4. Comme l'heuristique est optimiste, une action ignorée est nécessairement moins bonne, même en considérant la vraie valeur.
5. Une fois toutes les combinaisons possibles testées, on renvoie la politique calculée : on aura alors trouvé une politique optimale, tout en ne vérifiant que certaines des politiques jointes possibles.

Cette approche permet donc, là encore, de limiter la quantité de politiques jointes à tester. Dans le pire cas, il arrive que l'heuristique soit tellement mauvaise qu'on teste toutes les politiques jointes possibles, d'où une complexité équivalente à l'approche naïve. Le choix de l'heuristique est donc important : plus une heuristique calcule des valeurs réalistes, plus on élimine de solutions. D'un autre côté, en général, plus une heuristique est bonne et plus elle est longue à calculer. L'algorithme MAA* [Szer *et al.*, 2005] est une des implémentations les plus couramment utilisées de cette approche, en utilisant le MMDP sous-jacent au problème comme fonction heuristique (c'est à dire en supposant une observabilité totale). Quoi qu'il en soit, ces méthodes de résolution ne permettent pas de traiter des problèmes à plus de 3 agents.

### 4.1.2 Résolution approchée à horizon fini

La résolution optimale d'un DEC-POMDP étant très complexe, plusieurs techniques de résolution approchée ont été proposées. L'idée ici est de calculer une politique quasi-optimale, en autorisant une erreur $\epsilon$ sur le calcul des fonctions de valeurs. Bien que de telles approches

restent de complexité NEXP-complet, elles permettent d'augmenter la taille des problèmes que l'on peut résoudre (notamment en termes d'horizon de résolution). Nous introduirons ici les trois grandes familles d'algorithmes pour la résolution approchée (JESP, MBDP et PBIP).

#### Approche de type « JESP »

Les méthodes de **recherche de politiques à base d'équilibre conjoint** (JESP [19]) ont été introduites [Chades *et al.*, 2002, Nair *et al.*, 2003] afin de calculer une solution localement optimale pour un DEC-POMDP donné. L'idée est d'itérer sur les politiques individuelles.

1. On choisit une politique initiale (arbitraire) pour chaque agent.
2. On cherche la politique optimale pour l'agent 1 (on teste toutes les politiques possibles pour cet agent, sans modifier les politiques des autres agents), jusqu'à maximiser l'espérance jointe de gains.
3. On cherche ensuite la politique optimale pour l'agent 2 (sans modifier la politique choisie précédemment pour l'agent 1 ni les politiques des autres agents).
4. On continue ainsi pour les agents 3, puis 4, puis ..., puis $I$.
5. On recommence à l'étape 2.
6. On arrête lorsque plus aucune politique n'évolue (il s'agit là, en théorie des jeux, d'un équilibre de Nash).

Cette approche permet donc de calculer une politique jointe localement optimale. Rien ne garantit cependant qu'il n'existe pas une autre politique jointe localement optimale, de meilleure qualité : il s'agit donc d'une méthode de résolution approchée. Plusieurs algorithmes implémentent cette méthode, un des plus efficace étant DP-JESP, une version orientée programmation dynamique de JESP.

1. On applique la méthode JESP (chercher les politiques des agents un à un).
2. Au lieu de tester toutes les politiques possibles pour l'agent en cours, on construit sa politique optimale via la méthode de programmation dynamique vue précédemment.
3. Afin de calculer la valeur d'une politique jointe, on se base sur les croyances atteignables (cela signifie qu'on ignore les historiques joints impossibles).

L'algorithme DP-JESP permet alors de calculer une politique à horizon beaucoup plus grand que via une approche naïve.

#### Approche de type « MBDP »

Une autre difficulté se pose lorsque l'on résout de tels problèmes : ils sont très coûteux en mémoire. Des algorithmes « à mémoire bornée » ont été proposés, permettant de passer outre cette difficulté : l'idée est d'utiliser une approche par programmation dynamique, mais de modifier la phase d'élagage des politiques dominées. A chaque étape, plutôt que de garder l'ensemble

19. JESP : de l'anglais « Joint Equilibrium-based Search for Policies ».

des politiques non-dominées, on ne va conserver qu'un sous-ensemble de ces politiques. Pour cela, on va calculer l'ensemble des historiques joints possibles, et garder au moins une politique non-dominée (de préférence, la meilleure) pour chacun d'eux. L'algorithme de **programmation dynamique à mémoire bornée** (ou MBDP [20]), a été introduit [Seuken et Zilberstein, 2007] comme une implémentation de cette approche et a depuis été plusieurs fois étendu (IMBDP, MBDP-OC, ...).

#### Approche de type « PBIP »

Pour finir, l'algorithme de **résolution par induction rétrograde à base de croyances et à mémoire limitée** (ou PBIP [21]) a été introduit [Dibangoye *et al.*, 2009] pour améliorer l'efficacité des approches par programmation dynamique. Cet algorithme apporte un gain en termes de nombre de politiques à tester, d'espace mémoire nécessaire et d'efficacité lors de l'élagage des politiques dominées. L'idée est la suivante :

- on utilise une heuristique (afin d'estimer la valeur de chaque action pour un niveau donné dans l'arbre de politique), pour n'explorer qu'un sous-ensemble des politiques jointes possibles durant la phase de génération, puis on élimine les politiques dominées comme dans l'approche « normale ». L'auteur propose ainsi une heuristique peu coûteuse en temps de calcul, mais tout de même capable de diminuer la quantité d'actions à tester.
- L'algorithme permet ensuite de limiter l'espace mémoire nécessaire : pour un niveau donné, on commence par classer les actions candidates selon la valeur heuristique et on teste la première candidate. Si l'heuristique était juste, on construit du premier coup la politique optimale, sinon (lorsque l'heuristique a surestimé la valeur de l'action) on devra tester la suivante et ainsi de suite. Cela permet de limiter le nombre de politiques à explorer, et donc la quantité de données à garder en mémoire.
- L'élagage est également plus efficace : une politique $A$ de valeur heuristique inférieure à la valeur réelle d'une politique $B$ étant nécessairement dominée, elle sera élaguée sans que l'on ait à calculer sa fonction de valeurs réelle.

Cette approche a montré de très bons résultats, avec des temps de calcul jusqu'à 800 fois plus rapides que les meilleurs algorithmes existants. Elle a par la suite été étendue [Amato *et al.*, 2009], et adaptée aux jeux stochastiques [Oliehoek *et al.*, 2010]. Il demeure malgré tout impossible de passer à l'échelle, notamment en traitant des problèmes à plus de 3 agents.

### 4.1.3 Résolution à horizon infini

La résolution d'un DEC-POMDP à horizon infini est un problème indécidable. On se contentera donc de calculer des solutions quasi-optimales, à une borne $\epsilon$ prêt. Pour cela, on choisit de représenter la politique d'un agent par un automate déterministe à états finis (et la politique jointe par un ensemble d'automates). La politique optimale d'un DEC-POMDP à horizon infini

20. MBDP : de l'anglais « Memory Bounded Dynamic Programming ».
21. PBIP : de l'anglais « Point-Based Incremental Pruning ».

nécessiterai l'usage d'un automate de taille infinie, ce qui est impossible. On devra donc choisir un automate suffisamment grand pour permettre la représentation d'une politique de bonne qualité, mais suffisamment petit pour tenir en mémoire. Quoi qu'il en soit, les approches présentées ici ne permettent de résoudre que des problèmes de petite taille, notamment en nombre d'agents.

#### Approche MB-DEC-PI

L'approche par **itération de politiques décentralisées et bornées en mémoire** (ou MB-DEC-PI [22]) a été introduite [Bernstein *et al.*, 2005] afin de calculer une politique jointe minimisant la taille des automates de chaque agent. L'idée est d'alterner deux phases de calcul :

- une phase d'expansion, où on ajoute des noeuds aux automates de chaque agent,
- une phase de compression, où on évalue la qualité des politiques jointes afin de supprimer les nœuds inutiles dans chaque automate individuel.

Cet algorithme permet alors de calculer une politique jointe optimale à une borne d'erreur prêt. Cette approche pose cependant un certain nombre de difficultés. En effet, les politiques calculées ont la particularité d'être représentées par des automates stochastiques (en un état de l'automate, l'agent ne dispose pas d'une action à effectuer mais plutôt d'une distribution de probabilités sur un ensemble d'actions). Afin de coordonner les actions des agents malgré cette incertitude, les auteurs supposent alors l'existence d'une source d'aléa communes aux agents. Ainsi, on peut imaginer un problème où deux agents doivent choisir entre deux actions $A$ ou $B$. L'aléa commun garanti ici que les agents choisiront toujours en même temps $AA$ ou $BB$.

#### Recherche heuristique

L'approche par **recherche heuristique** a été introduite [Szer et Charpillet, 2005] afin de permettre une résolution à horizon infini optimale pour une taille d'automate donnée. Pour cela :

- on choisi une taille d'automate (c'est à dire un nombre de nœuds) : plus on met de nœuds, plus on aura une politique jointe de bonne qualité mais plus le temps de calcul sera long. On va alors calculer une politique jointe optimale pour cette dimension.
- On note $n_i$ le nœud sur lequel l'agent $i$ se trouve et $n$ le nœud joint,
- initialement, un agent peut appliquer n'importe quelle action en chaque nœud, et chaque observation reçue en un nœud $n_i$ peut mener à l'ensemble des nœuds $n'_i$. On désigne alors par $\pi(n_i)$ l'ensemble des actions possibles pour l'agent $i$ dans un nœud $n_i$ et par $\pi(n_i,o_i)$ l'ensemble des nœuds $n'_i$ vers lesquels l'agent $i$ peut transiter après observer $o_i$ dans $n_i$.
- On crée un MMDP heuristique dont les états sont l'ensemble des couples $(s,n)$ avec $n$ un nœud joint et $s$ un état du DEC-POMDP,
- la fonction de valeur de ce MMDP sera alors la fonction $V^{pi}(s,n) = \max_{a\in\pi(n)} R(s,a) + \gamma \left[\sum_{s',o} P(s',o|s,a).\max_{n'\in\pi(n,o)} V^{\pi}(s',n')\right]$ avec $P(s',o|s,a)$ la probabilité de passer dans l'état $s'$ et de recevoir l'observation $o$ lorsque l'on exécute l'action $a$ dans l'état $s$.

22. MB-DEC-PI : de l'anglais « Memory-Bounded Decentralized Policy-Iteration ».

– En résolvant ce MMDP, on trouve une action idéale par nœud et un nœud cible idéal par observation. On dispose alors d'une politique jointe optimale.

Cette approche, particulièrement intéressante, permet alors de résoudre efficacement un DEC-POMDP. Il est cependant difficile de choisir une taille arbitraire pour les automates de politique (il n'existe aucune façon de déterminer une taille minimum nécessaire).

## 4.2 Approches basées indépendance

Les approches « standards » présentées dans la section précédente offrent une base théorique intéressante pour la résolution de DEC-POMDP, mais sont toutes incapables de passer à l'échelle (notamment en termes de nombre d'agents). Pour cette raison, de nouvelles approches ont été proposées permettant de traiter les modèles tirant parti des indépendances entre agents.

### 4.2.1 Résolution des DEC-POMDPs factorisés

Nous avons présenté dans le chapitre précédent le cas particuliers des DEC-POMDPs factorisés. Ce modèle est un cas à part, en raison du type de problèmes adressés. Il faut, en effet, non seulement une structure statique d'interaction, mais également une représentation factorisable des états, transitions, observations et récompenses. Lorsque l'ensemble de ces hypothèses est vérifié, on peut calculer une politique jointe. Cette approche étant relativement éloignée du domaine que l'on traite, nous passerons très rapidement sur les algorithmes de résolution qui y sont dédiés. L'idée est, pour résoudre ce modèle, d'appliquer une méthode de recherche heuristique.

1. Pour chaque pas de temps, on calcule un graphe d'influence, permettant de déterminer quels agents sont en interaction.
2. On décompose la fonction de valeurs de la politique jointe selon ce graphe d'influence, en un ensemble de composantes mutuellement indépendantes.
3. On utilise ces composantes pour générer un jeu bayésien collaboratif par pas de temps.
4. On calcule la politique jointe optimale grâce un algorithme dérivé de MAA* (voir section 4.1.1), dans lequel on choisi l'action jointe d'un niveau donné en résolvant le jeux bayésien collaboratif associé .

Cette approche a permit de calculer des politiques optimales pour des problèmes à plus de 2 agents, ce qui est impossible via des méthodes « standards ». Elle demeure néanmoins insuffisante pour des problèmes de dimension réelle.

### 4.2.2 Résolution des ND-POMDPs

Nous avons présenté, précédemment, le modèle ND-POMDP pour la description de problèmes impliquant une indépendance entre agents, sauf en ce qui concerne les récompenses (où il y a dépendance par groupes). On choisi ici de représenter la politique d'un agent par un automate déterministe à états finis. [Marecki *et al.*, 2008] ont proposé l'algorithme FANS pour la résolution

de ce modèle, basé sur l'idée que les agents ne sont pas égaux en termes d'importance (un agent peut avoir à traiter un problème plus complexe qu'un autre), et donc qu'ils n'ont pas tous besoin d'un automate avec le même nombre d'états. Pour chaque agent, on décide du nombre de nœuds dans l'automate en fonction de son importance. Pour cela, on applique trois étapes en boucle.

1. Sélectionner, par heuristique, les agents dont il faut étendre l'automate.
2. Ajouter des nœuds aux automates de ces agents.
3. Calculer la nouvelle politique jointe optimale.

On répète ces trois étapes, jusqu'à ce que l'évolution de la fonction de valeurs entre deux passes successives soit négligeable. Plusieurs heuristiques sont proposées pour étendre les automates, de complexité et de qualité croissante. On peut citer par exemple :

- égalitaire : ajouter un nœud à tous les agents,
- gloutonne : on ordonne les agents par degré de connectivité croissante (c'est à dire selon le nombre de voisins avec qui ils sont en interaction), puis on place un pointeur sur le premier agent (de degré le moins élevé). À chaque passe, on augmente les nœuds de l'agent portant le pointeur ainsi que des agents suivants selon leur ordonnancement, puis on déplace le pointeur vers l'agent suivant,
- « node » : on calcule, pour chaque agent, une valeur heuristique (via un MDP) en supposant que les politiques des autres agents ne changent pas, puis on augmente les nœuds de l'agent ayant la meilleure heuristique.

Une fois que l'on a choisi une heuristique, il faut pouvoir calculer la nouvelle politique jointe optimale, selon les tailles d'automates décidées précédemment. On utilise pour cela un algorithme de type « branch and bound ».

1. On commence par calculer un arbre de dépendances : les nœuds sont des agents et les branches sont telles que si un agent $i$ est en interaction avec un agent $j$, alors $i$ sera parent de $j$ dans l'arbre (ou inversement).
2. On teste toutes les politiques possibles pour l'agent 1 (l'ordre des agents est donné par l'arbre de dépendances), ces politiques étant triées via une fonction de valeurs heuristique.
3. Pour chaque politique de l'agent 1, on teste toutes les politiques de l'agent 2.
4. On continue jusqu'au dernier agent (pour chaque politique de l'agent 2, on teste les politiques de l'agent 3, etc.) et on calcule alors la valeur réelle de la politique jointe obtenue.
5. On peut, en se basant sur cette valeur réelle, supprimer toutes les politiques jointes de valeur heuristique inférieure.

En général, on utilise un MMDP comme heuristique pour estimer la valeur d'une politique (en fixant les actions des agents dont la politique est déjà décidée). Cette approche offre de très bons résultats, aussi bien en termes de dimension du problème (nombre d'agents ou d'états) que de qualité des politiques calculées. Elle ne s'applique cependant qu'aux problèmes de type ND-POMDP, donc basés sur un modèle d'interaction statique.

## 4.3 Approches basées interactions

Les approches basées indépendance sont particulièrement efficaces pour la résolution de problèmes à plus de 2 agents. Malheureusement, ce type de méthode ne permet pas de traiter des problèmes plus complexe, dans lesquels les indépendances évoluent au cours du temps. Pour cela, des algorithmes ont été introduits permettant de traiter les modèles basés interactions [Mouaddib *et al.*, 2007, Boussard *et al.*, 2008]. Nous traiterons ici la résolution des deux modèles les plus récents dans ce domaine : les jeux markoviens orientés interaction et les POMDPs distribués avec points de coordination.

### 4.3.1 Résolution des DPCL

Nous avons présenté, précédemment, le modèle des POMDPs distribués avec points de coordination (DPCL). Pour rappel, ce modèle permet de représenter des problèmes dans lesquels la coordination se fait uniquement par le biais d'un ensemble de tâches à réaliser. La réalisation d'une tâche par un agent peut alors entraîner une fonction de transition ou de récompense jointe (STCL), ou avoir un impact sur les autres agents dans le futur (FTCL). L'idée, pour traiter ce modèle, est de calculer l'allocation optimale de tâches parmi les agents, puis de calculer pour chaque agent un POMDP optimal selon les tâches qu'il doit réaliser.

1. On calcule, pour chaque allocation de tâches possible, une fonction de valeurs heuristique et optimiste (ici, on choisi un MMDP).
2. On évalue ces allocations une par une, en commençant par celle d'heuristique la plus élevée.
3. Lorsque l'on évalue une allocation donnée, on calcule sa vrai fonction de valeurs.
4. On garde de coté l'allocation de valeur maximale : lorsque l'on calcule la valeur d'une allocation est qu'elle est inférieure à la maximale, on supprime cette allocation.
5. On supprime également, à chaque fois, toutes les allocations dont l'heuristique est inférieure à la valeur maximale calculée (car l'heuristique est optimiste).
6. On s'arrête lorsqu'il ne reste qu'une seule allocation : c'est la meilleure.

Cette méthode permet de déterminer l'allocation de tâches optimale. Il faut, pour cela, pouvoir évaluer une allocation donnée. Plutôt que de calculer une politique jointe optimale pour l'allocation, on calcule un ensemble de politiques individuelles localement optimales.

1. On calcule, pour chaque agent, une politique individuelle optimale.
2. On identifie les interactions (FTCL et STCL) impliquées par ces politiques.
3. On modifie les fonctions de transition et récompense des agents en interaction.
4. On recommence l'étape 1 avec l'ensemble des agents dont les fonctions de récompense ou de transition ont changé.
5. On s'arrête quand plus aucune politique ne change.

Il faut de plus, pour exécuter cette procédure, pouvoir identifier les interactions et modifier les fonctions de transition/récompense associées. Pour identifier les interactions :

- on a, pour chaque agent, une fonction de valeurs $V$ (selon sa politique),
- pour chaque interaction, on calcule sa probabilité d'arriver (au vu des politiques actuelles),
- pour chaque agent, on calcule $V'$ la fonction de valeurs qui intègre les probabilités d'interaction et qui utilise les fonctions de récompense et de transition jointes,
- on pose $\Delta = V' - V$ : si $\Delta > 0$, l'interaction est positive, sinon elle est négative.

Il faut maintenant modifier les fonctions de transition et de récompense selon les interactions identifiées. On veut s'assurer que les interactions positives arrivent, mais on veut éviter les négatives. Pour cela :

- on augmente la récompense des actions qui provoquent des interactions positives, afin de pousser les agents à choisir ces actions,
- on diminue la récompense, pour un seul des agents, des actions provoquant des interactions négatives (prenons l'exemple d'un couloir où les agents peuvent se cogner : si on diminue la récompense d'un des agents, il va changer de chemin),
- pour les interactions négatives, on ne modifie pas la fonction de transition. En effet, puisque l'on va éviter ces interactions, les transitions ne vont pas changer,
- pour les STCL positives, la probabilité de transition individuelle est une somme sur les transitions jointes vérifiant la réalisation de cette interaction,
- pour les FTCL positives, on ajoute à la transition individuelle une probabilité de réaliser la tâche concernée par cette interaction.

On peut donc, via cette approche, résoudre efficacement des problèmes de grande taille (notamment en termes de nombre d'agents). Cette méthode demeure toutefois limitée du point de vue de son applicabilité.

### 4.3.2 Résolution des IDMG

Nous avons précédemment décrit le modèle IDMG, qui permet la représentation de problèmes multiagents via un ensemble $\{M_1,M_2,\ldots,M_I,\{M_m\}\}$ où $M_i$ décrit le problème individuel et complètement observable d'un agent et où $\{M_m\}$ décrit un ensemble de problèmes d'interaction. On a de plus précisé que ce modèle supposait une observabilité locale complète ainsi qu'une communication gratuite et illimitée entre les agents présents dans la même zone d'interaction (ce qui permet de considérer chaque $M_m$ comme un problème complètement observable). On calcule alors un ensemble de politiques individuelles pour le problème global de la façon suivante (on suppose ici que l'on traite un problème $\{M_1,M_2,M_m\}$ à deux agents et une seule zone d'interaction) :

1. on calcule $Q_i^*(s_i,a_i) = \sum_{s'_i \in S_i} T_i(s_i,a_i,s'_i)[R_i(s_i,a_i,s'_i) + \gamma \max_{a'_i \in A_i} Q_i^*(s'_i,a'_i)]$ la fonction d'utilité individuelle pour chaque agent $i$ avec $M_i = \langle S_i,A_i,T_i,R_i \rangle$,
2. on calcule de même $Q_m^*(s,a) = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma \max_{a' \in A} Q_m^*(s',a')]$ la fonction d'utilité jointe pour la zone d'interaction $m$ avec $M_m = \langle S, A, T, R \rangle$ telle que $S = S_1 \times S_2$ et $A = A_1 \times A_2$ (par contre, $T$ et $R$ ne sont pas nécessairement décomposables),

3. on pose ensuite $\bar{Q}_i^*(s,a) = Q_i^*(s_i,a_i) + Q_m^*(s,a)$ la fonction d'utilité croisée pour l'agent $i$ (utilité intégrant la composante individuelle et la composante d'interaction),

4. on peut alors définir, pour tout état joint $s$, un jeu matriciel (au sens de la théorie des jeux) à deux joueurs, dont les actions respectives sont $A_1$ et $A_2$ et où la fonction d'utilité jointe est donnée par $q(a) = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma Nash(\bar{Q}_1^*,\bar{Q}_2^*)]$ avec $Nash()$ l'opérateur cherchant un équilibre de Nash entre les deux joueurs.

Ainsi, en résolvant l'ensemble des jeux matriciels possibles, on calcule la politique jointe. En dehors des zones d'interaction, il n'y a pas de communication et on suppose un comportement indépendant des agents (on utilisant les fonctions d'utilité individuelles). Cette approche a permis, grâce à sa structure d'interactions dynamique, de calculer des politiques quasi-optimales sur un ensemble de benchmarks classiques dans le domaine. Elle nécessite cependant deux hypothèses fortes (observabilité locale complète, communication gratuite et illimitée), rarement vérifiées. Cette approche implique de plus d'ignorer les effets à long terme des interactions, ce qui peut être problématique sur certains problèmes.

### 4.3.3 Résolution de DEC-SIMDP

Le modèle DEC-SIMDP est certes proche du modèle IDMG, mais sa résolution diffère sensiblement. L'idée ici est de résoudre le problème d'un agent donné, à partir de son MDP et de l'ensemble des MMDPs associés aux zones d'interaction. Pour cela, on calcule une fonction de $Q$-valeurs en appliquant une équation de Bellman [23] modifiée, dans laquelle :

- on raisonne sur les états joints,
- la transition jointe $T$ est obtenue par un produit de (1) la fonction de transition individuelle pour les agents n'étant pas en interaction, et (2) les fonctions tirées des MMDPs pour les agents évoluant dans une même zone d'interaction,
- la récompense jointe $R$ est obtenue par une somme des fonctions de récompense individuelles de chaque agent, et des fonctions de récompense jointes associées à chaque MMDP (c'est-à-dire à chaque zone d'interaction),
- on raisonne sur les actions individuelles, en supposant que les autres agents choisiront l'action maximisant cette $Q$-valeur.

On calcule donc cette fonction de $Q$-valeurs en supposant le problème complètement observable. Pourtant, à l'exécution, l'observabilité est partielle : l'agent observe totalement son état individuel et l'état de ses voisins, mais n'observe rien concernant les autres agents, hors voisinage. Pour palier à ce problème, on maintiendra un état de croyance $b$ sur l'état des autres agents et on exécutera l'action donnée par :

$$\pi(b) = \underset{a}{\text{argmax}} \sum_s b(s).Q(s,a)$$

23. Pour rappel : $Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s').\max_{a'} Q(s',a')$.

On applique donc le concept des Q-MDPs. Afin de maintenir l'état de croyance $b$, il faut connaître les actions exécutées par les autres agents. Puisque l'on ne connaît par ces actions, on utilise une heuristique pour les prévoir :

- soit on suppose que chaque agent exécute son MDP individuel, en ignorant l'existence des autres agents,
- soit on suppose que le groupe exécute le MMDP sous-jacent au problème traité.

Ces deux approches ne sont que des heuristiques, puisqu'en réalité les agents prennent en compte l'existence des voisins, mais n'ont pas une observabilité suffisante pour exécuter le MMDP sous-jacent. Elles ont toutefois permis, durant les tests réalisés, d'obtenir des politiques de bonne qualité. Cette approche est plus performante que celle utilisant le modèle IDMG, puisque l'on « n'oublie pas » les voisins une fois ceux-ci sortis de la zone d'interaction. L'approche conserve malgré tout certains défauts (l'hypothèse d'observabilité locale complète par exemple). Le défaut le plus contraignant vient de la formalisation même du problème : il est nécessaire d'identifier préalablement les zones d'interaction. Ainsi, si le problème est peu contraint (un ensemble d'agents évoluant dans un espace ouvert par exemple, étant amenés à tous se croiser à un moment donné, et ce n'importe où), on arrive à n'avoir plus qu'une seule zone d'interaction, impliquant tous les agents et tous les états. Ce modèle revient alors à remplacer le problème traité par son MMDP sous-jacent, ce qui est une hypothèse peu réaliste.

## Conclusion

Nous avons présenté, dans ce chapitre, un ensemble d'algorithmes de résolution pour calculer une politique jointe selon un DEC-POMDP, ou selon un des modèles dérivés. Nous avons notamment vu que le modèle « classique » DEC-POMDP ne pouvait être résolu que sur certaines instances particulièrement simples, notamment en termes de nombre d'agents (pas plus de 3 agents) ou encore de nombre d'états ou d'observations. Pour cette raison, des algorithmes ont été proposés pour la résolution des modèles dérivés, exploitant les indépendances entre agents. Nous avons en particulier vu le cas des ND-POMDP, qui ont permis la résolution de problèmes de grande taille. Ces modèles sont toutefois peu expressifs, puisqu'ils ne permettent de traiter que des problèmes à interactions statiques.

Nous avons finalement présenté le domaine, très récent, de la résolution basée interaction. Ces approches, qui permettent de traiter des problèmes où la structure d'interaction évolue avec le temps, semblent les plus prometteuses pour la résolution de problèmes réels. Nous avons ainsi montré comment traiter les deux type de modèles dont nous avions parlé dans le chapitre précédent, à savoir les DPCL et IDMG. Là encore cependant, utiliser ces modèles suppose un certain nombre d'hypothèses limitatives, qu'il s'agisse de la structure des interactions (DPCL) ou de besoins forts en communication et en observabilité (IDMG). Malgré tout, ce type d'approche (avec DEC-SIMDP notamment) ouvre des pistes encourageantes pour la résolution de problèmes réels. C'est donc sur ce type de raisonnement que se baseront nos travaux, présentés dans la suite de ce document.

# Conclusion de la partie I

Cette partie aura été l'occasion de poser le contexte dans lequel se place nos travaux. Nous avons donc introduit le problème de la planification multiagent, en environnement stochastique et partiellement observable. Nous avons notamment présenté le modèle DEC-POMDP, utilisé pour la représentation de ces problèmes, ainsi qu'un ensemble de modèles dérivés utilisables selon que certaines hypothèses soient vérifiées ou non. Nous avons finalement introduit les différents types d'algorithmes de résolution permettant de traiter l'ensemble de ces modèles. Il y a plusieurs points centraux, qu'il est important de retenir. Tout d'abord, il est absolument impossible de traiter des problèmes de taille réelle avec un DEC-POMDP « standard » (du moins, pas avec les moyens dont nous disposons actuellement). Ensuite, même si plusieurs méthodes approchées existent, toutes souffrent de problèmes d'applicabilité. Nous avons présenté un certain nombre de méthodes, ayant fourni de bons résultats sur plusieurs problèmes de test, mais il existe encore certains problèmes que l'on ne peut pas traiter via ces approches.

Nos travaux ont porté sur ce type de problèmes complexes, notamment grâce à notre collaboration avec les groupes industriels Thales TOSA et Dassault Aviation. Thales a posé le problème (que nous détaillerons par la suite) d'un groupe de véhicules autonomes devant se déplacer d'un point à un autre. Ces véhicules doivent alors adopter certaines formations, selon leur situation (type d'environnement, nombre de véhicules présents, existence d'un danger, etc.). Ce problème est complexe, pour plusieurs raisons. Tout d'abord, un convoi implique en général un nombre important d'agents, donc les méthodes classiques sont inapplicables. Ensuite, ces agents doivent adapter leur formation à la situation, les interactions sont donc amenées à évoluer dans le temps (et porteront aussi bien sur les récompenses et transitions que sur les observations). Pour finir, l'observabilité est partielle (de part les contraintes techniques dues aux capteurs embarqués) et la communication est à prohiber (les engins pouvant évoluer en territoire ennemi). Il s'agit donc bien d'un problème de type DEC-POMDP.

Pour ces raisons, les méthodes présentées dans cet état de l'art s'avèrent inapplicables à ce genre de problème. C'est sur ce constat que se basent nos travaux : nous proposons donc un nouveau modèle, pour la représentation de problèmes complexes (comme celui de Thales), mais capable de tirer parti de la structure inhérente à tout problème de décision lors de la phase de résolution. L'idée sera donc de baser la résolution du problème sur les interactions entre agents, comme le font les approches actuelles les plus prometteuses (IDMG, DPCL, DEC-SIMDP), sans pour autant introduire d'hypothèse restrictive. La partie suivante de ce document sera dédiée à

la définition de ce modèle, suivie d'une troisième partie présentant un ensemble d'algorithmes de résolution adaptés.

## Deuxième partie

# DyLIM : un modèle d'interactions locales et dynamiques

« *Pas trop d'isolement ; pas trop de relations ; le juste milieu, voilà la sagesse.* »
Confucius, expert en raisonnement basé interactions.

# Introduction à la partie II

Les problèmes de décision sous incertitude représentent une part importante de l'intelligence artificielle, permettant aux agents évoluant en environnement stochastique d'adopter un comportement rationnel. Il est impossible, dans ce type de problème, de calculer une série d'actions garantissant la réalisation d'un but final. On essaye plutôt de calculer une politique de comportement, permettant d'optimiser l'espérance de gain à long terme. Nous avons rappelé, dans la partie précédente, que le modèle DEC-POMDP permettait la représentation de tels problèmes pour le cas multiagent et partiellement observable.

Le modèle DEC-POMDP offre un cadre mathématique solide permettant la formalisation de plusieurs applications réelles, mais souffre en contrepartie d'une explosion combinatoire rapide. Celui-ci repose en effet sur l'hypothèse de dépendances totales, qui implique que l'ensemble des agents soient en interaction, et ceci de façon permanente (d'où l'explosion combinatoire). Nous avons cependant montré que cette hypothèse était irréaliste, les interactions étant en général locales (à un sous-groupe d'agents) et temporaires. Pour cette raison, plusieurs modèles que nous avons présenté relâchent cette hypothèse. Un nouveau problème se pose alors : comment, si on suppose que les interactions sont locales et temporaires, prévoir (ou détecter) leur occurrence ? Nous avons ainsi présenté deux types d'approches existantes. Les premières reposent sur des interactions statiques (on décide, à l'avance, qui interagit avec qui et de quelle façon) : ce type d'approche permet alors de casser la complexité pour passer à l'échelle en termes de nombre d'agents impliqués, mais souffre d'un champ applicatif réduit. Pour cette raison, une seconde approche a été explorée, reposant sur des interactions dynamiques (on les détecte au fur et à mesure de l'exécution). Deux techniques existent alors, pour détecter ces interactions : soit on décide à l'avance de « points de coordination » sur lesquels l'interaction peut se faire, soit on suppose une communication gratuite entre agents et une observabilité individuelle totale, afin de connaître l'état joint et donc les interactions en cours. Là encore cependant, le champ d'applications se réduit (relativement à une approche DEC-POMDP).

On constate donc le besoin d'un modèle de ce type, permettant de gérer des interactions dynamiques, mais moins restrictif que les modèles existants. Ainsi, le premier chapitre de cette partie (chapitre 5) expose les motivations justifiant nos travaux, via notamment un problème de décision concret, fourni par le groupe industriel Thales. Ce problème ne nécessite pas l'hypothèse de dépendances totales, mais implique des interactions évoluant en permanence (donc non-statiques), que l'on ne peut pas limiter à des points de coordination. L'observabilité est de plus

partielle, même sur l'état individuel de l'agent, celle-ci reposant sur des capteurs embarqués dans l'agent. La communication pour finir, est interdite entre agents, ceux-ci pouvant évoluer en milieu hostile. Nous ferons particulièrement attention à ne pas proposer une solution ad-hoc au problème de Thales, mais bien un modèle général, pour les problèmes de type DEC-POMDP.

Le chapitre suivant (chapitre 6) nous permettra de poser les bases de notre modèle. Puisque l'on se refuse à restreindre « l'applicabilité » de celui-ci, il faut travailler sur les autres aspects, pour maintenir possible le passage à l'échelle. Ainsi, nous montrerons comment on exploite l'indépendance entre agents pour limiter, autant que possible, l'explosion combinatoire. Nous poserons la notion de relation entre agents, afin de représenter ces interactions. Nous montrerons alors comment raisonner directement sur ces relations, pour transformer le problème multiagent en un ensemble de problèmes monoagents indépendants, sans trop perdre en qualité. Nous proposerons ainsi de calculer des politiques de « bonne qualité », au lieu de politiques optimales.

Pour finir, le dernier chapitre (chapitre 7) fournira une présentation formelle du modèle DyLIM (Dynamic Local Interaction Model). Nous commencerons donc par exposer le modèle en lui-même, et la façon dont on peut l'utiliser pour décrire un problème de décision. Nous montrerons ensuite comment formaliser l'ensemble des interactions possibles, afin de permettre un raisonnement basé sur celles-ci. Nous définirons alors deux grandes catégories d'interactions : exactes, ou approchées. Nous montrerons finalement comment ce modèle peut être utilisé pour réduire au maximum la complexité combinatoire du problème à traiter. Nous disposerons ainsi d'un modèle exploitant l'indépendance entre agents, non-restrictif en termes de représentativité, mais permettant uniquement le calcul de politiques sous-optimales (localement optimales, du point de vue de l'agent). La troisième partie de ce document (traitant des algorithmes) nous permettra de vérifier que l'on génère des comportements de groupe certes sous-optimaux, mais malgré tout de bonne qualité.

# Chapitre 5

# Motivations

**Sommaire**



Nos travaux ont été menés en collaboration avec les groupes THALES Optronics S.A. et Dassault Aviation, qui ont fourni un cadre applicatif industriel. Cette collaboration a notamment permis à Thales de nous fournir un problème « test » de déplacement en convoi, que nous présenterons ici. Ce problème nous a permis d'identifier un ensemble de difficultés algorithmiques, ne pouvant être traitées via les méthodes existantes. Ainsi, dans ce chapitre, nous décrirons le problème et les difficultés qu'il soulève, puis nous verrons en quoi les modèles existants sont inadaptés à sa résolution. Cela nous permettra de spécifier les caractéristiques que devrait respecter un modèle dérivé des DEC-POMDPs, afin de casser la complexité combinatoire sans pour autant se restreindre en termes de représentativité.

## 5.1 Point de départ : le problème de Thales

Le problème fourni par Thales nous servira de support pour illustrer nos propos, tout au long de ce document. Il soulève de plus un certain nombre de difficultés techniques, que nous détaillerons ici, et qui nous permettrons de mettre en avant les lacunes des modèles existants.

### 5.1.1 Problème à résoudre

Prenons un ensemble d'engins autonomes, ayant pour objectif de se déplacer d'un point à un autre au sein d'une zone de conflit. Ces engins sont, typiquement, des véhicules sans pilote (il peut s'agir de transporter des marchandises, ou d'explorer une zone par exemple). Nous supposerons, dans un premier temps, que ces engins sont homogènes (nous donnerons, plus loin dans ce document, quelques éléments relatifs à l'usage d'agents hétérogènes, mais ce cas ne sera pas explicitement traité). L'objectif est alors de calculer une politique jointe, leur permettant de naviguer d'un point à un autre, le tout en maintenant une **structure de convoi** donnée. On désigne par ce terme le positionnement (géographique) que les agents doivent adopter, les uns relativement aux autres. On peut par exemple imaginer un convoi simple, dans lequel les agents évoluent dispersés dans leur environnement (figure 5.1).

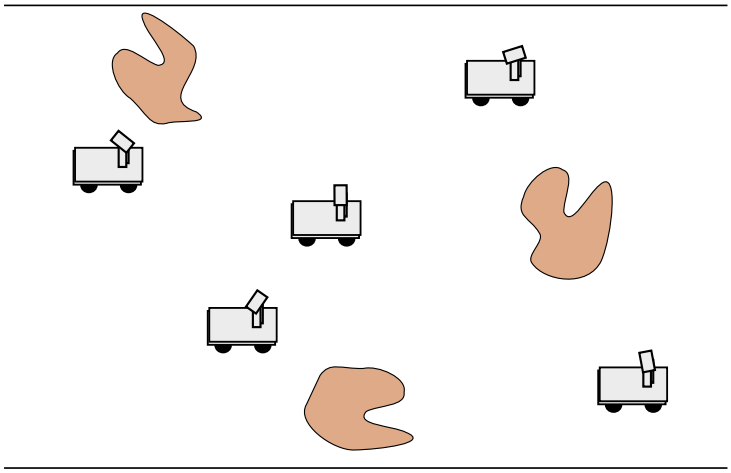

FIGURE 5.1 – Exemple de convoi « dispersé »

Un convoi de ce type sera par exemple utile lorsque les agents cherchent à analyser leur environnement (par exemple dans le cas d'une patrouille de surveillance) : ils se déplacent vers leur objectif, mais veulent acquérir un maximum d'informations sur l'environnement durant leur déplacement. Étant ainsi dispersés, ils vont couvrir une plus large zone, et donc acquérir plus d'informations. On pourra alors spécifier plus précisément la forme du convoi : on peut par exemple imaginer que les agents doivent s'éparpiller, mais pas trop, afin de rester à distance de contact visuel. Ce type de convoi n'est toutefois qu'un exemple parmi d'autres : on pourrait également imaginer un convoi « en ligne », afin de traverser une zone dangereuse par exemple (figure 5.2). Ici, les agents se déplaceraient les uns derrière les autres, afin de minimiser les risques. Imaginons que la zone soit très rocailleuse, et que les agents aient du mal à observer ces rochers. Ce type de convoi garantira alors que seul l'agent de tête sera en danger, les autres empruntant le même chemin que lui. On peut ainsi imaginer toutes sortes de convoi, en jouant

à la fois sur la structure (ligne, dispersion, ou formations plus complexes) et sur les distances entre agents.

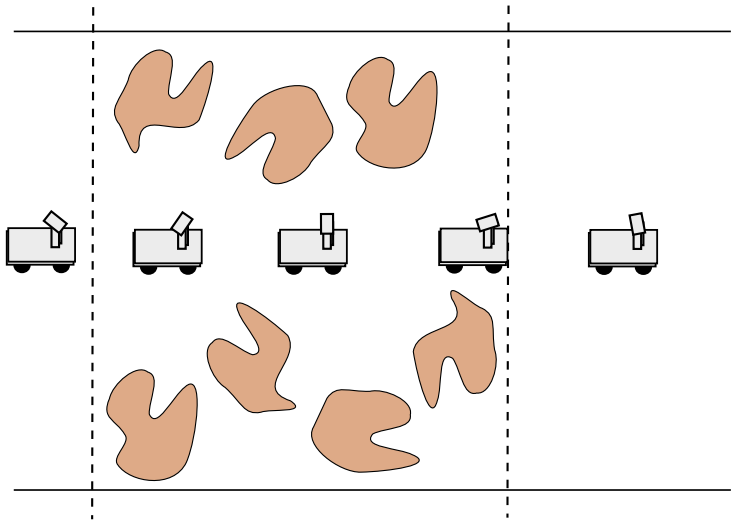

FIGURE 5.2 – Exemple de convoi « en ligne »

Les agents devront donc être capables de former ce type de convoi, afin d'optimiser leurs déplacements. Idéalement, les agents devront adapter leur type de convoi à la situation actuelle. On peut par exemple commencer par un convoi dispersé, puis détecter une zone dangereuse et passer à un convoi en ligne, avant de revenir à la formation initiale. Les agents étant autonomes, ils devront prendre par eux-même la décision de former un convoi plutôt qu'un autre, au vu de leurs connaissances locales. On peut imaginer une phase de planification initiale centralisée, afin d'embarquer au sein de chaque agent sa politique individuelle, mais l'exécution de cette politique devra être complètement décentralisée. Les agents devront également, tout en maintenant leur formation, s'assurer d'atteindre l'objectif du convoi et éviter d'entrer en collision avec d'autres agents. Ces agents embarqueront un ensemble de capteurs et d'effecteurs, leur permettant d'observer leur environnement et de se déplacer, mais ces éléments pourront être imparfaits (roue qui glisse, observation bruitée, etc.). Ils devront donc être à même de gérer cette incertitude durant l'exécution de leur politique. Pour finir, on interdira l'usage de toute communication (les agents pouvant évoluer en terrain hostile, où les communications pourraient être localisées, interceptées, voir même falsifiées). Les agents devront donc acquérir par eux-même des connaissances sur leur environnement et sur les autres agents.

### 5.1.2 Spécificités techniques

Ce problème soulève plusieurs difficultés intéressantes à traiter. Tout d'abord, on voit clairement qu'il s'agit d'un problème de type DEC-POMDP. On a en effet un problème multiagent, en environnement partiellement observable, impliquant des actions incertaines menant à des états intéressants (atteindre le but, adopter la formation voulue) ou à éviter (créer des collisions),

ainsi qu'une planification centralisée, mais une exécution décentralisée. Ce problème implique cependant d'autres hypothèses, pouvant poser problème si on utilise les modèles existants :

- **absence de communication**, les agents étant amenés à évoluer en environnement « hostile » (on a choisi, plutôt que de restreindre l'usage de la communication à certaines zones, de l'interdire complètement),
- **observabilité partielle**, même en ce qui concerne l'état individuel de l'agent (typiquement, on peut supposer que l'agent ne pourra pas toujours connaître sa position exacte),
- **plus de 3 agents**, puisqu'un convoi peut, à priori, impliquer plusieurs dizaines d'agents,
- **dépendances totales** entre les agents, c'est à dire aussi bien sur les récompenses (par exemple, maintenir la formation) et les transitions (deux agents qui entrent en collision vont être stoppés) que sur les observations (il faudra non seulement observer l'état individuel, mais aussi observer les autres agents pour connaître leurs états),
- **interactions ponctuelles**, les agents étant en interaction (c-à-d ayant des dépendances entre eux) uniquement lorsqu'ils sont proches les uns des autres (un agent à l'arrière du convoi par exemple, n'a pas besoin de prendre en compte les agents au début du convoi). On a donc des interactions locales uniquement, et n'impliquant que certains agents,
- **interactions complexes**, ne pouvant pas être représentées par une « simple » allocation de tâches (les interactions évoluant en permanence, il faudrait une infinité de tâches pour les représenter).

Ce type de problème est intéressant à traiter, puisqu'il correspond à une situation réelle, finalement assez différente des problèmes « jouets » traités par les modèles standards. Ce problème diffère également des benchmarks conçus pour être résolus via les modèles dérivés, ceux-ci étant en général artificiellement construits de façon à vérifier certaines hypothèses. Nous avons publié une première étude de cet exemple [Canu et Mouaddib, 2011a], dans le cas où le voisinage peut être complètement observé par l'agent, mais il est nettement plus difficile de modéliser un tel problème lorsque l'on n'a pas cette observabilité totale.

## 5.2 Conséquences sur la modélisation

Nous avons relevé, dans la section précédente, plusieurs hypothèses impliquées par ce problème de convoi. Nous allons maintenant voir en quoi ces hypothèses posent problème, lorsque l'on tente d'utiliser les modèles existants. Nous en déduirons alors un ensemble de besoins auxquels devrait répondre un modèle non-restrictif basé interactions.

### 5.2.1 Des modèles existants insatisfaisants

Le problème de Thales ne peut être représenté via les modèles existants, ceux-ci étant (pour ce problème) soit trop complexes à résoudre, soit inapplicables au vu des hypothèses nécessaires.

**Complexité inhérente au problème**

Le problème de Thales implique une complexité combinatoire forte. La complexité de résolution d'un DEC-POMDP « standard » est (pour rappel), exponentielle en le nombre d'agents, ce qui rend ce type d'approche inapplicable à un problème comme celui-ci. Les structures manipulées sont également très grandes. Prenons un exemple simple, dans lequel 5 agents évoluent sur une grille de dimension 10x20. Si l'on se contente d'un modèle simple, dans lequel l'état décrit uniquement la position ($x$,$y$) des agents, on aura à manipuler $(10 \times 20)^5 = 320$ milliards d'états. Il est bien entendu impossible de manipuler ce type d'instance : il faut donc décomposer le problème, afin de casser cette complexité combinatoire.

Traitons le problème d'un agent donné, parmi les 5 impliqués dans le problème, et voyons comment il peut travailler sur un sous-espaces des 320 milliards d'états possibles. On pourrait imposer, arbitrairement, une indépendance entre agents et ne considérer que les 200 états individuels, mais il serait alors impossible de calculer une bonne politique jointe. Par contre, les capteurs embarqués d'un agent ont en général des capacités limitées, notamment en termes de portée. Supposons alors que l'agent ne puisse observer ce qu'il passe autour de lui que dans un rayon de 2 cases (par exemple). Il pourra ainsi observer 0, 1, 2, 3 ou 4 agents sur ces 25 cases. On aura donc un total de $(10 \times 20) \times (1 + 25 + 25^2 + 25^3 + 25^4) \simeq 81{,}4$ millions d'états possibles. Ce nombre, bien que toujours trop grand pour être traité comme tel, est déjà bien plus raisonnable. Le problème de Thales est donc beaucoup trop complexe pour être traité de façon complètement jointe entre agents. Il faut, au contraire, tirer parti de la localité des interactions pour réduire le nombre d'états à traiter, ainsi que le nombre d'observations associées.

**Inapplicabilité des modèles dérivés**

Ce problème pouvant impliquer un grand nombre d'agents, il est inutile d'espérer le traiter via une approche DEC-POMDP naïve. Il faut donc s'intéresser aux modèles dérivés. On ne peut cependant pas restreindre les dépendances entre agents. Comment, par exemple, représenter les risques de collision si on suppose une indépendance sur les transitions ? Comment, de même, prendre en compte le voisinage lors de la formation du convoi si on suppose une indépendance sur les observations ? Nous étudions ici un problème de convoyage, mais cette difficulté apparaît dans de nombreux problèmes « réels ». Il sera donc impossible de modéliser ces problèmes via des méthodes basées indépendances, du type ND-POMDP ou DEC-POMDP factorisé.

Les modèles permettant de manipuler des interactions dynamiques semblent beaucoup plus souples du point de vue de la représentativité. Pourtant, là encore, des difficultés se posent quant à la façon de détecter ou prévoir ces interactions. Prenons l'exemple du modèle DPCL, utilisant des « points de coordination » sous forme de tâches à réaliser conjointement. Comment représenter l'évolution des interactions au sein d'un convoi via ce genre d'approche ? Les modèles IDMG et DEC-SIMDP permettent de manipuler des interactions plus complexes mais là encore, il semble difficile de vérifier les hypothèses d'observabilité totale sur l'état individuel (les capteurs étant rarement parfaits) et de communication gratuite/illimitée. Dans notre cas,

la communication est interdite, mais même autorisée celle-ci est coûteuse en temps et en énergie (donc non-gratuite), et repose sur une bande passante de taille fixe (donc non-illimitée). Ainsi, là encore, ces modèles ne permettent pas la représentation de ce type de problème.

### 5.2.2 Nos besoins en modélisation

Afin de traiter ce problème, il nous faut un modèle permettant de tirer parti de la localité des interactions, mais doté d'une représentative suffisamment large au vu des contraintes imposées.

#### Briser la complexité combinatoire

La complexité de ce type de problème vient principalement de deux aspects :

1. la taille de l'espace d'états, qui augmente exponentiellement en le nombre d'agents et qui impose de planifier sur un grand nombre de situations possibles,
2. la résolution jointe (ou centralisée), qui implique là encore une complexité exponentielle en le nombre d'agents.

Ainsi, si on veut pouvoir traiter ce type de problème, il faut avant tout surmonter ces deux difficultés. Les approches existantes supposent une résolution centralisée, mais est-ce vraiment la seule approche possible ? Il existe de nombreuses approches, notamment dans le domaine des SMA (systèmes multiagents), reposant sur une prise de décision complètement décentralisée. On peut par exemple citer l'étude du flocking, réalisée par [Reynolds, 1987], visant à reproduire avec un ensemble d'agents le déplacement « en nuée » d'un groupe d'oiseaux. Cette étude repose sur un concept simple : chaque agent observe, à tout moment, son voisinage. Il prend alors des décisions, basées sur ces observations, afin de se déplacer. La prise de décision est ainsi complètement locale, l'agent se contentant de réagir à ses voisins directs, mais le comportement de groupe qui en émerge est un déplacement en nuée.

Le flocking nous a inspiré dans notre réflexion, dans le sens où il serait possible d'obtenir un bon comportement de groupe, à partir de bons comportements individuels. Il ne faut pas, cependant, que les agents agissent de façon totalement égoïste. Prenons l'exemple d'un être humain se déplaçant au sein d'une foule. Cet humain n'a aucun contrôle sur les actions des autres individus présents, mais il peut malgré tout les prendre en compte et faire des prévisions sur leurs déplacements, afin de ne pas créer de collision. Pourquoi ne pas, de la même façon, faire calculer à l'agent une politique individuelle, qui serait basée sur des suppositions quant au comportement des autres agents ?

On prend bien entendu le risque, via cette approche, de perdre l'optimalité de la solution. Là encore, on peut faire le parallèle avec l'humain évoluant dans une foule : celui-ci pourrait raisonner sur l'ensemble des actions possibles pour l'ensembles des autres humains présents, afin de déterminer l'action optimale pour évoluer au sein de cette foule. Un tel raisonnement est toutefois complètement irréaliste, de par sa complexité. De la même manière, il est très difficile de résoudre des problèmes multiagents partiellement observables, même en se limitant aux

quelques agents présents dans le voisinage. Toutefois, rechercher une politique jointe optimale est-il obligatoire ? Nous faisons le choix, ici, de sacrifier cette optimalité en échange d'un passage à l'échelle, tout en supposant que nous serons capables de planifier des comportements certes sub-optimaux, mais malgré tout de qualité satisfaisante.

On peut, dès l'instant où on se base sur ce type d'approche (avec un agent qui calcule une politique individuelle basée sur sa propre vision du monde), réduire considérablement le nombre d'états à considérer. Supposons donc que l'on traite le problème de décision d'un agent donné. On peut alors, par exemple, considérer uniquement les agents observables, comme vu précédemment (la réduction de 320 milliards à 80 millions d'états). On pourrait pousser plus loin encore ce raisonnement, et ne considérer à un moment donné que les agents dont on dépend (qu'il s'agisse des récompenses ou des transitions), c'est à dire les **agents en interaction**. Pourquoi, en effet, maintenir dans l'état une information sur des agents n'ayant aucune influence sur notre prise de décision ? Une difficulté se pose cependant : comment, si on ignore l'état de ces agents, prévoir le moment où ils entreront en interaction ? Ces questions, du domaine de l'algorithmique, seront traitées dans la dernière partie.

#### Ne pas introduire d'hypothèse limitative

Gérer des interactions dynamiques ajoute une grosse part d'incertitude au problème. Ce type d'approche suppose en effet que l'on puisse prévoir (ou détecter) les interactions, contrairement aux modèles statiques dans lesquels ces interactions sont prédéterminées. On a vu que les modèles existants introduisent tous des hypothèses fortes pour pouvoir détecter ces interactions, qu'il s'agisse de les cadrer via une allocation de tâches ou d'avoir une observabilité totale sur celles-ci. Il nous faudra nous aussi traiter cette difficulté, mais nous ne pouvons pas nous permettre d'introduire ce type d'hypothèse (en raison des contraintes inhérentes au problème de Thales).

L'agent devra donc se contenter de ses observations pour prévoir les interactions, celles-ci pouvant survenir n'importe quand et avec n'importe quel agent. Dans le problème de Thales, nous supposerons que l'agent est capable d'observer ce qui se passe à 360° autour de lui, mais qu'il est limité en termes de portée. Nous supposerons de plus que ses capteurs étant bruités, les observations effectuées peuvent être erronées. L'agent devra donc maintenir une distribution de probabilités sur les états des agents en interaction, et considérer à tout moment la possibilité que de nouveaux agents, non-observés jusque là, intègrent l'interaction.

## 5.3 Autres applications envisageables

Nous avons décrit nos besoins, dans ce chapitre, au travers d'un exemple applicatif concret (le problème de Thales). Pourtant, notre travail ne se veut pas une réponse ad-hoc à ce problème, mais bien une approche générale, destinée à tout problème de type DEC-POMDP impliquant des interactions locales et dynamiques. Ainsi, cette section présente à titre d'exemple d'autres applications possibles à notre travail, qui nous servirons de benchmarks à la fin de ce document.

### 5.3.1 Le problème de Dassault

Le groupe industriel Dassault Av. a proposé un second exemple d'application pouvant être traité via nos travaux, basé sur la gestion d'un aéroport (figure 5.3). L'idée est la suivante : un ensemble d'avions évoluent sur un aéroport, chacun devant choisir les sections de piste à emprunter pour se rendre à la zone de décollage. Ces avions doivent de plus respecter des contraintes fortes les uns relativement aux autres, afin d'éviter tout risque d'accident. Il y a donc, là encore, interaction locale entre agents (un avion est en interaction avec ceux présents sur des pistes proches). Il y a également observabilité limitée, non seulement puisque des avions sont présents hors de l'aéroport et peuvent intégrer le problème à tout moment, mais aussi puisque les observations sont transmises aux avions par des opérateurs humains, qui ne peuvent pas toujours transmettre la totalité des informations.

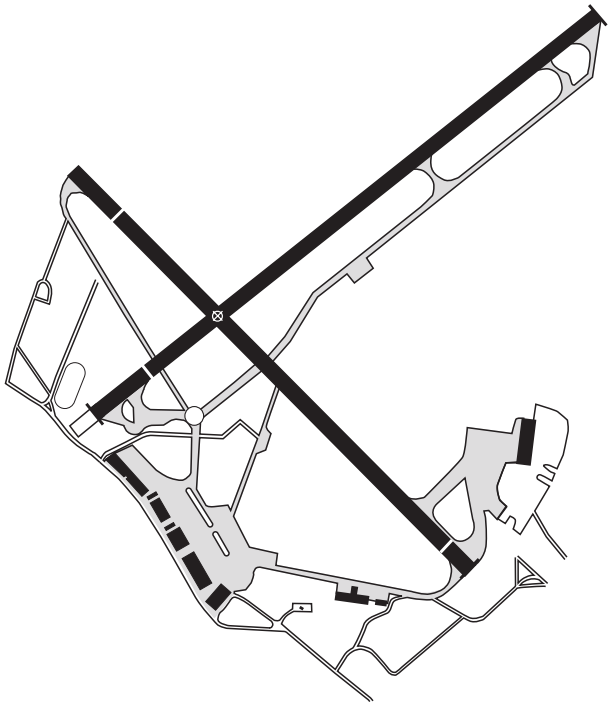

FIGURE 5.3 – Exemple d'environnement pour le problème de Dassault

Ce problème présente toutefois un certain nombre de différences. Ici, les interactions ne sont pas vraiment géographiques, puisque 2 avions ne seront jamais sur la même piste au même moment. Par contre, les avions doivent respecter un ordre de passage sur les pistes, afin de décoller chacun au bon moment. On pourra par exemple imaginer qu'un avion doive attendre à un croisement, n'ayant le droit de passer qu'après un autre avion qui n'est pas encore arrivé. Les agents devront de plus considérer l'incertitude inhérente au problème, les pistes pouvant être ouvertes ou fermées à la circulation selon les choix d'un opérateur humain, qui lui-même agit selon les conditions extérieures observables (par exemple, la météo).

### 5.3.2 Benchmarks diverses

On peut imaginer, au delà de ces deux exemples d'application tirés du monde réel, résoudre tout type de problème nécessitant un modèle décisionnel et reposant sur des interactions locales. Prenons par exemple le problème d'un complexe touristique, ayant à répartir les touristes au sein d'un ensemble d'hôtels. On imagine que l'on veuille automatiser ce processus, chaque hôtel étant représenté par un agent de gestion. Nous ne donnerons ici qu'une description générale de ce problème, le but étant de montrer comment on peut traiter des problèmes autres que des problèmes de navigation.

Ainsi, à chaque pas de temps, de nouveaux clients se présentent aux différents hôtels, et les clients déjà présents ont une probabilité de partir. Lorsqu'un client arrive, l'agent peut soit l'accepter, soit le transmettre vers un autre hôtel. Ainsi, loger un client rapporte une récompense, mais l'accepter lorsque l'hôtel est plein inflige une pénalité forte, tandis que le rediriger vers un autre hôtel inflige une pénalité faible. Pour finir, après chaque pas de temps, un agent reçoit un rapport des autres hôtels sur leur pourcentage d'occupation, mais ce rapport peut contenir des erreurs, ce qui se traduit par une observabilité limitée sur l'état des autres agents.

On dispose donc d'un problème de type DEC-POMDP, dans lequel les agents sont interdépendants. Les transitions d'un agent donné dépendent en effet non seulement de ses actions (gestion des clients présents) et du hasard (arrivée de nouveaux clients), mais également des actions des autres agents (transmission de clients). On ajoute une localité à ces interactions, en supposant qu'un hôtel dispose de $n$ bus pour transporter ses clients, et qu'il ne peut donc transmettre des clients que vers $n$ hôtels qu'il aura choisi. Modifier les destinations d'un ou plusieurs bus nécessite 1 pas de temps, durant lequel on ne pourra envoyer aucun client. Ainsi, l'agent sera en interaction non seulement avec les $n$ hôtels qu'il a choisi, mais aussi avec les hôtels qui ont choisi de lui envoyer des clients, et ces interactions évolueront dynamiquement.

On peut, pour finir, ajouter une possibilité de coordination explicite : deux hôtels A et B peuvent décider d'installer une ligne de bus de meilleur qualité entre leurs établissement respectifs. Il suffit pour cela que A attribue 2 de ses bus vers B, et vice-versa (B attribue 2 bus vers A). Ces 4 bus seront alors remplacés par un bus de qualité, qui se traduira par un coût nul pour les deux agents lors de l'envoi de client sur cette ligne. Dans le cas où un seul agent attribue 2 bus, il n'y gagne aucun avantage (relativement au fait de n'attribuer qu'un seul bus). Ce problème, artificiel, est intéressant dans le sens où il n'implique aucune interaction géographique, mais bien des relations de dépendance entre agents.

## Conclusion

Ce chapitre nous a permis d'identifier une gamme de problèmes « durs », non-traitables via les approches existantes. Ces problèmes sont toutefois porteurs d'une structure qu'il est possible d'exploiter, puisque les interactions sont en général locales. Nous avons ainsi pu constater qu'un agent n'était en interaction qu'à certains moments, avec certains agents, et qu'il était possible

d'utiliser cette structure pour simplifier le problème à résoudre. Nous avons également étudié la question de la résolution conjointe : l'explosion combinatoire empêchant ce type d'approche, on préférera une méthode où chaque agent résout son propre problème, mais intègre l'existence des autres agents et effectue des suppositions sur leurs actions à venir.

Nous disposons maintenant d'une première intuition quant à la façon dont on peut adresser ce type de problème. Le chapitre suivant propose une analyse plus poussée de la façon dont on peut décomposer un problème et en extraire une structure facilitant sa résolution. Nous introduirons les notions importantes pour notre approche (état relatif, différence entre tâche et coordination, etc.), et nous illustrerons celles-ci au travers du problème de Thales. Nous pourrons alors, dans le dernier chapitre de cette partie, présenter notre modèle en lui-même.

# Chapitre 6

# Préliminaires

**Sommaire**



Il faut, si on désire optimiser la résolution d'un problème multiagent, commencer par extraire la structure de celui-ci. Ainsi dans ce chapitre, nous montrerons comment extraire la structure d'un problème, et ce via trois axes différents. Nous commencerons par décomposer le problème en lui même, en identifiant les différents sous-problèmes qu'il englobe. Nous verrons ensuite comment chacun de ces sous-problèmes inclut une composante individuelle (comment l'agent atteint ses objectifs) et une composante d'interaction (prendre en compte l'influence des autres agents). Nous verrons finalement comment l'agent peut caractériser les états des autres agents, relativement au sien. Ces trois aspects, que nous illustrerons par le problème de Thales, constitueront le squelette de notre modèle, qui sera décrit dans le chapitre suivant.

## 6.1 Décomposition du problème

Un problème donné inclut, en général, plusieurs sous-problèmes. Si ces problèmes sont suffisamment décorrélés, il peut s'avérer intéressant d'extraire chacun d'eux afin de décomposer la résolution du problème global. Cette section montre l'intérêt qu'il y a à identifier chaque composante d'un problème donné, tout en maintenant des liens entre celles-ci.

### 6.1.1 Extraction des différentes composantes

On peut, en général, identifier plusieurs sous-problèmes, au sein d'un problème de planification. Prenons l'exemple d'un groupe d'agents déménageurs : ceux-ci sont entrain de ranger un entrepôt, qui contient un ensemble de caisses, dont certaines trop lourdes pour être portées par un agent seul. Ces agents devront alors faire face à plusieurs problèmes distincts. On peut citer par exemple :

- trouver les bonnes caisses à ranger pour optimiser le comportement de groupe,
- se déplacer dans l'entrepôt sans entrer en collision avec les autres agents,
- aider les autres agents à déplacer les caisses trop lourdes...

Ces composantes sont, de plus, **faiblement couplées**. Prenons par exemple le problème consistant à se déplacer dans l'entrepôt. Celui-ci a peu de liens (voir aucun) avec le problème consistant à s'unir pour soulever des caisses lourdes. Pourquoi, dans ce cas, traiter le problème comme un tout (ce qui implique une forte complexité combinatoire), alors que l'on pourrait résoudre chacune de ces composantes séparément, ou quasi-séparément ? Il est donc intéressant, dans ce genre de situation, d'extraire les sous-problèmes (ou composantes) inclus dans un problème donné, afin de simplifier la résolution de ce problème.

La question de la résolution sera laissée à la partie III (traitant des algorithmes), mais on peut dès à présent identifier ces composantes, afin de les inclure dans le modèle. Appliquons ce raisonnement au problème de Thales. Former un convoi va dépendre du nombre d'agents présents : une formation optimale pour 3 agents par exemple, ne sera pas nécessairement la meilleure pour 10 agents (figure 6.1).

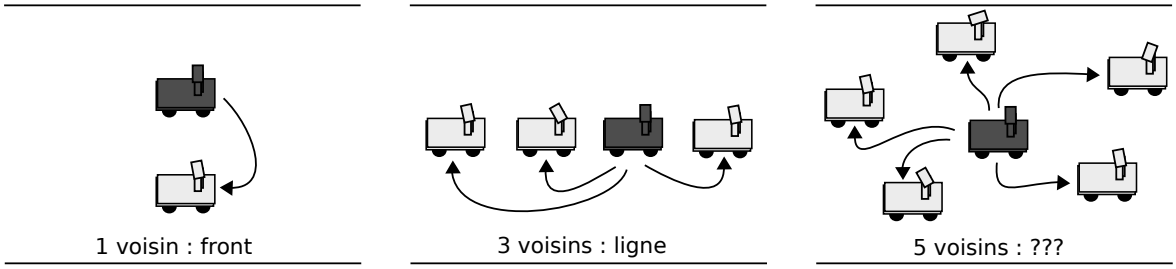


FIGURE 6.1 – Exemple de composantes au sein du problème de Thales

Trois situations sont représentées sur cette figure : interaction avec 1, 3 et 5 agents. Ici, l'agent cherche à rouler de front avec son voisin tant qu'il n'y a interaction qu'avec un seul agent (partie gauche de la figure), puis adopte une formation en ligne lorsque d'autres agents se joignent au groupe (partie centrale). Lorsque ceux-ci sont trop nombreux pour une coordination efficace (partie de droite), l'agent ne cherche plus à maintenir une formation en ligne mais se contente d'éviter les collisions dans une formation éparpillée (ces formations ne sont données qu'à titre d'exemple). On choisit donc de fournir $N+1$ composantes à l'agent, avec $N$ le nombre maximum d'agents en interaction simultanée avec lui. On aura ainsi une composante contenant les états où l'agent n'est en interaction avec personne, une autre pour les états dans lesquels il

est en interaction avec exactement 1 agent, etc., et une dernière composante contenant les états où l'agent est en interaction avec $N$ agents en même temps.

Ainsi à tout instant, l'agent choisira une composante selon le nombre d'agents en interaction (ainsi éventuellement que selon la situation : on peut imaginer que le problème à traiter sera différent selon qu'il y ait danger ou non par exemple), et fera en sorte de maintenir la formation associée à cette composante. On peut imaginer des convois mixtes (le choix de la formation dépendant uniquement des observations locales), où par exemple les agents en tête de convoi formeront une ligne, tandis que les agents à l'arrière seront éparpillés.

Ce processus d'extraction des composantes, que l'on a illustré via le modèle de Thales, peut être reproduit peut importe le problème que l'on traite. Il suffit pour cela d'identifier chaque sous-problème, tel que chacun puisse être résolu quasi-indépendamment des autres. On crée ensuite une composante pour chaque sous-problème, contenant l'ensemble des états associés. Dans le pire cas (problème absolument indécomposable), on n'aura qu'une seule composante, regroupant l'ensemble des états du problème.

### 6.1.2 Dépendances entre composantes

Nous avons réparti le problème en un ensemble de composantes *semi-indépendantes* : celles-ci ne sont, en effet, pas complètement disjointes. Prenons à nouveau l'exemple des agents déménageurs : on a considéré que se déplacer dans l'entrepôt pour aller chercher une caisse était un problème indépendant de la tâche consistant à se coordonner pour soulever une caisse lourde. Il est vrai que, lorsqu'il se déplace, l'agent n'a pas à prendre en compte la façon dont il soulèvera une caisse... Mais ce déplacement peut l'amener face à une caisse qui sera trop lourde pour être soulevée par lui seul ! Ainsi, lors de son déplacement, l'agent devra considérer cette possibilité, qui le poussera à se diriger ou non vers certaines caisses selon l'intérêt qu'il aura à les traiter. La composante de déplacement ne sera donc plus « se déplacer en évitant les collisions », mais « se déplacer en évitant les collisions, sachant où ce déplacement va nous mener ».

Ainsi, les composantes décrivent des problèmes certes disjoints, mais tels que l'on peut passer de l'un à l'autre. De la même façon, dans le problème de Thales, l'agent devra prendre en compte la possibilité qu'après chaque action, d'autres agents rejoignent ou quittent l'interaction, ce qui pousserait le groupe à changer de composante et donc modifier la formation. Partant de cette constatation, la résolution du problème devra se faire à deux niveaux : optimiser le comportement au sein de chaque composante, et atteindre la composante la plus intéressante (figure 6.2).

Sur cet exemple, la composante à 3 agents, permettant une formation en ligne, est la plus intéressante. Ainsi, l'agent optimisera son comportement en fonction de deux critères :

- son voisinage, afin de maintenir une formation optimale au vu des agents présents,
- les composantes atteignables, afin de faire transiter le groupe vers la meilleure possible.

Imaginons un exemple dans lequel l'agent est entouré de nombreux voisins (composante à plus de 3 agents). Celui-ci dispose de plusieurs actions qu'il peut exécuter :

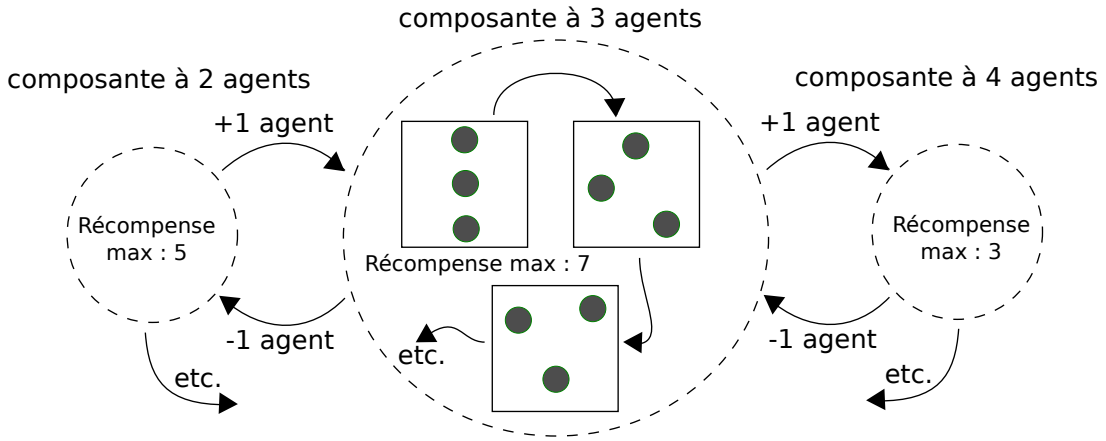


FIGURE 6.2 – Raisonnement intra et inter composantes

1. se déplacer vers le haut : cette action permet de quitter l'amas d'agents pour intégrer une formation en ligne, mais risque de créer une collision,
2. se déplacer à gauche ou à droite : cette action maintient l'agent dans l'amas,
3. se déplacer vers le bas : cette action permettra à l'agent, au tour suivant, de quitter l'amas pour intégrer une formation en ligne, sans créer de collision.

Ici, si l'agent se contente d'un raisonnement intra-composante, il choisira au hasard les actions 2 ou 3, celles-ci étant équivalentes (pas de collision). Si l'agent suit un raisonnement inter-composantes au contraire, il choisira l'action 1, qui l'amènera immédiatement dans la meilleure composante possible. Un agent qui raisonne selon ces deux critères finalement, choisira l'action 3. Là encore, le calcul de la politique optimisant ces deux aspects relève de l'algorithmique et sera donc abordé dans la partie III de ce document. On doit malgré tout conserver ce point à l'esprit, afin de modéliser correctement le problème.

## 6.2 Séparer la tâche et les interactions

Diviser un problème en sous-problèmes n'est pas le seul moyen pour casser la complexité combinatoire lors de la résolution. On peut également réduire sa difficulté en séparant sa composante individuelle (la tâche) de sa composante de coordination (les interactions). Cette section montre comment décomposer un problème selon ces deux éléments, et baser la résolution sur cette décomposition.

### 6.2.1 Problème individuel, ou problème inter-agents ?

Partons de l'exemple, simple, d'un être humain évoluant au sein d'une foule afin de sortir d'un bâtiment. Cet humain gardera deux problèmes bien distincts à l'esprit : se diriger vers la porte du bâtiment, et éviter les collisions avec les autres humains tentant d'entrer. Bien entendu, des relations existent entre ces problèmes (typiquement, l'humain devra dévier de son chemin

vers la porte pour éviter des collisions). Dans l'ensemble cependant, ces deux problèmes sont semi-indépendants et peuvent être traités séparément. Les problèmes « réels » de planification permettent, en général, d'appliquer ce raisonnement en séparant l'aspect individuel de l'aspect interactions (figure 6.3).

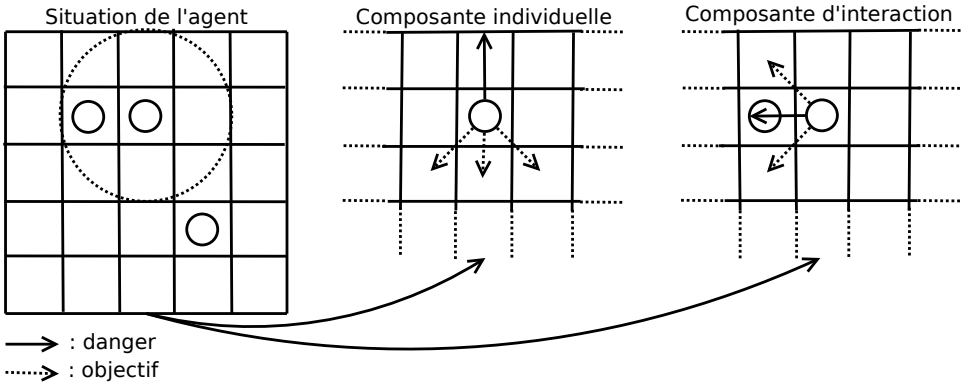


FIGURE 6.3 – Exemple de décomposition

Sur cette figure (partie gauche), un agent est en interaction avec un voisin (l'autre étant trop loin pour être perçu). Cet agent doit alors prendre une décision en se basant à la fois sur un critère individuel (partie centrale de la figure : éviter les collisions avec les murs) et sur un critère d'interaction (partie droite : éviter les collisions entre agents). Il prend alors la décision qui représente le juste milieu entre ces deux critères.

### Un problème individuel : la tâche

On peut, en général, extraire une composante individuelle à tout problème que l'on tente de résoudre. Sur le problème de Thales par exemple, le groupe se déplace vers un objectif donné, en maintenant une formation en convoi. On peut en extraire le problème individuel, consistant à se déplacer vers l'objectif (sans prendre en compte la présence des autres agents). Ce problème (que l'on nommera la **tâche de l'agent**) peut alors être traité et résolu via une approche monoagent. Il faudra pour cela définir :

- l'ensemble des états (les états individuels de l'agent),
- les fonctions de transition et récompense associées, qui peuvent être « fausses » puisque l'agent considère qu'il agit toujours seul (par exemple, l'agent peut avoir un voisin en face de lui et donc risquer la collision, mais considérer qu'il va être capable d'avancer car il ignore ce voisin),
- une fonction d'observation, sur les états individuels.

On pourra alors calculer une politique individuelle, permettant à l'agent d'adopter un comportement égoïste optimal (c'est à dire le meilleure comportement possible, tant qu'on ignore l'existence des autres agents).

**Un problème inter-agents : les interactions**

On peut, tout comme on extrait une composante individuelle, extraire la composante d'interaction. Cette composante décrira alors la façon dont l'agent interagit avec les autres agents, c'est à dire l'influence qu'auront ceux-ci sur ses transitions et récompenses. Sur le problème de Thales par exemple, la composante d'interaction décrira le problème de la formation du convoi et de l'évitement des collisions, indépendamment de la tâche de l'agent (c'est à dire en ignorant le problème consistant à se rendre sur l'objectif).

L'idée est ici de raisonner sur les interactions entre agents. Une collision par exemple, sera toujours une collision, indépendamment de la position de l'agent. Celle-ci aura toujours le même effet (immobiliser l'agent) et infligera toujours la même pénalité (récompense négative représentant le coût d'une collision). Il est donc inutile, au sein de cette composante, de considérer les états joints : on choisira plutôt d'énumérer l'ensemble des interactions possibles, et de considérer chaque interaction comme un état dans lequel l'agent peut se situer (ainsi, on considérera chaque interaction comme une « compression » d'un ensemble d'états joints). On peut alors construire le problème d'interaction, en définissant :

- l'ensemble des interactions possibles (qui joueront le rôle d'états),
- une fonction de transition entre ces interactions, selon les actions de l'agent (par exemple la probabilité que l'agent entre en collision avec un voisin, après s'être déplacé),
- la fonction de récompense associée (pénaliser les collisions par exemple),
- une fonction d'observation sur la voisinage (l'agent observera par exemple qu'un voisin est présent devant lui).

On pourra alors calculer une politique d'interaction, permettant à l'agent d'optimiser son comportement vis-à-vis de ses voisins, sans se soucier de se tâche individuelle (politique altruiste).

### 6.2.2 Tâche et interactions : distincts, mais interdépendants

On peut choisir de traiter séparément la tâche de l'agent, et son problème d'interaction, mais on disposera alors de deux politiques distinctes. Comment, dans ce cas, choisir le comportement à suivre ? On aura besoin d'une méthode permettant la fusion de ces deux politiques (figure 6.4).

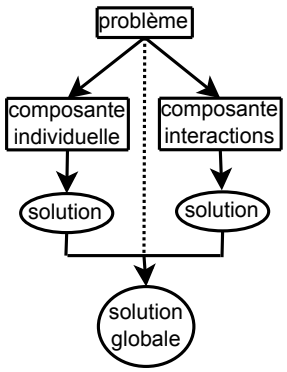


FIGURE 6.4 – Décomposition tâche/interactions

Une première approche serait de choisir le « juste milieu » entre les deux politiques. Prenons un exemple : l'agent est face à un mur, sa politique individuelle attribue une valeur de 3 à l'action `reculer`, $-3$ à `avancer` et 0 à l'action `pivoter`. D'un autre côté, cet agent a un voisin derrière lui, donc sa politique d'interaction attribue une valeur de $-3$ à `reculer`, 3 à `avancer` et 0 `pivoter`. Dans ces conditions, quelle politique suivre ? La politique individuelle, qui dicte de reculer, ou la politique d'interactions qui dicte d'avancer ? En réalité, ni l'une ni l'autre. Le juste milieu consisterait ici à pivoter (pour peut-être, plus tard, s'éloigner via un côté).

Cette approche est intéressante, mais soulève tout de même certains problèmes :

- tout d'abord, l'aspect interaction et l'aspect individuel sont souvent entremêlés (certaines interactions par exemple, n'arriveront jamais dans certains états individuels de l'agent). On pourra donc probablement améliorer la politique d'interactions, si on a connaissance de la politique individuelle (en diminuant l'erreur sur les interactions atteignables).
- Ensuite, cette approche suppose un impact à horizon 1 des interactions sur le comportement individuel (après chaque transition, on cherche à nouveau le juste milieu entre les deux politiques). On pourrait, là encore, améliorer le comportement global en considérant l'impact à long terme des interactions.

Une seconde approche, permettant de palier ces difficultés, serait de calculer une seule politique, dont les états seraient l'ensemble des couples `(état-individuel;état-interaction)` possibles. Calculer une telle politique serait plus dur que de calculer les deux politiques séparées, mais malgré tout nettement plus simple que de résoudre le DEC-POMDP équivalent. En effet, dans un DEC-POMDP, le nombre d'états à traiter serait de l'ordre de $|S_i|^I$ ($S_i$ l'ensemble des états individuels et $I$ le nombre d'agents), tandis qu'avec cette approche, il faudrait traiter au pire $|S_i| \times (|\{interactions\}|^I)$ états, sachant qu'il y a en général peu d'interactions possibles (sur le problème de Thales par exemple, on aura `devant`, `derrière`, `collision`, etc.).

Cette seconde approche permet donc une résolution fortement simplifiée du problème multiagent, tout en maintenant dans l'état une information aussi expressive que dans un DEC-POMDP. Gardons malgré tout à l'esprit que notre but est de développer une approche fournissant des politiques de bonne qualité, mais pas nécessairement optimales. Ainsi, l'état sera certes complet, mais la résolution se faisant sur les actions de l'agent uniquement (et non sur les actions jointes), les politiques obtenues seront sous-optimales. Ces questions seront développées plus en détails dans la partie III (algorithmes).

## 6.3 Interactions locales et dynamiques

La notion d'interaction entre agents est, au vu des sections précédentes, centrale dans notre approche. Il nous faut donc formaliser cette notion : pour cela, nous introduisons dans cette section la notion d'état relatif, que nous illustrons par le problème de Thales.

### 6.3.1 Notion de relation entre agents, état relatif

Il faut, si on veut raisonner sur les interactions, commencer par énumérer celles-ci. On introduit pour cela la notion de **relation** entre agents. Dans le reste de ce document, on considérera que l'état d'un agent dans le problème de Thales est donné par un triplet ($x$,$y$,$d$) avec ($x$,$y$) ses coordonnées géographiques (dans le plan) et $d$ sa direction (nord, sud, est ou ouest).

**Définition 27 (Relation)** *Une relation $\mathcal{R}^l$ désigne une propriété $l$ que vérifient deux états $s_i$ et $s_j$ (on peut imaginer la propriété* `face` *par exemple, qui sera vérifiée entre autres par les états $s_i = (x_0,y_0,nord)$ et $s_j = (x_0,y_1,est)$, d'où* `face(`$s_i$`;`$s_j$`)=vrai`*). On peut alors noter :*

$$\mathcal{R}^l = \{(s_i; s_j) | l(s_i; s_j) = vrai\}$$

On va donc énumérer toutes les relations pouvant lier deux agents, afin de raisonner sur celles-ci. La figure 6.5 montre l'application de ce procédé au problème de Thales.

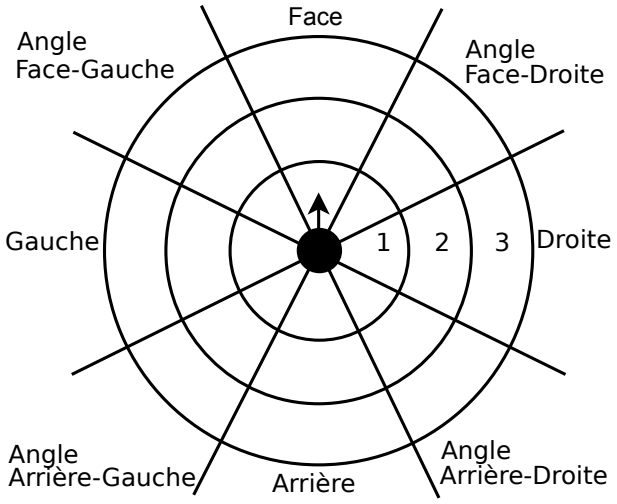


FIGURE 6.5 – Notion de relation appliquée au problème de Thales (1)

On a donc fait le choix, sur ce problème, de diviser l'espace autour de l'agent selon deux critères : l'orientation relative (face, arrière, etc.) et la distance (couche 1, 2, etc.). On aura donc une relation pour chaque combinaison possible de ces critères (face1, face2, face3, arrière1, arrière2, etc.). Notre objectif était de raisonner sur les interactions, indépendamment des états individuels. On introduit pour cela la notion d'**état relatif**.

**Définition 28 (État relatif)** *Pour un agent $i$ donné, l'état relatif $sr_j$ désigne la relation entre son état $s_i$ et l'état $s_j$ de l'agent en interaction $j$.*

Sur l'exemple de Thales, imaginons que l'agent $i$ perçoive un voisin $j$ en face de lui, ainsi qu'un autre voisin $k$ sur sa droite. Les états de ces trois agents vérifieront alors `face(`$s_i$`;`$s_j$`)=vrai` et `droite(`$s_i$`;`$s_k$`)=vrai`. On en déduit donc les états relatifs $sr_j = face$ et $sr_k = droite$, d'où l'**état joint relatif** $sr = (sr_j,sr_k)$.

**Définition 29 (État joint relatif)** *L'état joint relatif sr d'un agent i désigne, à un moment donné, l'ensemble des états relatifs $sr_j$ entre i et chacun des agents $j \neq i$ en interaction avec i.*

Ainsi, à chaque couple $(s_i,s_j)$ correspondra exactement un état relatif $sr_j$. Un voisin en face de l'agent par exemple sera toujours dans l'état relatif `face`. Littéralement, cela signifie (avec $\mathcal{R} = \{\mathcal{R}^1,\ldots,\mathcal{R}^{|\mathcal{R}|}\}$ l'ensemble des relations possibles) que $\forall \mathcal{R}^i,\mathcal{R}^j \in \mathcal{R}, \mathcal{R}^i \cap \mathcal{R}^j = \emptyset$.

Cette propriété d'exactitude des états relatifs n'interdit par pour autant qu'il y ait une incertitude dans leur observation. Prenons à nouveau l'exemple de Thales, et imaginons qu'un agent observe un voisin dans la zone face-droite. Les capteurs de l'agent sont, en général, imparfaits. On peut alors supposer que cette observation sera bruitée, et l'agent déduira de son observation que le voisin est probablement en face-droite (avec, par exemple, une probabilité de 0,9), mais peut-être simplement en face ou à droite (probabilité de 0,05 chacun).

La figure 6.6 est un exemple d'application des états relatifs au problème de Thales. Ici, l'idée est de plaquer la représentation des interactions proposée dans la figure précédente, sur la grille représentant l'environnement de l'agent. On peut alors construire les états relatifs, selon les zones dans lesquelles se situent les états des voisins. Imaginons par exemple que l'agent se trouve dans l'état $(x_0,y_0,nord)$ et qu'un voisin soit en $(x_0,y_4,-)$. Cet état tombe alors dans la section « face », dans la deuxième couche. L'état relatif de cet agent sera alors `face2`. On peut donc, pour tout état de l'agent et tout état d'un voisin, déduire l'état relatif associé.

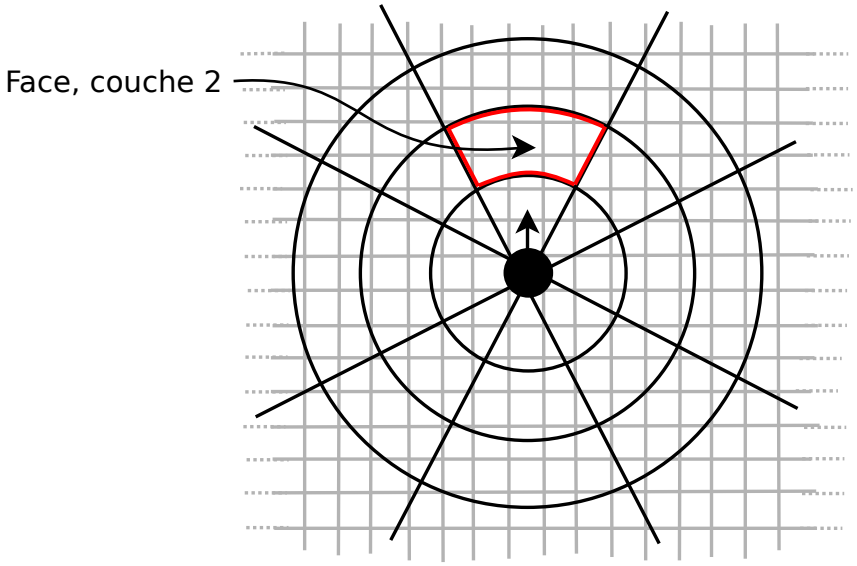


FIGURE 6.6 – Notion de relation appliquée au problème de Thales (2)

Ainsi, chaque situation est associée à exactement un état relatif, mais l'inverse n'est pas forcément vrai ! Sur le schéma précédent, on voit que la relation `gauche2` par exemple, correspond à 6 états possibles pour l'agent concerné. Ainsi, selon la précision que l'on donne au découpage des relations, on aura une connaissance plus ou moins exacte de l'état du voisinage. On introduit ici la notion d'instance d'un état joint relatif.

**Définition 30 (Instance d'un état joint relatif)** *Un état relatif $sr_j$ correspond à plusieurs états possibles pour l'agent $j$. De même, un état joint relatif $sr = (sr_j, sr_k, \ldots, sr_l)$ correspond à plusieurs états joints possibles $s = (s_i, s_j, s_k, \ldots, s_l)$. L'ensemble de ces états joints sont appelés les* ***instances*** *de $sr$.*

D'autres sources d'incertitude sont possibles au sein des états relatifs. Tout d'abord, est-il possible de distinguer combien d'agents vérifient en même temps une relation donnée ? Imaginons par exemple que l'on observe l'état relatif `gauche2` : cela signifie que l'agent a pu observer, via ses capteurs, la présence d'agents à gauche de sa position, dans la deuxième couche. Mais si on se place dans un cadre « réaliste », il sera probablement difficile de distinguer combien d'agents sont présents à cette position (ceux-ci pouvant se cacher mutuellement). Pour cette raison, on pourra supposer qu'un état relatif concerne « exactement un agent », ou « au moins un agent ».

Ensuite, toujours dans cette idée d'imperfection des capteurs utilisés, ne peut-on pas imaginer que certaines relations en cachent d'autres ? Imaginons par exemple qu'il y ait des agents en `gauche2`, et d'autres en `gauche3` : une hypothèse réaliste serait de supposer que les agents de la deuxième couche cachent ceux de la troisième couche. L'agent observerait alors simplement `gauche2`. On a donc, pour résumer, trois sources possibles d'incertitude lorsque l'on observe un état joint relatif $sr$ :

1. $sr$ correspond à plusieurs instances possibles,
2. chaque $sr_j \in sr$ peut correspondre à un ou plusieurs agents,
3. certaines relations peuvent exister, sans être observées dans $sr$.

La figure 6.7 propose un bilan des relations obtenues pour le problème de Thales.

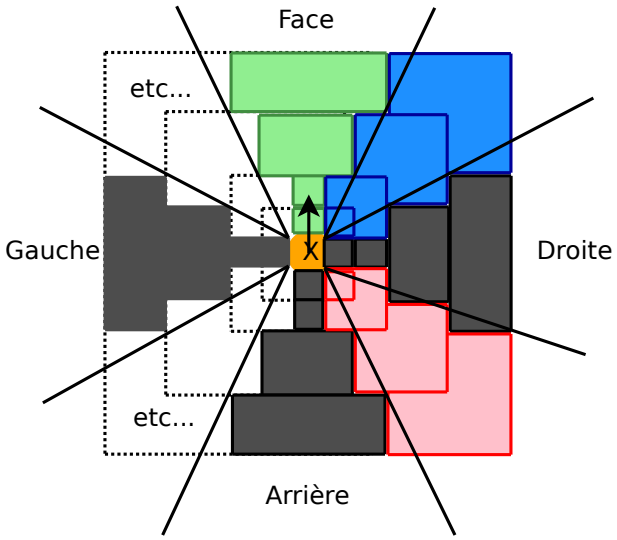


FIGURE 6.7 – Notion de relation appliquée au problème de Thales (3)

On a ici $8k+1$ relations possibles, avec $k$ le nombre de couches considérées (le $+1$ venant de la relation particulière `collision`, lorsque l'agent se trouve sur la même case qu'un de ses voisins).

Plus la valeur de $k$ est grande, et meilleur sera le comportement de l'agent (celui-ci pouvant prendre en compte un plus grand nombre de voisins à la fois). Cette valeur sera cependant limitée par deux critères : la précision des capteurs (l'agent ne pouvant tout simplement pas observer ses voisins au delà d'une certaine distance), et sa capacité de calcul (augmenter $k$ impliquant plus d'états relatifs, et donc plus de situations à traiter dans la phase de planification).

On peut également remarquer le nombre d'états concernés par une relation donnée, qui augmente lorsque l'on s'éloigne de l'agent. Cette structure particulière vient du fait que les capteurs perdent en général en précision à mesure de l'éloignement. On aurait, en effet, pu introduire de nouvelles relations au fur et à mesure de l'écartement (par exemple : face/face-face-droite/face-droite/droite/arrière-droite/arrière-arrière-droite/arrière/etc.), afin de maintenir le même nombre d'états au sein de chaque relation, mais cela aurait eu peu de sens. Les capteurs perdant en précision à mesure que l'on s'éloigne, il y aurait eu une grosse incertitude sur les observations, entre face-face-droite et face-droite par exemple.

Cette répartition est de plus logique d'un point de vue calculatoire : il est important d'être précis sur la position des agents proches, ceux-ci étant en interaction immédiate (à un ou deux pas de temps). Les agents éloignés au contraire, n'auront d'influence sur l'agent qu'après plusieurs pas de temps. On peut donc se permettre d'être moins précis sur leur représentation, puisque l'on aura le temps d'affiner celle-ci lorsqu'ils se rapprocheront.

### 6.3.2 Manipulation des états joints relatifs dans le problème de Thalès

Afin de clore ce chapitre, voyons comment instancier la composante d'interaction dans le problème de Thales. Nous avons précédemment énuméré l'ensemble des interactions mises en jeu. Il ne reste donc qu'à créer les états relatifs associés à ces interactions, puis à définir les fonctions d'observation, de transition et de récompense sur les états joints relatifs. Afin de définir la fonction d'observation, il faut prendre en compte les propriétés et restrictions des capteurs embarqués par l'agent :

- l'agent dispose, grâce à ses capteurs, d'une vision à 360° (il peut donc observer, à tout moment, l'ensemble de son voisinage),
- les capteurs souffrent d'une imprécision qui augmente lorsque la cible observée s'éloigne de l'agent (l'état relatif d'un voisin proche sera connu à 100%, tandis que l'état relatif d'un agent éloigné sera soumis à une forte incertitude),
- la distance d'observabilité est bornée par une portée maximale,
- les capteurs ne permettent pas de « voir au travers » des obstacles (ainsi, des agents peuvent en cacher d'autres).

La figure 6.8 présente un exemple d'observation de l'état joint relatif, sur le problème de Thales, avec un agent entouré de 4 voisins évoluant en formation « dispersée ». Un des voisins est hors portée des capteurs, et sera donc absent de l'état joint relatif. Les trois autres voisins sont observés, deux dans la couche 2 et un dans la couche 1 (voir l'énumération des interactions, précédemment). L'état joint relatif `(arrière2,arrière-droite1,face-gauche2)` décrit ce voi-

sinage, éventuellement soumis à une incertitude. L'état `(arrière2,droite1,face-gauche2)` par exemple est possible (au vu de l'imprécision des capteurs), avec une faible probabilité.

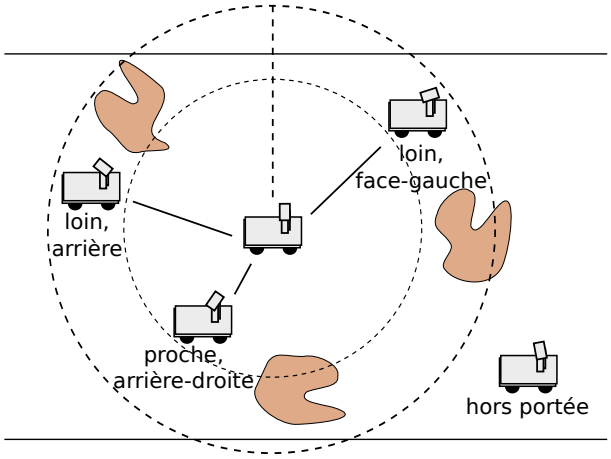


FIGURE 6.8 – Exemple d'observation de l'état joint relatif

Si on avait choisi de représenter ce problème via un DEC-POMDP « classique », l'état observé pour cet exemple serait l'état joint $([x;y],[x-2;y],[x-1;y-1],[x+2;y+2])$, c'est-à-dire une description des positions de chaque agent (avec $[x;y]$ la position de l'agent central, suivie des positions de chaque voisins). Ainsi, pour chaque position $[x;y]$ possible, on aurait un état joint différent décrivant cette interaction (l'état $([2;3],[0;3],[1;2],[4;5])$ par exemple, serait différent de l'état $([3;7],[1;7],[2;6],[5;9])$, alors que ces deux états joints décrivent les mêmes relations entre agents). La représentation sous forme d'états joints relatifs permet donc de compresser considérablement l'espace d'états, simplifiant ainsi la complexité du processus de résolution.

On peut ensuite définir une fonction de transition entre les états joints relatifs. Ici, il faudra distinguer 4 cas :

1. si l'état joint relatif contient la relation `collision`, l'agent et le voisin concernés sont « propulsés » en arrière (probabilité 1),
2. des agents jusque là non-observés peuvent intégrer l'interaction : on suppose pour cela une distribution uniforme de ces agents sur l'ensemble des états non-observés,
3. les agents présents peuvent quitter l'interaction (en transitant vers un état hors portée des capteurs), la probabilité de cet événement étant calculée via leurs fonctions de transition individuelles,
4. les autres voisins se déplacent relativement à l'agent, là encore selon leurs fonctions de transition individuelles.

Ces probabilités dépendent non-seulement des états des voisins, mais aussi de leurs actions : cet aspect sera traité dans la section algorithmique. Il ne reste plus qu'à définir la fonction de récompense associée : celle-ci attribue une pénalité de $n \times c$ à l'agent lorsqu'il entre en interaction avec $n$ voisins en même temps ($c$ étant le coût engendré par une collision), ainsi qu'une récompense aux états joints relatifs tels que la formation voulue est maintenue.

## Conclusion

Ce chapitre nous a permis d'identifier trois points centraux dans notre approche : décomposer le problème de décision en un ensemble de sous-problèmes semi-indépendants, séparer l'aspect « coordination » (les interactions) de la composante individuelle (la tâche) et observer les interactions de l'agent via les états relatifs de ses voisins. Ces points nous permettront de casser la complexité combinatoire du problème multiagent, afin de traiter des instances de grande taille (intraitables via les approches existantes, en raison justement de cette complexité combinatoire).

Le chapitre suivant propose une description du modèle en lui-même : nous verrons donc comment retranscrire directement les éléments de ce chapitre. Ainsi, nous verrons comment formaliser séparément la tâche de l'agent et ses interactions. Nous verrons ensuite comment indiquer, au sein de cette composante d'interaction, l'existence de plusieurs sous-problèmes. Nous définirons également, toujours au sein de cette composante d'interaction, l'ensemble des états relatifs possibles. Nous utiliserons alors ceux-ci pour décrire l'influence des voisins sur l'agent.

# Chapitre 7

# Cadre formel

**Sommaire**



Ce chapitre introduit le modèle DyLIM [24], pour la représentation de problèmes de planification multiagent sous-incertitude en environnement partiellement observable (problèmes de type DEC-POMDP). L'idée est de formaliser (entre autres) l'ensemble des interactions possibles entre agents, afin que celles-ci puissent être exploitées durant le processus de résolution du problème. On parlera ici d'interaction au sens large : l'exemple donné du problème de Thales montre des interactions ≪ géographiques ≫, mais celles-ci pourront être d'un tout autre type. Seront en interaction deux agents tels que l'un pourra avoir une influence sur l'autre. Nous commencerons par décrire le modèle en lui-même, puis nous nous attarderons sur la description des interactions. Nous donnerons finalement deux variantes pour ce modèle, selon que l'on considère une représentation exacte ou approchée des interactions.

## 7.1 Le modèle : DyLIM

Cette section introduit le modèle DyLIM, utilisé pour la représentation de problèmes de type DEC-POMDP. Ce modèle propose une représentation dynamique des interactions locales entre agents, afin de permettre leur utilisation durant le processus de résolution. Il existe, au sein de

24. DyLIM, de l'anglais : *Dynamic Local Interaction Model*

la littérature, plusieurs approches visant à voir le problème multiagent comme un ensemble de problèmes monoagents [Gmytrasiewicz et Doshi, 2005, Capitán *et al.*, 2011]. Ici, nous poussons plus loin ce raisonnement en ajoutant au problème individuel un second problème, décrivant l'influence du voisinage sur l'agent.

**Définition 31 (DyLIM)** *DyLIM est un modèle pour la représentation de problèmes de planification multiagents, sous-incertitude et en environnement partiellement observable. On représente ces problèmes par un couple* $< Pb^{ind}, Pb^{coo} >$*, avec* $Pb^{ind}$ *un problème individuel, représentant la tâche de l'agent, et* $Pb^{coo}$ *un problème de coordination, représentant ses interactions.* $Pb^{ind}$ *et* $Pb^{coo}$ *forment alors les deux* ***composantes*** *du problème global.*

Nous présenterons donc ces deux aspects et montrerons comment on peut les formaliser, indépendamment l'un de l'autre.

### 7.1.1 Composante individuelle

On formalise la tâche de l'agent via un POMDP $\langle S,A,T,R,\Omega,O\rangle$. $S$ désigne l'ensemble des états (individuels) dans lesquels l'agent peut se trouver, $A$ désigne les actions qu'il peut accomplir, $T : S \times A \times S \to [0;1]$ est une fonction de transition entre ces états et $R : S \times A \to \mathbb{R}$ est la fonction de récompense associée. Ce modèle est destiné aux environnements partiellement observables. Ainsi, l'ensemble $\Omega$ désigne les observations que l'agent peut effectuer sur son état et la fonction $O : S \times A \times S \times \Omega \to [0;1]$ donne la probabilité d'effectuer ces observations.

Ce POMDP décrit le problème de l'agent seul, on ignore donc pour l'instant l'existence des voisins. Sur le problème de Thales par exemple, un état décrira la position $(x,y)$ de l'agent, ainsi éventuellement que son orientation. Les transitions décriront alors la probabilité qu'a l'agent de se déplacer d'un état à l'autre, selon ses actions. Sur le problème de Thales, cette fonction décrira donc la probabilité qu'a l'agent d'avancer, de tourner, ou encore d'entrer en collision avec les obstacles présents dans l'environnement. De telles transitions peuvent toutefois s'avérer erronées. Imaginons que l'agent ait un voisin en face de lui : si il avance, il va entrer en collision avec ce voisin. La probabilité *réelle* de se déplacer vers l'avant est donc nulle. Pourtant, si on se fie à la fonction de transition individuelle, l'agent considérera qu'il peut avancer sans problème. Il y aura donc, parfois, décalage entre la réalité et ce modèle individuel (d'où l'utilité de la composante d'interaction, voir section 7.1.2).

De la même façon, la fonction de récompense individuelle indiquera à l'agent la récompense associée à chaque transition, lorsque l'on ignore l'existence des voisins. Sur le problème de Thales par exemple, l'agent recevra une récompense positive en atteignant son objectif, un coût faible pour chaque déplacement et une pénalité élevée s'il entre en collision avec un obstacle. Là encore, ces récompenses pourront être faussées : on pourra considérer qu'une action est intéressante, et apporte une récompense positive, alors qu'en réalité elle va provoquer une rupture du convoi. Ce problème individuel permettra donc à l'agent de calculer une politique qui parfois sera optimale, et parfois devra être pondérée par le problème de coordination.

Pour finir, la fonction d'observation sera elle-aussi purement individuelle : l'agent observera par exemple les obstacles autour de lui, afin d'en déduire sa position, mais ignorera les observations concernant les agents voisins. On obtient alors un problème de dimension raisonnable, l'explosion combinatoire due à l'aspect multiagent ayant disparue. Il va ensuite falloir, au sein de la composante d'interaction, veiller à ne pas (trop) réintroduire cette complexité.

### 7.1.2 Composante d'interaction

Tout comme on formalise la tâche de l'agent par un POMDP, on va formaliser ses interactions comme un problème de décision à part entière. Ce problème permettra donc à l'agent de prendre une décision en fonction de son voisinage, indépendamment de son objectif individuel.

On formalise le problème d'interaction par un n-uplet $\langle SR,\Omega R,OR,C\rangle$, tel que :

- $SR$ désigne l'ensemble des états relatifs possibles. Afin de connaître les instances associées à chaque état relatif, on aura en général $SR = \{[sr^1 : inst(sr^1)], \ldots, [sr^{|SR|} : inst(sr^{|SR|})]\}$ avec $inst(sr^i)$ l'ensemble des instances possibles pour $sr^i$,
- $\Omega R$ est un ensemble d'observations que l'agent peut recevoir au sujet de son voisinage (sur Thales par exemple, l'agent pourrait observer un voisin en face, à droite, à gauche, etc.),
- $OR : \bigcup_{i=0}^{m} SR^{(i)} \times \Omega R \to [0;1]$ est la fonction associant à tout état joint relatif concernant $i$ interactions, $0 \leq i \leq m$, une distribution de probabilités sur les observations possibles. Ici, on note $X^{(i)}$ l'ensemble $X$ porté à la puissance $i$, pour différencier cette notion de $X^i$ où $i$ est un exposant quelconque (comme dans $S = \{s^1,s^2,\ldots\}$ par exemple),
- $C$ est un ensemble de clusters d'interactions (voir ci-dessous).

Les éléments $SR$, $\Omega R$ et $OR$ permettent donc à l'agent d'observer son voisinage, et d'en déduire l'état joint relatif dans lequel il se trouve. On a choisi ici d'observer uniquement l'état joint relatif en cours, mais on pourrait également définir la fonction d'observation $OR$ sur les couples $(sr,a)$ (avec $a$ une action individuelle ou jointe), voir directement sur les transitions $(sr,a,sr')$. L'ensemble des états joints relatifs est donc noté $\bigcup_{i=0}^{m} SR^{(i)}$, avec $m$ la taille de la plus grande interaction possible (c'est à dire le nombre maximum d'interactions simultanées) : cet ensemble contient donc tous les états joints relatifs décrivant 0, 1, 2, ... ou $m$ interactions simultanées. On verra par la suite (partie III, algorithmes) que l'on peut borner $m$, afin de limiter la complexité du problème.

Il reste donc à définir $C$. Cet élément décrit un ensemble de clusters d'interactions, chaque cluster correspondant à un sous-problème donné de la composante d'interaction. Dans le problème de Thales par exemple, on aura $C = \{C^0,C^1,\ldots,C^m\}$ avec $C^1$ le cluster associé aux états joints relatifs décrivant 1 interaction, $C^2$ pour les états joints relatifs à 2 interactions, etc. On notera les cas particuliers de $C^m$, associé au nombre max d'interactions simultanées possibles, et $C^0$ associé à l'état « sans interaction ». Il est important de noter que la façon dont on a choisi les clusters d'interactions dans le problème de Thales n'est pas la seule approche possible : on aurait également pu imaginer un cluster par type de convoi par exemple, indépendamment du nombre d'agents impliqués.

## 7.2 Notion de cluster d'interactions

Nous avons cité, dans la section précédente, l'ensemble de clusters d'interactions $C = \{C^1,\ldots,C^{|C|}\}$, chaque cluster décrivant un sous-problème de la composante d'interaction. Voyons maintenant la définition formelle d'un cluster d'interactions $C^i$.

### 7.2.1 Cluster d'interactions : définition

**Définition 32 (Cluster d'interactions)** *Un cluster d'interactions $C^i$ est défini par un triplet $C^i = (S^i,T^i,R^i)$, dans lequel :*

- $S^i = \{sr^1,\ldots,sr^{|S^i|}\}$ *est l'ensemble des états joints relatifs concernés par ce cluster,*
- $T^i : S^i \times A \times (S^i \cup C) \rightarrow [0;1]$ *est une fonction de transition propre au cluster,*
- $R^i : S^i \times A \times S^i \rightarrow [0;1]$ *est la fonction de récompense associée.*

*On peut traiter chaque cluster d'interactions indépendamment des autres, comme un sous-problème de la composante d'interaction.*

#### Raisonner sur les actions individuelles

Un cluster va donc définir un problème de décision, que l'on pourra résoudre. Deux choix s'offrent à nous ici :

1. soit $A$ représente les actions jointes de l'agent et de ses voisins en interaction,
2. soit $A$ représente les actions individuelles de l'agent.

La deuxième solution a l'avantage de permettre une résolution monoagent, mais suppose que les transitions ne dépendent pas de l'action des voisins, ce qui est (la plupart du temps) faux. La première solution semble donc s'imposer, mais implique de calculer une politique jointe, ce qui est en général trop difficile dans les problèmes traités. Ne pourrait-on pas, alors, calculer une politique individuelle localement optimale pour l'agent (afin de rester dans notre optique, de calculer des « bonnes » politiques en un temps raisonnable, plutôt que de viser l'optimal). On peut, pour cela, faire des suppositions sur le comportement des voisins de l'agent. On pose $A$ l'ensemble des actions individuelles et $A^{(|rs|)}$ l'ensemble des actions jointes pour les voisins impliqués dans l'état joint relatif $rs$, puis on calcule :

$$\forall rs,rs' \in S^i, \forall a \in A, T^i(rs,a,rs') = op_{a_v \in A^{(|rs|)}} T(rs,(a,a_v),rs')$$

L'opérateur *op* sera donc un opérateur de choix sur les actions de voisinage $a_v \in A^{(|rs|)}$ (max, min ou moyenne par exemple), qui dépendra du problème que l'on traite (veut-on être optimiste, pessimiste, etc). Une seule contrainte ici : il faut s'assurer que la fonction de transition est cohérente, c'est-à-dire vérifier $\forall rs \in S^i, \forall a \in A, \sum_{rs' \in S^i} T(rs,a,rs') = 1$. Cette question sera, là encore, développée dans la partie III (algorithmes). On définira donc (grâce à cette méthode) les fonctions $T^i$ et $R^i$ sur l'ensemble des actions individuelles $A$.

**Le problème de l'expressivité**

Le problème de coordination, tel qu'il est représenté ici, ne permet pas de remettre en question le modèle individuel. Prenons l'exemple d'un robot, ayant un voisin (immobile) en face de lui. Si le robot avance, il va entrer en collision avec son voisin : que se passe t'il alors ? Mettons que la collision immobilise les deux robots, et voyons comment on peut représenter cette situation avec notre modèle.

1. le problème individuel décrit la façon dont le robot se déplace : avancer, par exemple, amènera le robot sur l'emplacement en face de lui dans 90% des cas, et le maintiendra sur place (échec de l'action) dans 10% des cas,
2. le problème de coordination décrit l'évolution des relations entre l'agent et ses voisins : si le robot avance, dans l'état relatif `face` (signifiant qu'il a un voisin en face de lui), il aboutira systématiquement dans l'état relatif `collision`.

Ainsi, si on fait la « somme » de ces deux problèmes, on en déduit que le robot ayant un voisin en face de lui, va soit avancer ($P = 0{,}9$), soit rester sur place ($P = 0{,}1$), le tout en créant une collision ($P = 1$). Il est par contre impossible de représenter le fait que la collision va immobiliser le robot, car cela signifierait remettre en question le modèle individuel. Il semble donc qu'un problème d'expressivité se pose. En réalité, ce problème n'en est pas un, car on peut le contourner en filtrant les actions applicables. Dans notre cas, imaginons que seule l'action `reculer` soit applicable dans l'état relatif `collision`. On modélise alors bien le fait que cette collision « immobilise » l'agent. En ajoutant une pénalité forte à cet état relatif, on a modélisé le problème initial.

**Évolution inter et intra clusters**

Nous avions précédemment vu que les sous-problèmes de la composante d'interaction étaient semi-indépendants, l'agent pouvant transiter d'un sous-problème à l'autre. Cet aspect s'exprime dans la fonction de transition, qui donne la probabilité de passer d'un état de $S^i$ vers un état de $(S^i \cup C)$. Cela signifie que l'agent peut transiter :

– soit vers un nouvel état de $S^i$,
– soit vers un autre cluster $C^j \in C$.

Dans ce second cas, l'agent transitera dans *un des états* de $C^j$. On ne définit pas ici vers quel état l'agent transite, mais uniquement le changement de cluster. On casse ainsi la complexité combinatoire, en remplaçant le problème global par un ensemble de sous-problèmes faiblement couplés. De plus, l'évolution au sein du cluster $C^j$ n'étant pas du ressort du problème que l'on résout actuellement (évoluer au sein de $C^i$), on considère que l'état $C^j$ représente une « sortie » du problème actuel. On symbolisera cette sortie par un état puits ($\forall sr \in S^i, \forall a \in A, T^i(C^j,a,sr) = 0$ et $\forall a \in A, T^i(C^j,a,C^j) = 1$). Nous verrons dans la partie III comment calculer la valeur de cet état particulier (qui, elle-même, prendra en compte les probabilités de transition vers d'autres clusters $C^k$). Ainsi, l'agent pourra calculer une politique propre à son cluster $C^i$

actuel, lui permettant éventuellement de se diriger vers un autre cluster $C^j$, plus intéressant. Une fois ce changement de cluster effectué, l'agent se mettra à suivre la politique propre à $C^j$.

### 7.2.2 Définir un « bon » ensemble de clusters

Définir l'ensemble des clusters d'interactions se fait nécessairement en deux étapes, que nous exposons dans cette section.

#### Déterminer l'ensemble des sous-problèmes de la composante d'interaction

Il faut commencer par déterminer l'ensemble des sous-problèmes. Pour cela, se baser sur la connectivité des états semble une bonne approche. En effet, le fait de résoudre séparément chaque cluster repose sur l'hypothèse que ceux-ci sont semi-indépendants : il faut donc minimiser le nombre de transitions inter-clusters. Ainsi, si le découpage en clusters est de bonne qualité, les états au sein d'un même cluster seront fortement couplés tandis que les états appartenant à deux clusters différents seront faiblement couplés.

Sur le problème de Thales par exemple, l'arrivée ou le départ d'un agent dans le convoi est un événement relativement rare, d'où le fait de baser la décomposition sur ce critère (un cluster par « taille » d'états joints relatifs). Quoi qu'il en soit, tout état joint relatif doit appartenir à exactement un cluster. Cela se traduit par l'équation :

$$\left(\bigcup_{i=0}^{m} SR^{(i)}\right) = \left(\bigcup_{C^i \in C} S^i\right)$$

Ainsi, lorsque l'agent observe son état joint relatif, il peut toujours identifier le cluster associé (de façon plus ou moins certaine selon l'incertitude sur les observations, bien entendu). On n'oubliera pas d'inclure le cas particulier $sr = \emptyset$, où l'agent n'est soumis à aucune interaction. En général, ce cas sera associé à un cluster dans lequel l'agent se contentera d'exécuter sa politique individuelle (celui-ci n'étant soumis à aucune interaction, toute action est équivalente d'un point de vue coordination).

#### Identifier les dépendances entre agents au sein de chaque sous-problème

Une fois cette division du problème effectuée, on dispose des ensembles $S^1, \ldots, S^m$ associés aux $m$ clusters d'interactions. Il faut alors définir les fonctions $T^i$ et $R^i$ correspondantes à ces clusters. Définissons, par exemple, la fonction de transition $T^i$. Pour un triplet $(sr,a,sr')$ donné, deux situations peuvent se poser :

1. les agents sont indépendants pour cette transition : la probabilité peut alors être obtenue par un produit des fonctions de transition individuelles,
2. il y a dépendance entre l'agent et ses voisins : la probabilité de transition $T^i(sr,a,sr')$ ne pourra pas être calculée et doit donc être fournie lors de la définition du problème.

Il en va bien entendu de même pour les récompenses. Prenons à nouveau l'exemple de Thales : la probabilité de transition de l'agent dépendra des voisins les plus proches, puisque ceux-ci pourront entraîner des collisions. Les voisins un peu plus éloignés par contre, seront indépendants du point de vue de la transition, mais seront tout de même inclus dans l'interaction puisque leur état relatif influera sur la récompense (convoi bien formé, ou non). Ainsi, certains états impliqueront une dépendance sur les transitions, d'autres sur les récompenses (voir sur les deux).

Il suffira donc, lors de la définition du problème, d'exprimer les valeurs de $T^i$ et de $R^i$ lorsqu'il y a dépendance. Les valeurs indépendantes quant à elles, seront calculées à partir des fonctions de transition individuelles, lors de la résolution du problème. Pour finir, nous avons vu précédemment que ces transitions et récompenses pouvaient être exprimées en fonction des actions individuelles de l'agent, via une approximation sur les actions jointes (max, min ou moyenne par exemple). Il faudra alors choisir l'opérateur à appliquer sur les actions jointes, selon le problème que l'on traite. Est-on :

1. *optimiste* - On prend le risque d'exécuter une action jointe même si on n'est pas sur d'être suivi (typiquement, sur un problème d'agents déménageurs : au pire, si on tente de soulever une caisse lourde et que l'autre agent ne nous aide pas, on passe à une autre caisse). Cet opérateur aura l'avantage de privilégier les interactions, mais entraînera parfois des échecs dans le ≪ comportement de groupe ≫.

2. *pessimiste* - On n'exécute l'action jointe que si on est sur d'être suivi (sur le problème de Thales par exemple, il faut être certain que l'on ne va pas créer de collision). Cet opérateur au contraire, va limiter les interactions : l'agent tirera moins parti du groupe auquel il appartient, mais se mettra moins souvent en situation d'échec.

3. *en moyenne* - On exécute les actions jointes en acceptant de se tromper, mais pas trop souvent (s'applique aux problèmes où c'est le résultat final qui importe). Cet opérateur est un intermédiaire, entre l'optimisme et le pessimisme.

La définition du problème d'interaction sera donc partielle, puisque l'on fournira uniquement les transitions et récompenses lorsqu'il y a dépendance entre agents. Les autre valeurs seront calculées lors de la phase de résolution.

## 7.3 Types d'interactions envisageables

Nous avons, dans le chapitre précédent, envisagé de façon intuitive l'usage d'interactions approchées. Nous avons par exemple cité le problème des états relatifs pouvant concerner ≪ exactement un agent ≫, ou bien ≪ au moins un agent ≫, introduisant une part d'incertitude dans le traitement des interactions. Cette section propose une analyse formelle de cet aspect, via deux types d'interactions : exactes, ou approchées.

### 7.3.1 Interactions exactes

Les interactions exactes permettent une description précise du problème, l'état joint relatif fournissant autant d'information qu'un état joint « classique ». En contrepartie, cette approche peut réintroduire l'explosion combinatoire dans le problème de décision.

#### Concept d'interaction exacte

On considère que les interactions manipulées sont exactes si et seulement si celles-ci vérifient les trois conditions suivantes :

1. au sein de l'état joint relatif, un état relatif décrit une interaction avec exactement un voisin. Cela implique donc que l'état `(face,face)` par exemple, est différent de l'état `(face)` (le premier décrit une interaction avec deux agents, le second avec un seul agent).
2. Si on connaît l'état $s_i$ de l'agent, et l'état relatif $sr_j$ d'un voisin, on peut en déduire l'état $s_j$ de ce voisin. Cela signifie que l'ensemble des instances ($s_a$,$s_b$) de $sr_j$ ne contient qu'un seul élément pour lequel $s_a = s_i$.
3. L'état joint relatif décrit l'ensemble des interactions à un moment donné (on ne peut pas choisir d'ignorer certaines des interactions).

Ces interactions sont donc exactes, dans le sens où on décrit la totalité des informations dont on dispose sur le voisinage de l'agent. Si une de ces conditions n'est pas vérifiée, on considère que les interactions sont approchées. Utiliser des interactions exactes est intéressant pour obtenir des politiques de bonne qualité, mais implique parfois un coût important en termes de complexité.

#### Avantages et inconvénients des interactions exactes

Les interactions exactes ont l'avantage de permettre une description de l'ensemble des informations dont dispose l'agent quant à son voisinage :

- on connaît, pour commencer, le nombre exacte d'agents en interaction,
- on peut de plus retrouver, pour chaque voisin $j$, son état réel $s_j$ (à partir de l'état de l'agent et de l'état relatif du voisin en question),
- pour finir, on sait que les agents n'étant pas en interaction n'ont (par définition) aucune influence sur les transitions, ni sur les récompenses.

Il est donc possible, à tout moment, de retrouver l'état joint réel et de baser la décision sur celui-ci. Pourtant, la dimension du problème à manipuler est largement inférieure à celle d'une représentation « classique ». Il y a beaucoup moins d'états joints relatifs par exemple, que d'états joints « réels ». Il est, pour finir, simple de manipuler cette représentation (les transitions par exemple, lorsqu'elles ne sont pas une donnée du problème, peuvent être facilement calculées comme un produit des fonctions de transition individuelles).

Pourtant, cette approche pose plusieurs problèmes. Tout d'abord, même si la dimension du problème est réduite par l'usage des états relatifs, l'explosion combinatoire est toujours présente.

On remplace effectivement les $S^I$ états joints possibles par $k$ états joints relatifs. Il y a l'état vide, plus les états à un voisins, plus ..., plus les états à $(I-1)$ voisins, soit :

$$k = \sum_{i=0}^{I-1} |SR|^i = \frac{1-|SR|^I}{1-|SR|}$$

Le nombre d'états relatifs dans $SR$ est (en général) petit, mais l'exponentielle en le nombre d'agents est malgré tout toujours présente. Pour finir, même en admettant que l'on parvienne à traiter un tel ensemble d'états, cette représentation ne sera pas toujours réaliste du point de vue de l'observabilité. Nous avons décrit, dans le chapitre précédent, comment les agents pouvaient se cacher mutuellement. Peut-on, dans ce cas, garantir que l'on parviendra à observer l'ensemble du voisinage ? Nous avons donc étudié comment le modèle pouvait être « assoupli », en autorisant l'usage d'interactions approchées.

### 7.3.2 Interactions approchées

Les interactions approchées permettent de borner l'explosion combinatoire du problème, mais peuvent diminuer la valeur des politiques calculées.

#### Différences entre les interactions exactes et approchées

Nous avons donné précédemment la définition d'une interaction exacte, en précisant qu'une telle interaction devait vérifier trois conditions. Ainsi par opposition, une interaction approchée permettra trois types d'incertitude sur le voisinage :

1. une relation peut impliquer plusieurs agents en même temps. Un état relatif décrira donc l'interaction entre l'agent et **au moins un** de ses voisins. Imaginons, par exemple, que l'agent ait trois voisins devant lui et deux sur sa gauche, on considérera que l'état joint relatif est `(face, gauche)` au lieu de `(face,face,face,gauche,gauche)`.
2. Soit $i$ l'agent, et $j$ un de ses voisins : plusieurs instances de l'état relatif $sr_j$ pourront correspondre à l'état $s_i$. Il sera donc impossible de connaître avec certitude l'état $s_j$. Si par exemple l'agent $i$ se trouve aux coordonnées $s_i = (2,3)$ et que l'état relatif de l'agent $j$ est $sr_j =$ (`nord`), alors les états $s_j = (2,4)$ ou $s_j = (2,5)$ seront deux instances admissibles de $sr_j$, au vu de l'état $s_i$. On ne pourra donc par connaître avec certitude l'état $s_j$.
3. On suppose un ordre de préférence sur les relations, et un nombre maximum $m$ de relations considérées au même moment. Par exemple, avec $m = 2$ et l'ordre $face > gauche > derriere$, l'état joint relatif `(face,derrière,gauche)` sera réduit à `(face,gauche)`.

Le point le plus important ici est la possibilité que l'on a de borner le nombre de relations considérées à un moment donné, stoppant ainsi l'explosion combinatoire au delà d'un point fixe (choisi, logiquement, selon les capacités de calcul dont on dispose).

**Définition d'un ordre de préférence sur les interactions**

Les interactions approchées permettent de ne considérer que les $m$ premiers états relatifs, selon un ordre de préférence. Cet aspect est inspiré du comportement qu'adopte un être humain, lorsqu'il évolue au sein d'une foule. Idéalement, cet humain devrait prendre en compte l'intégralité des personnes présentes, estimer l'ensemble de leur trajectoires possibles, et choisir ainsi un chemin optimal. Dans les faits, il est impossible de prendre autant d'éléments en considération, et l'humain se contente d'observer les gens qui l'entourent et d'estimer leurs déplacements, afin de ne pas créer de collision. De la même manière, considérer l'ensemble des agents impliqués dans le problème étant très complexe, nous nous contenterons de prendre en compte les interactions les plus critiques. On choisira alors de prendre en compte, au même moment, un maximum de $m$ interactions, cette valeur dépendant directement des capacités de calcul dont on dispose.

On va donc, grâce aux interactions approchées, diminuer le nombre d'états joints relatifs possibles. On avait, jusqu'à présent, $\frac{1-|SR|^I}{1-|SR|}$ états possibles (avec les interactions exactes). On va désormais considérer $i$ interactions simultanés, $0 \leq i \leq m$, ceci via un tirage de $i$ éléments parmi les $|SR|$ interactions possibles, le tout sans remise (puisqu'une interaction partagée par plus de 1 agent ne sera prise en compte qu'une seule fois). Le nombre total d'états joints relatifs possibles est donc de :

$$\sum_{i=0}^{m} \binom{|SR|}{i} = \sum_{i=0}^{m} \frac{|SR|!}{i!(|SR|-i)!}$$

Ainsi à titre d'exemple, sur un problème impliquant 10 agents, 15 états relatifs possibles et en bornant la taille de l'interaction à $m = 4$, on passe de 41 milliards d'états joints relatifs à 1941. Sur le problème de Thales (en supposant des agents orientés, pouvant uniquement `avancer`, `attendre` ou `tourner`), les interactions les plus critiques sont celles concernant les agents présents dans la première couche (pour rappel, on passe d'une couche à l'autre à mesure que l'on s'éloigne de l'agent). Parmi ces interactions, les voisins devant l'agent sont prioritaires, suivis de ceux sur les côtés, les moins importants étant ceux derrière l'agent. Ainsi, en notant `face2` par exemple la relation « face » dans la $2^{eme}$ couche, on donne l'ordre suivant (en partant de la relation la plus critique) : `Collision`, `face1`, `angleFaceGauche1`, `angleFaceDroite1`, `gauche1`, `droite1`, `angleArrièreGauche1`, `angleArrièreDroite1`, `arrière1`, `face2`, `angleFaceGauche2`, etc.

On a donc $|SR| = 1 + 8l$, avec $l$ le nombre de couches considérées. Dans ce problème, poser $m = 4$ semble raisonnable pour pouvoir considérer au moins les trois cases en face de l'agent en plus de la case sur laquelle il se situe. Si on suppose la présence de 10 agents par exemple, et que l'on observe $l = 4$ couches, on aura $|SR| = 33$ d'où 46938 états joints relatifs possibles, contre 46 billions avec des interactions exactes. On peut également envisager l'emploi d'un ordre conditionnel sur les interactions (par exemple, rendre `face2` critique si `face1` n'est pas observé) : cela ne changera rien aux algorithmes employés, mais la définition d'un tel ordre sera plus dur à réaliser qu'une simple énumération des relations possibles. Le tableau 7.1 indique le nombre d'états à manipuler dans une instance donnée du problème de Thales (grille $10 \times 10$ avec

$I = 10$ agents), selon que l'on utilise des états joints, des interactions exactes ou des interactions approchées. On constate ainsi l'intérêt de ces différentes approches, pour diminuer la taille du problème à manipuler.

| # couches observées | états joints « classiques » | interactions exactes | approchées ($m = 3$) | approchées ($m = 4$) | approchées ($m = 5$) |
|---|---|---|---|---|---|
| $l = 1$ ($\|SR\| = 9$) | $10^{20}$ | $10^{8}$ | 130 | 256 | 382 |
| $l = 2$ ($\|SR\| = 17$) | $10^{20}$ | $10^{11}$ | 834 | 3 214 | 9 402 |
| $l = 3$ ($\|SR\| = 25$) | $10^{20}$ | $10^{12}$ | 2 626 | 15 276 | 68 406 |
| $l = 4$ ($\|SR\| = 33$) | $10^{20}$ | $10^{13}$ | 6 018 | 46 938 | 284 274 |
| $l = 5$ ($\|SR\| = 41$) | $10^{20}$ | $10^{14}$ | 11 522 | 112 792 | 862 190 |

TABLE 7.1 – Nombre d'états, problème de Thales sur une grille $10 \times 10$ avec $I = 10$ agents.

**Avantages et inconvénients des interactions approchées**

Utiliser des interactions approchées permet, comme nous avons pu le constater, de borner le nombre maximal d'interactions considérées simultanément. On limite ainsi le nombre d'états joints relatifs possible, ce qui nous permet de casser l'explosion combinatoire du problème. L'avantage de cette solution est qu'elle permet de ne pas « trop » remettre en question la possibilité de calculer de bonnes politiques. En effet, le fait de regrouper les agents par interaction permet de maximiser le nombre de voisins pris en compte, et l'ordre sur ces interactions permet de garantir que l'on prendra toujours en compte, au minimum, les états relatifs critiques. Cette approche est, de plus, réaliste au vu des contraintes d'observabilité. Comme nous l'avons précédemment montré sur le problème de Thales, il est courant que des interactions en « cachent » d'autres. Utiliser des interactions approchées signifie entre autres prendre en compte cette possibilité, qu'une partie des interactions soient absentes de l'état joint relatif. Pour autant, cette approche souffre également de certains inconvénients.

Il existe des situations dans lesquelles le nombre d'agents présents a un impact direct sur les récompenses obtenues. Prenons le problème de Thales : entrer en collision avec 1 seul agent sera bien moins coûteux qu'une collision avec, par exemple, un groupe de 10 agents. Dans ce cas, imaginons qu'un agent utilise des interactions approchées et perçoive l'état relatif `angleFaceDroite1` : on peux imaginer que cet agent tentera malgré tout d'avancer, en espérant éviter la collision. Imaginons maintenant le même agent, utilisant cette fois des interactions exactes, et qui constate que 5 agents sont présents dans cet angle (`angleFaceDroite1`) : l'agent

évitera probablement d'avancer, la multiplication des voisins augmentant considérablement le risque d'entrer en collision avec au moins l'un d'eux. Les politiques calculées seront donc, inévitablement, moins précises avec des interactions approchées : le tout sera alors de trouver le juste milieu, entre exactitude des interactions et taille de l'espace d'états engendré. Nous verrons par exemple que l'on peut dire d'une interaction qu'elle concerne non pas $n > 0$ voisins, mais $0 < n < k$ voisins, avec $k$ une valeur dépendant de l'interaction en question.

De la même façon, il sera plus difficile de calculer les transitions lorsque l'on utilise des interactions approchées. Dans le cas exacte, lorsque les transitions ne sont pas fournies, on effectue un produit des fonctions individuelles. Comment, lorsque l'on ignore le nombre d'agents impliqués et leurs états exacts, appliquer ces fonctions individuelles ? Nous verrons dans la partie III que ce problème peut être traité mais n'est pas trivial.

## Conclusion

Ce chapitre nous a permis de présenter, formellement, notre modèle. Nous avons donc montré comment décrire séparément la tâche de l'agent, sous la forme d'un problème de décision individuel (plus précisément, un POMDP), et ses interactions, là aussi sous forme d'un problème de décision. Nous avons ensuite détaillé ce problème d'interaction : nous avons tout d'abord montré comment représenter le voisinage de l'agent, puis nous avons décrit comment formaliser l'ensemble des sous-problèmes formant la composante d'interaction, afin notamment de spécifier les dépendances entre agents.

L'intérêt du modèle obtenu repose principalement sur la taille des structures manipulées. En effet, en séparant la tâche des interactions, et en n'introduisant de dépendance que lorsque cela s'avère nécessaire, on a réduit au maximum l'explosion combinatoire propre à ce type de problème. Il est de plus possible de borner « arbitrairement » cette explosion combinatoire, en limitant le nombre maximum d'interactions prises en compte simultanément (selon un ordre de préférence sur ces interactions).

Pour autant, le modèle conserve une expressivité forte, puisque l'on peut représenter des dépendances aussi bien sur les transitions que sur les récompenses, et ce dynamiquement (l'ensemble des agents en interaction n'étant pas défini au préalable). On a cependant fait le choix de raisonner sur les actions individuelles des agents, plutôt que sur les actions jointes du groupe : on perd donc la possibilité de faire de la coordination explicite entre agents. Ainsi, la seule solution pour exécuter des actions jointes est d'utiliser une action individuelle, en supposant que les autres agents seront coopératifs. Il reste donc à vérifier la qualité des politiques que l'on peut produire, en utilisant ce modèle. Pour cela, nous montrerons dans la partie suivante (algorithmes) que l'on peut calculer de bons comportements de groupe, grâce à la prise en compte du voisinage dans la politique individuelle de l'agent.

# Conclusion de la partie II

On dispose maintenant du modèle DyLIM, pour la représentation de problèmes de décision multiagents, sous-incertitude, en environnement partiellement observable. Ce modèle permet de traiter séparément la tâche de l'agent et son problème d'interactions, mais également d'identifier chaque sous-problème au sein de cette composante d'interaction. On peut alors, au sein de chaque cluster d'interactions engendré, raisonner sur les états relatifs du voisinage. On traite ainsi dynamiquement les interactions entre l'agent et ses voisins et on peut prendre en compte les dépendances locales sur les transitions, récompenses et observations. Ce modèle nous permet notamment de formaliser le problème de Thales, que nous tâcherons de résoudre dans la partie suivante de ce document (partie traitant des algorithmes et expérimentations).

L'avantage principal de DyLIM, relativement aux autres modèles existants, est qu'il permet un raisonnement basé interactions sans pour autant se limiter en termes d'applicabilité. D'un autre côté, il nous a fallu réaliser d'autres simplifications afin de maintenir possible le passage à l'échelle. Nous avons donc transformé le problème multiagent en un problème monoagent, en ne raisonnant que sur les actions individuelles. Cette simplification empêche alors l'usage de coordination explicite (prendre une décision jointe), et remplace celle-ci par une coordination probabiliste (j'effectue cette action, en espérant que l'autre agent soit coopératif). Malgré tout, la représentation des interactions utilisée permet d'intégrer le voisinage dans la connaissance de l'agent et de raisonner sur l'évolution de ces voisins. Cela nous permet de résoudre des problèmes qui étaient, jusque là, intraitables via les approches existantes. Nous verrons par la suite que les politiques calculées, bien que sous-optimales, restent de bonne qualité.

Ainsi, la troisième partie de ce document introduira plusieurs algorithmes de résolution possibles. Nous montrerons, via une série de benchmarks, les points forts et points faibles de cette approche (nous testerons entre autre des problèmes nécessitant une coordination explicite). Il nous faudra pour cela analyser deux aspects en particulier : la qualité des politiques calculées, et les temps de résolution. Nous verrons alors comment la structure de notre modèle permet de simplifier le processus de résolution, à travers deux algorithmes en particulier (l'un privilégiant la vitesse de résolution, l'autre la qualité des politiques produites). Nous verrons également comment ces différentes approches peuvent être traitées via des méthodes de calcul parallèle.

# Troisième partie

# Algorithmes de résolution

# Introduction à la partie III

La partie II nous a permis d'introduire le modèle DyLIM (*Dynamic Local Interaction Model*), pour la représentation de problèmes de planification multiagents, en environnement partiellement observable et sous incertitude. Pour rappel, ce modèle permet de décrire un problème multiagent comme étant l'union de deux composantes : la tâche, et les interactions. Ainsi, la tâche décrit le problème individuel d'un agent, c'est à dire la façon dont il évolue dans son environnement, et les récompenses/pénalités qu'entraînent ses actions. D'un autre côté, la composante d'interactions décrit l'influence qu'auront les autres agents sur la tâche, c'est-à-dire non seulement sur les transitions, mais également sur les récompenses obtenues. Ainsi, l'agent cherchant à calculer une politique basée sur ce modèle pourra optimiser son comportement individuel, tout en prenant en considération l'existence des autres agents.

Ce modèle offre plusieurs avantages, relativement aux autres modèles existants. Il permet, principalement, de simplifier considérablement la complexité combinatoire du problème à résoudre, en comparaison à une représentation traditionnelle de type DEC-POMDP. D'autres approches existent, basées sur cette notion d'interaction, mais ont toutes pour inconvénient de réduire le champ d'application des DEC-POMDPs. DyLIM, en permettant de traiter tout type d'interaction, ne souffre pas de cet inconvénient. Notre objectif ici est donc de calculer une politique basée sur ce modèle, en tirant parti de ces avantages (complexité combinatoire réduite, et large champ d'application). Les résultats présentés dans cette partie ont notamment été publiés dans [Canu et Mouaddib, 2011b].

Les données, telles qu'elles sont représentées dans le modèle DyLIM, ne sont pas directement exploitables pour générer une politique. Ainsi, le chapitre 8 propose un ensemble d'algorithmes permettant de générer un problème pouvant être soumis au solveur, à partir des données fournies. Nous verrons notamment comment calculer les transitions et récompenses jointes, à partir des fonctions individuelles, ainsi que les deux principaux types d'interaction pouvant être rencontrés.

Une fois ce pré-traitement réalisé, on peut envisager le calcul d'une politique. Le chapitre 9 fourni donc les algorithmes nécessaires au calcul de cette politique, en détaillant notamment deux approches possibles, plus ou moins performantes, selon la dimension du problème à résoudre. Ce chapitre montre également comment optimiser nos algorithmes, via une approche de calcul parallèle. Nous verrons ainsi que l'on peut drastiquement diminuer le temps nécessaire au pré-traitement des données, pour peu que l'on dispose d'une machine avec suffisamment de cœurs.

Ces deux chapitres suffisent à la résolution du problème de planification. Il serait intéressant, ensuite, d'étudier l'efficacité des approches présentées ici. Pour cela, le chapitre 10 propose une analyse de complexité pour l'ensemble des algorithmes introduits dans cette partie, qu'il s'agisse du pré-traitement des données (approche parallèle, ou classique) ou du calcul des politiques. Nous verrons alors l'impact fort qu'a le type d'interactions considérées (exactes ou approchées) sur la complexité en temps des algorithmes.

Pour finir, nous validerons à la fois la qualité de nos algorithmes, et leur complexité en temps, via une série de benchmarks. Ainsi, le chapitre 11 propose un ensemble de tests réalisés sur des problèmes devenus standards au sein de la communauté. Nous verrons notamment l'impact du nombre d'agents, impliqués dans le problème et considérés dans le voisinage de l'agent, sur les temps de résolution et la qualité des politiques produites. Nous étudierons finalement la résolution du problème de Thales, afin de valider l'applicabilité de notre approche à des problèmes applicatifs « réels ».

# Chapitre 8

# Construction du problème via DyLIM

**Sommaire**



Afin de résoudre un problème de planification exprimé via le modèle DyLIM, deux étapes algorithmiques sont nécessaires : générer des données manipulables (le modèle exigeant un prétraitement), puis traiter ces données afin d'en extraire une politique. Ce chapitre décrit la phase de prétraitement. Ainsi, nous commencerons par décrire l'approche envisagée, dont la philosophie s'écarte des approches « traditionnelles ». Nous donnerons ensuite l'algorithme de pré-traitement, permettant de générer (à partir des données fournies) un processus de décision markovien dont la solution sera une politique monoagent pour le problème traité. Nous finirons par montrer comment modifier cet algorithme pour prendre en compte les relations approchées (nous verrons notamment l'impact sur le calcul des fonctions de transition et de récompense entre états joints relatifs).

## 8.1 Principe des algorithmes envisagés

Les algorithmes classiques, de recherche d'une politique jointe parmi l'ensemble des politiques possibles, ne sont pas applicables aux problèmes de grande taille (ceux-ci souffrant

inévitablement d'une explosion combinatoire). Nous avons donc choisi de calculer une politique individuelle, sous-optimale (relativement à la politique jointe optimale) mais malgré tout de bonne qualité. On rappelle que le problème est exprimé via notre modèle DyLIM, c'est-à-dire via un couple $(\langle S,A,T,R,\Omega,O\rangle;\langle SR,\Omega R,OR,\{C^1,C^2,\ldots,C^{|C|}\}\rangle)$.

### 8.1.1 Résolution complètement décentralisée

Nous avons vu, dans le chapitre précédent, en quoi résoudre le problème multiagent devenait rapidement trop complexe, lorsque l'on augmente le nombre d'agents (en raison de l'explosion combinatoire). On va donc plutôt calculer une politique par agent. On peut alors se permettre de décentraliser complètement le processus, chaque agent calculant sa propre politique, sans utiliser de planificateur central ni échanger d'informations avec les autres agents. Il faut, si on veut calculer des politiques de qualité satisfaisante, considérer l'existence des autres agents (notamment les voisins) lors de la prise de décision. On parle ici de voisinage au sens large.

**Définition 33 (voisinage)** *Soit deux agents $A$ et $B$. On dit que l'agent $B$ est un voisin de l'agent $A$ ssi $T(s_A,a,s'_A) \neq T((s_A,s_B),a,s'_A)$ ou $R(s_A,a,s'_A) \neq R((s_A,s_B),a,s'_A)$.*

Les états de ces voisins évoluent à chaque pas de temps, mais on n'a aucun contrôle sur cette évolution. On peut par contre observer ces états. Ainsi, l'agent va considérer son voisinage comme une variable incontrôlable de l'environnement. Le problème sera donc bien monoagent, tout en intégrant le voisinage, et on raisonnera donc sur les actions individuelles. Pourtant, les fonctions de transition $T^i$ et de récompense $R^i$ associées à chaque cluster d'interactions $C^i$ peuvent être définies sur les actions jointes. On doit alors approximer ces fonctions par leur projection sur l'ensemble des actions individuelles :

- pour les fonctions de récompense, cette approximation est simple à réaliser : pour une action individuelle donnée, on fait un max, un min ou une moyenne sur l'ensemble des actions jointes possibles, associées à cette action individuelle,
- pour les fonctions de transition, on ne peut pas se contenter d'appliquer un opérateur de ce type, d'une part parce qu'un max ou un min (par exemple) appliqué à des transitions n'aurait aucun sens, mais aussi parce qu'il faut garantir la cohérence des probabilités (elles doivent sommer à 1).

Il semble donc nécessaire de trouver une heuristique, pour prévoir les actions du voisinage (toujours dans cette optique d'approximer la fonction de transition jointe par une fonction basée sur les actions individuelles). Il est toutefois difficile de trouver une telle heuristique, à moins de proposer une solution ad-hoc à chaque problème traité. On peut alors se détacher de tout choix d'action, en calculant une **atteignabilité** sur les états joints relatifs. L'idée est la suivante :

1. pour tout état joint relatif $sr$ et action individuelle $a$, on cherche l'ensemble des états joints relatifs $sr'$ atteignables (l'union des états joints relatifs vers lesquels les voisins peuvent transiter si ils exécutent l'action jointe $a^1$, ou $a^2$, etc. pour tout action jointe $a^i$ possible),

2. on calcule, pour chaque $sr'$, un poids donné par la somme sur les actions jointes $a^i$ : $\sum_{a^i} P(sr'|sr,a^i)$, avec $P(sr'|sr,a^i)$ la probabilité de passer de $sr$ à $sr'$ en appliquant $a^i$,
3. on ordonne ces états joints relatifs selon leur poids (avec $sr^1$ le $sr'$ de poids le plus fort),
4. on choisit alors une répartition de probabilités, telle que $\forall i, T(sr,a,sr^i) > T(sr,a,sr^{i+1})$,
5. on dispose maintenant d'une fonction basée sur les actions individuelles, approchant la fonction de transition jointe, sans pour autant faire de supposition sur les actions jointes exécutées par les autres agents.

Finalement, on disposera bien d'un problème monoagent, dont les états seront les états individuels de l'agent joints aux états observés des voisins, et dont les actions possibles seront l'ensemble des actions individuelles de l'agent. Cette représentation a de plus l'intérêt de permettre l'usage d'états de croyance (ceux-ci étant incalculables pour un DEC-POMDP, l'action jointe étant inconnue), ce qui permettra une résolution simplifiée du problème.

### 8.1.2 Traiter la tâche indépendamment des interactions

Le problème, tel qu'il est formalisé avec le modèle DyLIM, implique la résolution de deux composantes : la tâche individuelle, et les interactions. Nous verrons dans le chapitre suivant que celles-ci peuvent être traitées séparément, ou comme un tout. Dans tous les cas, ces deux composantes sont décrites séparément, comme des problèmes monoagents (les fonctions de transition et récompense dépendant uniquement des actions individuelles de l'agent), tout en prenant en compte le voisinage (comme nous l'avons montré dans la section précédente).

La tâche de l'agent est décrite sous la forme d'un POMDP, qui est directement exploitable. Les interactions par contre, ne sont que partiellement formalisées. En effet, le modèle permet (comme nous l'avons vu dans la partie II) de décrire les relations entre agents, et de spécifier les probabilités de transition et récompenses associées lorsque celles-ci ne peuvent être décomposées en un ensemble de fonctions individuelles (afin, donc, de spécifier l'influence qu'ont les interactions sur l'agent). On dispose donc bien, pour tout cluster d'interactions $C^i$, de :

- un ensemble d'états $S^i$ (les états joints relatifs concernés par le cluster),
- un ensemble d'actions (les actions individuelles de l'agent),
- un ensemble d'observations $\Omega R$ que l'on peut effectuer au sujet des états de $S^i$,
- une fonction d'observation associée à $\Omega R$.

Les fonctions de transition et de récompense par contre, ne sont que partiellement définies. Il va donc falloir construire ces fonctions, pour tout triplet $(sr,a,sr')$. Pour chaque triplet, deux possibilités s'offrent à nous : soit la transition (ou la récompense) est indécomposable entre agents (auquel cas sa valeur est spécifiée dans le modèle), soit elle est décomposable. Dans ce cas (transition décomposable), la probabilité n'est pas fournie, mais peut être obtenue par un produit (ou une somme, pour les récompenses) des fonctions individuelles.

On a donc, finalement, un POMDP pour résoudre la tâche de l'agent, et un POMDP pour résoudre ses interactions (celui-ci reposant sur les états joints relatifs et les actions individuelles). Nous verrons que le POMDP représentant les interactions peut-être vu comme un MDP muni

d'une fonction d'observation (en appliquant le principe des Q-MDPs). Cette fonction d'observation étant déjà fournie, il nous reste simplement à générer le MDP associé. Finalement, chaque cluster d'interactions décrivant un problème semi-indépendant, on pourra simplifier la résolution en générant un MDP par cluster, puis en calculant les transitions permettant de passer d'un MDP à l'autre. La section suivante décrit l'algorithme permettant ce pré-traitement des données.

## 8.2 Générer des données manipulables

Cette section présente les algorithmes nécessaires à la génération des MDP associés aux différents clusters d'interactions. Nous verrons tout d'abord les algorithmes principaux, permettant de générer puis connecter les MDPs entre eux, puis nous détaillerons le calcul des fonctions de transition et de récompense.

### 8.2.1 Associer un MDP à chaque cluster d'interactions

L'algorithme 3 est l'algorithme de base, permettant de construire le MDP associé à chaque cluster d'interactions. Cette construction repose sur deux autres algorithmes :

- `construire()`, pour le calcul des transitions et récompenses manquantes,
- `relier()`, pour calculer les transitions permettant de passer d'un MDP à l'autre (ceux-ci étant faiblement couplés).

Ces deux algorithmes sont donnés dans la suite de ce document.

**Algorithme 3** : Génération des MDPs.

**Entrées** : $C = \{C^1, \ldots, C^{|C|}\}$ l'ensemble des clusters d'interactions
**Sorties** : les MDPs associés aux clusters $\{M^1, \ldots, M^{|C|}\}$
**pour chaque** $C^i \in C$ **faire**
  `// on a` $C^i = (S^i, T^i, R^i)$.
  $S^{MDP} \leftarrow S^i$;
  $A^{MDP} \leftarrow A^{POMDP}$;
  $T^{MDP} \leftarrow \emptyset$;
  $R^{MDP} \leftarrow \emptyset$;
  $M^i \leftarrow$ nouveau MDP $\langle S^{MDP}, A^{MDP}, T^{MDP}, R^{MDP} \rangle$;
  `// l'algorithme 4 permet d calculer les transitions et récompenses :`
  construire($M^i$, $T^i$, $R^i$);
`// on connecte les MDPs entre eux grâce à l'algorithme 5 :`
**pour chaque** $M^i$ **faire**
  **pour chaque** $M^j \neq M^i$ **faire** relier($M^i$,$M^j$);
  `// on fait en sorte que` $\forall s \in S^{M^i}, \forall a \in A, \sum_{s' \in S^{M^i}} T(s,a,s') = 1$ :
  $normaliser(M^i)$;
**retourner** $\{M^1, \ldots, M^{|C|}\}$;

l'algorithme 4 décrit la construction des transitions et récompenses, pour un MDP donné. Certaines transitions (ou récompenses) sont fournies lors de la définition du problème, parce qu'elles sont indécomposables entre agents : on peut donc se contenter de les conserver comme telles. Les autres transitions et récompenses sont calculées, à partir des fonctions individuelles.

**Algorithme 4** : `construire()`, calcule les transitions et récompenses d'un MDP.

**Entrées** : $M = \langle S,A,T,R\rangle$ un MDP et $T^i$, $R^i$ provenant du cluster d'interactions associé
**pour chaque** $(sr,a,sr') \in S \times A \times S$ **faire**
  `// on calcule les transitions et récompenses (algorithme 6) :`
  **si** $T^i(sr,a,sr')$ *est défini* **alors** $T(sr,a,sr') \leftarrow T^i(sr,a,sr')$;
  **sinon** $T(sr,a,sr') \leftarrow$ transition$(sr,a,sr')$;
  **si** $R^i(sr,a,sr')$ *est défini* **alors** $R(sr,a,sr') \leftarrow R^i(sr,a,sr')$;
  **sinon** $R(sr,a,sr') \leftarrow$ recompense$(sr,a,sr')$;

Les MDPs sont faiblement couplés : il existe donc quelques transitions, permettant de passer d'un MDP à l'autre (algorithme 5). Supposons que l'on traite un MDP $M^i$, permettant de transiter vers un autre MDP $M^j$. On représentera cette possibilité en ajoutant à $M^i$ un état abstrait, représentant la transition vers un des états de $M^j$. On pose alors $\bar{S}^i$ qui désigne l'ensemble $S^i$ privé des états abstraits. On calcule ensuite la probabilité de transiter vers ces états abstraits, et la récompense associée. On considère que ces états sont des états puits : ainsi, à l'exécution, l'agent suivra la politique de $M^i$, prenant en compte la possibilité de transiter vers un autre cluster d'interactions. Une fois cette transition effectuée, il suivra la politique de $M^j$. Nous verrons par la suite comment calculer la valeur de ces états particuliers.

**Algorithme 5** : `relier()`, calcule les transitions entre deux MDPs.

**Entrées** : $M^i = \langle S^i,A^i,T^i,R^i\rangle$ et $M^j = \langle S^j,A^j,T^j,R^j\rangle$ deux MDPs
`//` $j$ `un nouvel état, abstrait, représentant` $M^j$`:`
$S^i \leftarrow S^i \cup \{j\}$;
`//` `l'état` $j$ `est considéré comme un état puits :`
**pour chaque** $a \in A^i$ **faire**
  $R^i(j,a,j) \leftarrow 0$;
  $T^i(j,a,j) \leftarrow 1$;
`//` $A^i(sr)$ `les actions appartenant à` $A^i$`, applicables dans l'état` $sr$`,`
**pour chaque** $sr \in \bar{S}^i$ *et* $a \in A^i(sr)$ **faire**
  `// calcul des transitions et récompenses (algorithme 6) :`
  $T^i(sr,a,j) \leftarrow \sum_{sr' \in \bar{S}^j}$ transition$(sr,a,sr')$;
  $R^i(sr,a,j) \leftarrow \dfrac{\sum_{sr' \in \bar{S}^j} [\text{transition}(sr,a,sr') \times \text{recompense}(sr,a,sr')]}{\sum_{sr' \in \bar{S}^j} \text{transition}(sr,a,sr')}$;
  $T^i(j,a,sr) \leftarrow 0$;

Ces trois algorithmes forment le squelette du processus de génération des MDPs associés aux différents clusters d'interactions. Il reste à montrer comment calculer une transition (ou une récompense), à partir des fonctions individuelles.

### 8.2.2 Calcul des transitions et récompenses entre états joints relatifs

On s'intéresse maintenant au calcul des transitions et récompenses, à partir des fonctions individuelles. On cherche donc la probabilité qu'a le voisinage de passer d'un état joint relatif $sr$ à un état $sr'$, lorsque l'agent exécute l'action individuelle $a$ (et qu'on ignore quelles actions ont exécuté les voisins). Durant cette transition, plusieurs choses peuvent se passer :

- l'ensemble des agents de $sr$ peuvent simplement transiter vers l'état joint relatif $sr'$,
- durant la transition, des agents présents en $sr$ peuvent **quitter l'interaction**. Cela arrive lorsqu'un ou plusieurs voisins transitent, individuellement, vers des états dans lesquels il n'y a plus aucune interaction avec l'agent,
- de même, des agent venus de l'extérieur (c'est-à-dire n'étant, précédemment, pas en interaction avec l'agent) peuvent intégrer le voisinage.

Bien sûr, seules les **combinaisons cohérentes** sont à considérer, c'est à dire celles pour lesquelles le nombre d'agents en $sr'$ vérifie l'équation $|sr'| = |sr| - out + in$, avec $out$ le nombre d'agents quittant l'interaction, et $in$ le nombre d'agents (venus de l'extérieur) intégrant celle-ci. Il faut donc, pour calculer la probabilité de transition de $rs$ vers $rs'$ par $a$, prendre en compte l'ensemble des combinaisons cohérentes (voir l'exemple de la figure 8.1).

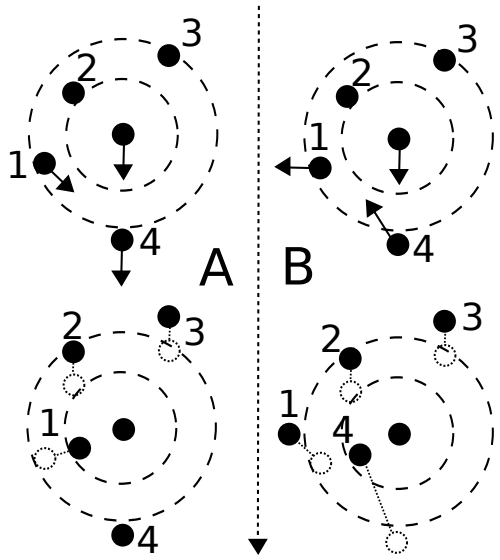


FIGURE 8.1 – transition entre états joints relatifs

Ainsi, on devra considérer toutes les valeurs de $out$ possibles : y'a t'il $out = 1$ agent qui quitte l'interaction ? Ou bien $out = 2$, $out = 3$, etc. Dans tous les cas, on posera $in = |sr'| - |sr| + out$. De plus, pour chaque valeur de $out$, plusieurs situations sont à considérer : quels sont les agents de $sr$ qui quittent l'interaction ? De même, quels sont les états relatifs $sr'_i \in sr'$ associés à des agents

venus de l'extérieur, et ceux correspondant aux agents de $sr$ n'ayant pas quitté l'interaction ? L'exemple donné en figure 8.1 représente un agent (le point central) et ses voisins. L'espace autour de l'agent est divisé en 3 zones, représentées par les cercles en pointillés : proche, loin, et hors-voisinage. Les flèches indiquent les déplacements des agents. Ainsi, la partie haute de cette figure décrit l'état joint relatif initial `sr=(loin-nord-ouest,loin-sud-ouest,loin-nord-est)` tandis que la partie basse décrit l'état final `rs'=(loin-nord-ouest,proche-sud-ouest)`. Les colonnes gauche (zone A) et droite (zone B) décrivent deux façons de passer de $sr$ à $sr'$ :

1. Initialement, l'agent est en interaction avec 3 autres agents (l'agent 4 étant hors-voisinage). Dans l'exemple A, l'agent se déplace vers le sud, les agents 2 et 3 ne bougent pas, l'agent 1 se déplace vers le sud-est et l'agent 4 se déplace vers le sud. La partie en bas à gauche de la figure montre le résultat de ces mouvements. L'agent 3 a quitté l'interaction, aucun nouvel agent n'est venu de l'extérieur et les agents 1 et 2 ont changé d'état relatif.

2. Dans l'exemple B, la situation initiale est la même, mais l'agent 1 se déplace vers l'ouest tandis que l'agent 4 se déplace vers le nord-ouest. Le résultat est pourtant le même que dans l'exemple A, mais c'est les agents 1 et 3 qui ont quitté l'interaction, tandis que l'agent 4 a intégré celle-ci et que l'agent 2 a simplement changé d'état relatif.

On a donc deux façons d'effectuer la transition de $sr$ à $sr'$, mais il en existe de nombreuses autres. Ainsi sur cet exemple, toute combinaison impliquant le départ de $out = 1$ ou $out = 2$ agents est à considérer à condition, si 2 agents quittent l'interaction, qu'un agent venu de l'extérieur intègre celle-ci (ici, l'agent 4). Pour chacune de ces combinaison, on peut calculer la probabilité de transition associée : en faisant la somme de ces probabilités, on obtient la transition de $sr$ à $sr'$ par $a$. Pour une combinaison donnée, la probabilité est donnée par un produit de trois éléments :

- $p^{out}$ : la probabilité, pour les $out$ agents étant amenés à quitter l'interaction, de transiter (à partir d'un état relatif $sr_i$ présent dans $sr$) vers un état hors-voisinage (peu importe lequel, le tout étant de quitter l'interaction),
- $p^{in}$ : la probabilité que $|sr'| - |sr| + out$ agents, venus de l'extérieur, intègrent l'interaction (en transitant vers un état relatif $sr'_i$ présent dans $sr'$). Comme ces agents sont non-observés, on ne connaît par leurs états : on suppose donc (de façon heuristique) une distribution uniforme des agents non-observés sur l'ensemble des états hors-voisinage,
- $p^{stay}$ : la probabilité que les $|sr| - out$ agents n'ayant pas quitté l'interaction (et donc, occupant les $sr_i$ restants) transitent vers un des états $sr'_i$ non-occupé par un agent venu de l'extérieur.

Il reste une dernière difficulté, pour calculer ces fonctions de transition (ainsi que les fonctions de récompense associées). Prenons le calcul de $p^{stay}$ par exemple (mais ce raisonnement s'applique aussi aux deux autres éléments de la transition jointe). On veut la probabilité qu'un groupe d'agents transite vers un nouvel état joint relatif : il suffit de calculer, pour chaque agent, la probabilité qu'il a d'effectuer cette transition, puis de faire un produit de ces transitions individuelles. Le calcul de cette probabilité, pour un agent donné, n'est toutefois pas trivial.

On sait que l'agent est dans l'état relatif $sr_i$, et qu'il doit transiter vers $sr'_i$. Les fonctions individuelles ne sont cependant pas définies sur les états relatifs, mais bien sur les états individuels. Il faut donc connaître l'état réel de l'agent, et non son état relatif. Imaginons par exemple que, au vu de l'état relatif $sr_i$, l'agent $i$ puisse être dans un état $s_1$ ou $s_2$ (de même, au vu de $sr'_i$, l'agent $i$ peut transiter vers $s_3$ ou $s_4$). Imaginons maintenant que l'on ait $P(s_3|s_1,a) \neq P(s_4|s_1,a) \neq P(s_3|s_2,a) \neq P(s_4|s_2,a)$ : quelle est alors la probabilité $P(sr'_i|sr_i,a)$ ? L'approche la plus simple serait de calculer une moyenne de toutes les situations possibles, c'est à dire pour toutes les **instances** possibles de $sr_i$ et de $sr'_i$ (voir définition 30, page 94). Celles-ci sont, cependant, potentiellement très nombreuses. Afin de limiter les temps de calcul, on pourra se contenter de $n$ instances tirées au hasard parmi l'ensemble des instances possibles.

L'algorithme 6 décrit le calcul d'une probabilité de transition $T(sr,a,sr')$. On parlera notamment d'une « partie » $X$ (de taille $n$) d'un état joint relatif $sr$, désignant ainsi cet état $sr$ réduit à $n$ agents. L'état joint relatif $X = (sr_1,sr_3)$ par exemple est une partie de taille 2 de l'état jointe relatif $sr = (sr_1,sr_2,sr_3)$ : sur cet exemple, on a 3 façons d'extraire une partie de taille 2.

**Algorithme 6** : `transition()`, calcule une probabilité de transition $T(sr,a,sr')$.

**Entrées** : $sr$, $a$, $sr'$
**Sorties** : la probabilité de transition $T(sr,a,sr')$
$T^i(sr,a,sr') \leftarrow 0$;
// pour rappel, $I$ désigne le nombre d'agents impliqués dans le problème :
**pour chaque** $max(0,|sr| - |sr'|) \leq$ `out` $\leq min(|sr|, I - |sr'|)$ **faire**
  `groups` $\leftarrow$ l'ensemble des parties `G` de `sr` possibles, telles que $|G| =$ `out`;
  **pour chaque** *partie* `G` $\in$ `groups` **faire**
    **pour chaque** *partie* `sr`" *de* `sr'` *telle que* $|sr"| = |sr| - out$ **faire**
      // probabilité que les agents de $G$ quittent l'interaction :
      $p^{out} \leftarrow \sum_{RC^j \neq RC^i} P_G(RC^j|RC^i,a)$;
      // probabilité que les agents de $sr$, n'appartenant pas à $G$ (on notera $(sr/G)$ cet ensemble), transitent vers $sr"$ :
      $p^{stay} \leftarrow P(sr"|(sr/G),a)$;
      // probabilité que des agents venus de l'extérieur génèrent les états relatifs $(sr'/sr")$ :
      $p^{in} \leftarrow P((sr'/sr")|sr,a)$;
      // la probabilité totale :
      $P^{(G,sr")}(sr,a,sr') \leftarrow p^{out} \times p^{stay} \times p^{in}$;
    $P^G(sr,a,sr') \leftarrow \sum_{sr"} P^{(G,sr")}(sr,a,sr')$;
  $T^i(sr,a,sr') \leftarrow T^i(sr,a,sr') + \sum_{G \in groups} P^G(sr,a,sr')$;
**retourner** $T^i(sr,a,sr')$;

Cet algorithme (algo. 6) considère, comme décrit précédemment, toutes les combinaisons possibles avec *out* agents quittant l'interaction et $in = |sr'| - |sr| + out$ agents intégrant celle-ci. La valeur de *out* est ici définie sur l'intervalle $max(0,|sr| - |sr'|) \leq \texttt{out} \leq min(|sr|, I - |sr'|)$. Cet intervalle se justifie de la façon suivante :

- $max(0,|sr| - |sr'|) \leq \texttt{out}$ : il faut au moins que $|sr| - |sr'|$ agents quittent l'interaction, sans quoi trop de voisins seront présents après la transition. On ne peut toutefois pas faire quitter l'interaction à un nombre négatif de voisins (ce qui pourrait arriver lorsque $sr'$ implique plus d'agents que $sr$), d'où le $max(0,|sr| - |sr'|)$.
- $\texttt{out} \leq min(|sr|, I - |sr'|)$ : on ne peut pas faire quitter l'interaction à plus d'agents qu'il n'y en a de présents, d'où $\texttt{out} \leq |sr|$. De même, il faudra $|sr'|$ voisins présents après cette transition : si on fait quitter l'interaction à *out* agents, il en restera $(|sr| - out)$, auxquels s'ajouteront au maximum $(I - |sr|)$ nouveaux agents. Il faut donc vérifier l'équation $(I - |sr|) + (|sr| - out) \geq |sr'|$, d'où $out \leq (I - |sr'|)$ et donc $\texttt{out} \leq min(|sr|, I - |sr'|)$.

Ensuite, n'importe quelle situation menant au $sr'$ voulu est acceptable, on fait donc la somme des probabilités de transition pour chaque situation possible. Ainsi, on somme non-seulement sur les valeurs possibles pour *out*, mais aussi sur les différentes parties $sr''$ de $sr'$ et $G$ de $sr$. On ne donne ici qu'une intuition sur la façon dont on peut calculer $p^{out}$, $p^{stay}$ et $p^{in}$, ces calculs relevant du détail algorithmique. Le lecteur désireux d'implémenter cette méthode trouvera les algorithmes détaillés (utilisés dans notre solveur) en annexe A. Le calcul des récompenses associées à chaque transition suit le même algorithme, la seule différence se situant au niveau de $p^{out}$, $p^{stay}$ et $p^{in}$ à remplacer par des fonctions spécifiques au calcul de récompense.

On dispose maintenant d'un ensemble d'algorithmes permettant de générer le MDP associé à chaque cluster d'interactions, puis de connecter ces MDPs entre eux. Ces algorithmes permettent également d'effectuer le pré-traitement nécessaire à la résolution de ces MDPs, en calculant les transitions et récompenses manquantes, à partir des fonctions individuelles. Les algorithmes fournis ici se veulent génériques, et devront être adaptés au type d'interactions employées (exactes, ou approchées).

## 8.3 Interactions exactes ou approchées ?

Comme nous l'avons défini précédemment, deux types d'interactions sont envisageables. Pour rappel, les relations exactes supposent (1) que l'agent sache exactement combien de voisins sont en interaction avec lui, (2) que l'on puisse déduire, de l'état de l'agent et de l'état relatif d'un voisin, l'état réel de ce voisin et (3) que toutes les interactions observées fassent partie de l'état joint relatif. À l'inverse, les relations approchées permettent non seulement d'ignorer les états relatifs de faible importance (lorsque l'état joint relatif inclut déjà plus de $k$ relations), mais aussi une incertitude sur le nombre de voisins en interaction (un état relatif pouvant concerner un ou plusieurs voisins) et sur les états exacts des voisins.

### 8.3.1 Impact sur les algorithmes employés

Les algorithmes 3, 4 et 5 ne dépendent pas du type d'interactions employées. L'algorithme 6 par contre, suppose que l'on connaisse le nombre de voisins actuellement en interaction, afin d'en déduire le nombre *out* d'agents pouvant quitter cette interaction :

$$max(0,|sr| - |sr'|) \leq \texttt{out} \leq min(|sr|, I - |sr'|)$$

Ainsi, dès l'instant où l'on manipule des interactions approchées, on risque de calculer des valeurs impossibles pour *out*, puisque la taille de $sr$ ou de $sr'$ ne correspond pas nécessairement au nombre d'agents impliqués. On introduit donc un biais dans le calcul des transitions, puisque l'on va parfois supposer que des agents peuvent quitter l'interaction alors qu'ils ne sont pas assez nombreux pour cela, ou à l'inverse qu'ils ne peuvent pas quitter l'interaction alors que ce serait possible. On constate cependant, expérimentalement (voir chapitre 11), que cette approche reste la meilleure lorsque l'on ne peut pas détecter le nombre exacte de voisins présents.

### 8.3.2 Notion de « diffusion » des agents en interaction

Contrairement à ces premiers algorithmes, le calcul des transitions $p^{stay}$, $p^{out}$ et $p^{in}$ (ainsi que le calcul des récompenses associées) dépend directement du type d'interactions considérées. Étudions chacune des trois hypothèses vérifiées par les interactions exactes :

1. *L'agent sait exactement combien de voisins sont en interaction avec lui.* Si cette condition est vérifiée, la probabilité de transition jointe $P(sr'|sr)$ sera un simple produit des probabilités de transition $P(sr'_i|sr_i)$ de chaque agent $i$. Si on ne sait pas combien de voisins sont présents, comment calculer cette probabilité jointe ?
2. *On peut déduire les états exacts de chaque voisin.* Cette condition permet d'utiliser la fonction individuelle $T(s_i,a,s'_i)$ de chaque agent $i$, pour en déduire la probabilité $P(sr'_i|sr_i)$. Dans le cas contraire, comment calculer cette probabilité de transition ?
3. *Toutes les interactions observées font partie de l'état joint relatif.* Grâce à cette condition, on est certain que les agents n'appartenant pas à $sr$ sont dans des états hors-interaction. Dans le cas contraire, ces agents peuvent être n'importe où, car on peut choisir d'ignorer les interactions de faible importance, d'où une perte de précision.

Il semble donc relativement simple de calculer les fonctions $p^{stay}$, $p^{out}$ et $p^{in}$ dans le cas des interactions exactes, puisqu'il suffit de calculer l'état exact de chaque voisin, puis leurs probabilités de transition individuelles, avant d'en déduire la probabilité jointe. Le cas des interactions approchées est nettement plus complexe.

On peut introduire ici le concept d'**atteignabilité d'un état**. L'idée est la suivante : on ne sait pas exactement combien de voisins sont présents, on ne connaît pas leurs états exacts et on ne sait pas quelles actions ceux-ci vont exécuter. Il est donc impossible de calculer, de manière exacte, leurs probabilités de transition jointes. Par contre, on peut tenter de « faire au

mieux » avec les informations disponibles. Ainsi, certains états joints relatifs $sr'$ seront inatteignables, depuis l'état joint relatif $sr$ actuel. En ce qui concerne les $sr'$ atteignables, certains seront plus probables que d'autres (on peut imaginer, par exemple, un $sr'$ dont la plupart des instances peuvent être atteintes depuis $sr$, et un autre $sr''$ n'ayant que peu d'instances atteignables depuis $sr$). Ainsi, on peu attribuer un poids à chaque $sr'$ en fonction de son atteignabilité, pour en déduire une probabilité de transition. La fonction de transition ainsi obtenue reflétera donc le comportement en moyenne d'un groupe de voisins. Les expérimentations réalisées (chapitre 11) valident l'efficacité de cette approche.

L'annexe A décrit les algorithmes employés dans le cas des interactions approchées, en appliquant cette notion d'atteignabilité. En ce qui concerne les algorithmes permettant de traiter les interactions exactes : ceux-ci sont triviaux, et ne seront donc pas détaillés. On dispose donc maintenant de l'ensemble des algorithmes nécessaires à la génération des MDPs associés aux clusters d'interactions. Il reste désormais à résoudre ceux-ci, afin d'en extraire une politique de comportement pour chaque agent.

## Conclusion

Avant de chercher à calculer les politiques de chaque agent, un pré-traitement des données était nécessaire. Ainsi, dans ce chapitre, nous avons vu comment générer un ensemble de MDPs, représentant les différents clusters d'interactions décrivant le problème de coordination. Nous avons notamment divisé ce pré-traitement en trois grandes phases : (1) générer le MDP associé à chaque cluster, (2) « compléter » ces MDPs puis (3) les connecter entre eux. En effet, les données fournies par le modèle ne sont pas toujours suffisantes pour générer des MDPs cohérents, certaines transitions ou récompenses pouvant être manquantes (lorsque le voisinage de l'agent n'a pas d'influence sur ces fonctions). Il convient alors de compléter ces MDPs en calculant les transitions manquantes à partir des fonctions de transition individuelles de chaque agent (de même pour les récompenses). Il faut ensuite calculer les transitions permettant de passer d'un MDP à l'autre, symbolisant ainsi le changement de cluster d'interaction.

Nous avons également abordé le problème des interactions approchées, impliquant certains raffinements algorithmiques lors du calcul des transitions et récompenses jointes. Ces calculs étant peu intuitifs, nous avons décrit en annexe la façon dont nous les avons appliqués durant nos expérimentations. Nous disposons donc, à l'issue de ce chapitre, d'un ensemble d'algorithmes permettant de générer un problème multiagent « complet ». Le chapitre suivant traitera de la résolution de ce problème, afin de calculer la politique associée à chaque agent.

# Chapitre 9

# Résolution du problème précédemment construit

**Sommaire**



Le chapitre précédent nous a permis de générer un problème multiagent « complet », décrivant l'évolution des interactions entre l'agent et ses voisins via un ensemble de MDPs (basés, pour rappel, sur les actions individuelles de l'agent, une « moyenne » ayant été effectuée sur les actions jointes des voisins). Nous disposons également d'un problème de type POMDP, décrivant la tâche de l'agent. Il nous faut maintenant extraire de ces deux problèmes une politique unique de comportement pour l'agent. Nous verrons, dans ce chapitre, deux méthodes de résolution distinctes pour le calcul d'une telle politique. La première méthode propose de résoudre séparément la tâche de l'agent et ses interactions, puis de fusionner les deux politiques obtenues en une seule politique globale. Nous verrons que cette approche, bien que très efficace en temps de résolution, ne permet pas d'obtenir une politique optimale. À l'inverse, la seconde approche propose de traiter ces deux problèmes (tâche et interactions) comme un tout, afin d'en extraire une politique optimale. Cette approche génère des politiques de très bonne qualité, mais est plus coûteuse en temps. Nous verrons pour finir comment paralléliser les algorithmes proposés jusque là, afin de réduire drastiquement les temps de calcul.

## 9.1 Approche réactive : POMDP+MDPs

On peut traiter séparément la tâche de l'agent et son problème d'interactions. Ce faisant, on calculera une fonction de $Q$-valeurs **vecteur-valuée** au lieu d'une fonction de $Q$-valeurs traditionnelle. Cette fonction sera de la forme $Q = \langle Q^{ind}, Q^{coo} \rangle$, avec $Q^{ind}$ la fonction de $Q$-valeurs obtenue en résolvant le problème individuel (le POMDP décrivant la tâche de l'agent) et $Q^{coo}$ la fonction de $Q$-valeurs obtenue en résolvant le problème de coordination (l'ensemble des MDPs représentant l'évolution des interactions). Nous verrons alors qu'il est possible, à partir de cette fonction $Q$, d'extraire une politique individuelle pour l'agent. Cette approche ne considère que les interactions immédiates (horizon 1) : nous la qualifierons donc d'**approche réactive**.

### 9.1.1 Résolution du POMDP et des MDPs

On dispose, à priori, de deux composantes « simples » que sont :

1. *la tâche de l'agent* - Un POMDP est un problème complexe (PSPACE-complet), mais la tâche de l'agent est en général suffisamment simple pour que le POMDP associé puisse être résolu en un temps raisonnable.
2. *les interactions* - Cette composante est de dimension nettement plus importante, mais le formalisme MDP permet une résolution efficace de ce type de problème (P-complet).

Ainsi, la séparation du problème en deux composantes permet de manipuler les données volumineuses via un formalisme efficace, et de ne garder que le minimum de données nécessaires pour le formalisme coûteux en temps. Il paraît donc logique de maintenir cet avantage, en résolvant séparément ces deux aspects du problème global. On aura alors deux fonctions de $Q$-valeurs, l'une portant sur les états individuels de l'agent et l'autre portant sur les états relatifs du voisinage. Il faudra donc, pour pouvoir utiliser ces fonctions, maintenir deux états de croyances. On maintiendra ainsi $b^{ind}$, une distribution de probabilités sur les états individuels et $b^r$, une distribution de probabilités sur les états joints relatifs. La façon dont on maintient à jour ces états de croyance est détaillée plus loin dans ce document (section 9.1.2).

La composante individuelle (tâche de l'agent) est représentée par un POMDP « classique » et peut donc être résolue via n'importe quel algorithme existant. Nous avons choisi, durant nos expérimentations, d'utiliser une implémentation existante de l'algorithme SARSOP [Kurniawati *et al.*, 2008]. Une fois cette composante résolue, on obtient une fonction de $Q$-valeurs $Q^{ind}(b^{ind}, a)$, associant une valeur espérée à toute action $a$ exécutée dans l'état de croyances individuel $b^{ind}$. La composante de coordination (interactions) est moins classique, puisqu'elle implique un ensemble de MDPs tels que l'on puisse transiter de l'un à l'autre. On ne peut donc pas se contenter de résoudre ces MDPs séparément les uns des autres, la valeur de chaque MDP $i$ dépendant directement des valeurs des MDPs $j \neq i$ vers lesquels on peut transiter. L'idée sera alors de résoudre chaque MDP via l'algorithme de Value-Iteration habituel, puis de remplacer la valeur des états abstraits par une valeur représentant le MDP associé (ici : la valeur max pouvant être obtenue dans le MDP en question). Ce processus de résolution est décrit dans l'algorithme 7.

**Algorithme 7** : Résolution d'un ensemble de MDPs quasi-indépendants.

**Entrées** : $M$ un ensemble de MDPs, $\epsilon$ une borne proche de zéro

**Sorties** : $Q$ une fonction de Q-valeurs

// $E$ l'ensemble des états abstraits, tels que $E[i]$ représente une transition vers le MDP $i$ :

$E \leftarrow \{etats\ abstraits\}$;

// rappel : on manipule (dans cet algorithme) des états joints relatifs :

**pour chaque** $MDP^i \in M$ *et* $sr \in S^i$ **faire**

  **si** $sr \in E$ **alors** $V[i](sr) \leftarrow 0$ **sinon** $V[i](sr) \leftarrow R^i(sr)$;

**répéter**

  // phase 1 (exécuter un Value-Iteration, sans mettre à jour $E$) :

  **pour chaque** $MDP^i \in M$ **faire**

    $V'[i] \leftarrow ValueIteration(MDP^i)$;

    **pour chaque** $sr \in E$ **faire** $V'[i](sr) \leftarrow 0$;

  // phase 2 (propager les valeurs des états abstraits) :

  **pour chaque** $MDP^i \in M$ **faire**

    **pour chaque** $MDP^j \in M$ *tel que* $j \neq i$ **faire**

      // (rappel) $E[i]$ représente $MDP^i$ :

      $V'[j](E[i]) \leftarrow \max_{sr \in S^i}(V'[i](sr))$;

  // phase 3 (mettre à jour les fonctions de valeurs):

  $\Delta \leftarrow 0$;

  **pour chaque** $MDP^i \in M$ **faire**

    $\delta \leftarrow 0$;

    **pour chaque** $sr \in S^i$ **faire**

      $d \leftarrow |V[i](sr) - V'[i](sr)|$;

      **si** $d > \delta$ **alors** $\delta \leftarrow d$;

    **si** $\delta > \Delta$ **alors** $\Delta \leftarrow \delta$;

  $V \leftarrow V'$;

**jusqu'à** $\Delta < \epsilon$ ;

// on « fusionne » les fonctions de valeur en une fonction $Q^r$ globale :

**pour chaque** $MDP^i \in M$ **faire**

  **pour chaque** $sr \in S^i$ *tel que* $sr \notin E[i]$ **faire**

    **pour chaque** *action* $a$ **faire** $Q^r(sr,a) \leftarrow R^i(sr,a) + \sum_{sr' \in S^i} T^i(sr,a,sr').V[i](sr')$;

**retourner** $Q^r$;

Cet algorithme repose donc sur un processus de résolution itératif, dans lequel on répète successivement deux phases de calcul jusqu'à stabilisation des valeurs : résoudre chaque MDP, et propager les valeurs vers les états abstraits. On obtient finalement une fonction de $Q$-valeurs $Q^r(sr,a)$, associant une valeur espérée à chaque action $a$ exécutée par l'agent, lorsque l'état joint relatif de ses voisins est $sr$. Afin de manipuler cette fonction dans un cadre d'observabilité partielle, on peut appliquer la méthode utilisée pour les Q-MDPs, et définir $Q^{coo}(b^r,a) = \sum_{sr \in b^r} b^r(sr).Q^r(sr,a)$. Cette fonction donnera donc la valeur espérée, lorsque l'on exécute une action (individuelle) $a$ dans l'état de croyance $b^r$.

Nous avons vu, en introduisant le formalisme POMDP, comment maintenir un état de croyance sur les états individuels de l'agent. Maintenir l'état de croyance sur les états joints relatifs sera par contre plus complexe, la probabilité qu'un voisin soit dans tel ou tel état relatif dépendant de l'état individuel actuel (en effet, selon l'état de l'agent, certain états relatifs pourront être rendus impossibles). La prochaine section traite de ce problème particulier.

### 9.1.2 Maintient des états de croyance et fonction de valeur

Nous avons précédemment montré (partie I, chapitre 2, section 2.2.2) comment maintenir l'état de croyance $b^{ind}$ sur les états individuels. Pour rappel, à l'instant $t$, la mise à jour de $b^{ind}_{t-1}$ vers $b^{ind}_t$ après avoir exécuté $a$ et observé $o$ se fait de la façon suivante :

$$\forall s' \in S, b^{ind}_t(s') = \frac{\sum_s b^{ind}_{t-1}(s) \cdot T(s,a,s') \cdot O(s,a,s',o)}{\sum_{s''} \sum_s b^{ind}_{t-1}(s) \cdot T(s,a,s'') \cdot O(s,a,s'',o)}$$

On cherche, de la même façon, à calculer un **état de croyance relationnel** $b^r$, c'est à dire une distribution de probabilités sur l'ensemble des états joint relatifs. On rappelle que l'on dispose d'un ensemble d'observations $\Omega R$ que l'agent peut recevoir au sujet de son voisinage (c'est-à-dire sur l'état joint relatif actuel), et d'une fonction d'observation $OR$ associée. On pose de plus :

- $S(sr) = \{s \in S | \forall sr_i \in sr, (s,-) \in sr_i\}$ désigne l'ensemble des états individuels dans lesquels l'agent peut se trouver, lorsqu'il observe l'état joint relatif $sr$,
- $b^{ind}(sr) = \sum_{s \in S(sr)} b^{ind}(s)$ donne la probabilité qu'a l'agent d'être dans un état pour lequel l'état joint relatif $sr$ est possible,
- $a$ est l'action exécutée par l'agent et $o$ l'observation reçue sur le voisinage,
- $T^{sr}(sr,a,sr')$ est la probabilité de transition issue du MDP contenant l'état $sr$.

On définit alors l'équation utilisée pour mettre à jour l'état de croyance relationnel $b^r_{t-1}$ vers $b^r_t$ de la façon suivante :

$$b^r_t(sr') = \frac{OR(sr',o) \sum_{sr} b^{ind}(sr) \cdot b^r_{t-1}(sr) \cdot T^{sr}(sr,a,sr')}{\sum_{sr''} OR(sr'',o) \sum_{sr} b^{ind}(sr) \cdot b^r_{t-1}(sr) \cdot T^{sr}(sr,a,sr'')}$$

### 9.1.3 Avantages et inconvénients de l'approche

On dispose finalement d'un état de croyance individuel $b^{ind}$ et d'un état de croyance relationnel $b^r$, d'où l'état de croyance global $b = (b^{ind},b^r)$. On dispose également d'une fonction de $Q$-valeurs $Q^{ind}(b^{ind},a)$ et d'une autre fonction $Q^{coo}(b^r,a) = \sum_{sr} b^r(sr).Q^r(sr,a)$. On peut en déduire l'opérateur final (avec $0 \leq \alpha \leq 1$) définissant la politique $\pi$ :

$$\pi(b) = \underset{a \in A}{\operatorname{argmax}}\, \alpha.Q^{ind}(b^{ind},a) + (1-\alpha)Q^{coo}(b^r,a)$$

Le paramètre $\alpha$ utilisé ici permet de donner la priorité à une composante ou à l'autre. En général, on posera $\alpha = 0{,}5$ afin d'appliquer un comportement en moyenne. On pourra toutefois rendre l'agent plus égoïste ($\alpha > 0{,}5$) ou plus altruiste ($\alpha < 0{,}5$), selon le type de problème traité. Nous verrons par la suite (lors des expérimentations présentées) que cette approche permet une résolution assez rapide, chaque composante étant assez simple à traiter. D'un autre côté, on ne parvient pas à atteindre un comportement optimal, mais seulement de qualité « moyenne », et ce pour deux raisons en particulier.

1. Il y a, tout d'abord, le problème de l'approche Q-MDP. Les MDPs ayant été résolus selon ce principe, on aura tendance à être trop optimiste quant à l'évolution des interactions (cette approche pose en effet l'hypothèse qu'après l'action actuelle, l'agent aura une observabilité totale de son environnement, ce qui - à l'exécution - s'avère faux). Cela implique une erreur sur la fonction de $Q$-valeurs calculée.
2. Il y a ensuite le problème de l'impact des interactions, sur le comportement individuel. Cette approche considère uniquement l'impact des agents actuellement en interaction, et ce seulement à horizon 1 (c'est à dire après un pas de temps). Pourtant, on peut imaginer que les interactions auront un impact plus tard, à horizon $n > 1$, ou que d'autres agents intégrerons ou quitterons l'interaction. Il y a donc, là encore, une perte en qualité.

Cette approche est donc intéressante, puisqu'elle permet une résolution rapide des problèmes traités. Elle s'applique donc particulièrement bien aux problèmes de grande taille. D'un autre côté, il est dommage de ne calculer qu'une politique sous-optimale, même si celle-ci reste de bonne qualité. Pour cette raison, une seconde approche a été étudiée, certes plus coûteuse mais permettant une résolution quasi-optimale du problème.

## 9.2 Deuxième approche : POMDP Augmenté

L'approche réactive ne permet pas, comme nous l'avons expliqué, de calculer une politique optimale. Une autre approche a donc été proposée, que nous décrivons ici. L'idée est de générer un POMDP augmenté, décrivant à la fois la tâche de l'agent et l'évolution de ses interactions. Cette approche permet de résoudre les deux difficultés soulevées par l'approche réactive (optimisme et impact des interactions), mais sera nettement plus coûteuse en temps de résolution.

### 9.2.1 Génération du POMDP augmenté

L'objectif est la construction d'un **POMDP augmenté** $\langle S^{aug},A^{aug},T^{aug},R^{aug},\Omega^{aug},O^{aug}\rangle$, représentant à la fois la tâche de l'agent et l'évolution des interactions avec ses voisins. On dispose, pour rappel, d'un POMDP $\langle S,A,T,R,\Omega,O\rangle$ associé à la tâche de l'agent, d'un ensemble de MDPs $M^i = \langle S^i,A,T^i,R^i\rangle$ décrivant les $|C|$ classes d'interaction et d'un ensemble d'observations $\Omega R$ réalisables au sujet du voisinage, muni de la fonction d'observation $OR$. La construction du POMDP augmenté implique de générer plusieurs éléments :

- *l'ensemble des états* $S^{aug}$ - un état sera un couple devant décrire à la fois l'état individuel de l'agent, et les états relatifs de ses voisins, d'où $S^{aug} = S \times (\bigcup_{i=0}^{|C|} S^i)$. Il y a malgré tout une difficulté : selon l'état individuel de l'agent, certains états relatifs peuvent devenir inaccessibles. Il faut donc supprimer de $S^{aug}$ tous les couples $(s,sr)$ incohérents, c'est-à-dire ceux vérifiant $\exists sr_i \in sr \ tq. \ (s,-) \notin sr_i$,
- *l'ensemble des actions* $A^{aug}$ - il suffit ici de poser $A^{aug} = A$ puisque l'on ne raisonne que sur les actions individuelles, qu'il s'agisse de la tâche ou des MDPs,
- *la fonction de transition* $T^{aug}$ - on pose simplement [25], pour tout $(s,sr) \in S^{aug}$, $a \in A^{aug}$ et $(s',sr') \in S^{aug}$, l'équation $T^{aug}((s,sr),a,(s',sr')) = T(s,a,s') \times T^i(sr,a,sr')$ avec $M^i$ le MDP contenant l'état joint relatif $sr$,
- *la fonction de récompense* $R^{aug}$ - celle-ci se construit de la même façon que la fonction de transition, avec $R^{aug}((s,sr),a,(s',sr')) = R(s,a,s') + R^i(sr,a,sr')$,
- *l'ensemble des observations* $\Omega^{aug}$ - à chaque pas de temps, l'agent observe à la fois son état individuel et les relations avec ses voisins, d'où $\Omega^{aug} = \Omega \times \Omega R$,
- *la fonction d'observation* $O^{aug}$ - tout comme les transitions, on calcule simplement les probabilités d'observation par $O^{aug}((s,sr),a,(s',sr'),(o,or)) = O(s,a,s',o) \times OR(sr',or)$.

On dispose alors d'un POMDP complet, basé sur les actions individuelles de l'agent. Durant l'exécution de ce POMDP, l'agent observera son état ainsi que les variables indépendantes de l'environnement (les interactions avec les voisins, sur lesquels on n'a aucun contrôle). Cette représentation est beaucoup plus riche que l'approche réactive, puisque l'on intègre réellement les interactions dans les états individuels. D'un autre côté, la taille de l'espace d'états et de l'espace des observations augmente rapidement.

Prenons par exemple le problème de Thales, dans un environnement de petite taille (disons 10x10). On aura $|S| = 100$ et $|\Omega| = 9$. Admettons de plus que l'on utilise des interactions approchées, bornées à $m = 3$ voisins et que l'on observe $l = 2$ « couches » autour de l'agent (voir partie II, chapitre 7, section 7.3.2), on a alors $|SR| = 1 + 8l = 17$ d'où un total de $1 + \sum_{i=1}^{m} \binom{|SR|}{i} = 834$ états joints relatifs possibles et 16 observations relatives possibles. Ces deux composantes, indépendamment l'une de l'autre, sont relativement simple à traiter. Le POMDP augmenté en résultant, par contre, implique $|S^{aug}| \simeq 83400$ états et $|\Omega^{aug}| = 144$ observations, ce qui est nettement plus long à traiter.

25. On pensera à normaliser $T^{aug}$, pour que les probabilités de transition somment à 1.

### 9.2.2 Résolution, état de croyance et fonction de valeur

Le POMDP augmenté peut être résolu via n'importe quel solveur existant. Nous avons donc, là encore, utilisé une implémentation de l'algorithme SARSOP durant nos expérimentations, cette implémentation étant d'une efficacité satisfaisante et nous épargnant la tâche d'écrire un solveur (l'aspect « résolution d'un POMDP » n'étant pas l'objectif de cette thèse). On obtient donc, après résolution, une fonction de $Q$-valeurs basée sur les états de croyance (ou plus exactement un ensemble d'alpha-vecteurs, voir partie I, chapitre 2, section 2.2.2). Ainsi, durant l'exécution du problème, l'agent va :

1. observer son état individuel,
2. observer les états relatifs de ses voisins,
3. construire, à partir de ces différentes observations, une observation globale $o \in \Omega^{aug}$,
4. utiliser cette observation pour mettre à jour son état de croyance,
5. appliquer sa politique, selon l'état de croyance mis à jour.

La mise-à-jour de l'état de croyance se fait grâce à l'équation de mise-à-jour standard (rappelée dans la section précédente). La méthode POMDP augmenté permet alors de palier aux défauts de l'approche réactive :

- on considère en même temps les états individuels et les interactions durant le processus de planification, d'où une prise en compte de l'impact à long terme des interactions entre agents. Ici, raisonner sur les états joints (raisonnement classique, façon DEC-POMDP) n'aurait rien apporté de plus, puisque les voisins hors interaction n'ont, par définition, aucune influence (immédiate) sur les transitions ni sur les récompenses de l'agent,
- on raisonne directement sur les états de croyance, alors que l'approche réactive implique un raisonnement sur les états (approche Q-MDP). Pour cette raison, le problème de le surestimation de la fonction de Q-valeurs ne se pose pas.

Ainsi, on montrera sur un ensemble de benchmarks (chapitre 11) que l'on peut obtenir, via cette méthode de résolution, des politiques de très bonne qualité. D'un autre côté, cette approche s'avère - comme prévu - beaucoup plus coûteuse du point de vue des temps de résolution. Ainsi, certaines benchmarks ne peuvent tout simplement pas être résolus en un temps raisonnable (ou, du moins, pas avec le solveur de POMDP que nous avons utilisé). Le nombre d'observations possibles, en particulier, est central dans la complexité de résolution d'un POMDP. Il faudra donc veiller, si on veut utiliser cette approche, à maintenir un nombre d'observations raisonnable. Si cela s'avère impossible, il faudra se rabattre sur l'approche réactive. La question de la complexité est abordée plus en détails dans le chapitre suivant.

On dispose donc de deux approches pour la résolution du problème, le choix de l'approche se faisant selon la taille du problème traité. Il semble difficile de réduire d'avantage ces temps de résolution, la partie la plus coûteuse étant - dans les deux cas - la résolution du POMDP et reposant donc sur l'efficacité du solveur choisi. On a par contre supposé, jusqu'à présent, que l'on avait pu générer sans difficulté les MDPs associés aux clusters d'interactions. Pourtant, générer

ces MDPs peut rapidement devenir coûteux en temps, lorsque le nombre d'agents augmente (voir le chapitre suivant, traitant de complexité). Pour cette raison, nous avons identifié et exploité une propriété intéressante de ce processus de génération des MDPs : son aspect hautement parallélisable.

## 9.3 Apport du calcul parallèle

Le calcul parallèle permet, en théorie, de réduire drastiquement les temps de résolution de certains problèmes. Ainsi, toujours en théorie, si on dispose d'une machine pouvant traiter $n$ processus en parallèle (par exemple, une machine à $n$ cœurs), alors on pourra diviser par $n$ le temps de calcul global. Cette valeur reste cependant théorique, de nombreux autres paramètres étant à prendre en compte (le coût de la communication entre cœurs par exemple). Cette section n'a pas pour objectif d'étudier en profondeur la théorie du calcul parallèle, mais simplement de montrer comment ces techniques peuvent aider à la génération des MDPs associés aux clusters d'interactions. Nous commencerons donc par montrer comment paralléliser les algorithmes employés, après quoi nous fournirons quelques résultats obtenus lors de nos expérimentations.

### 9.3.1 Parallélisation des algorithmes employés

Nous cherchons à paralléliser les algorithmes employés lors de la génération des MDPs associés aux clusters d'interactions. Afin de générer ces MDPs, on considère chaque triplet ($sr$,$a$,$sr'$) possible et on calcule, à chaque fois, la probabilité de transition de $sr$ vers $sr'$ via l'action $a$, ainsi que la récompense associée. Chaque triplet représente donc un bloc de calculs à réaliser, indépendant des autres triplets. On peut alors envisager un traitement en parallèle de ces triplets, en les répartissant au sein des $n$ cœurs disponibles.

Il faut faire particulièrement attention, lorsque l'on cherche à paralléliser un algorithme, au problème de la communication entre processus. En effet, si ces processus doivent échanger des informations entre eux, alors ils devront certainement « s'attendre » mutuellement, l'un ne pouvant pas terminer son calcul sans le résultat de l'autre par exemple. On perd alors beaucoup de temps à ne rien faire, diminuant ainsi les performances de l'approche. Pour cette raison, il est important de minimiser les échanges entre processus. Dans notre cas, les calculs associés à un triplet sont complètement indépendants des autres triplets : il n'y a donc aucune communication entre processus, ce qui nous place dans un cadre idéal pour la parallélisation de nos algorithmes.

Nous avons donc parallélisé l'algorithme `construire` (algo. 4), dans lequel on calcule les probabilités de transition et les récompenses associées pour chaque triplet ($sr$,$a$,$sr'$). Il suffit, pour cela, de distribuer les triplets aux $n$ cœurs disponibles. On aurait pu se contenter ici de séparer l'ensemble des triplets en $n$ « paquets », un par cœur, mais cela aurait été peu judicieux. En effet, certains triplets sont plus longs à calculer que d'autres, et ce pour plusieurs raisons :

– certains états joints relatifs génèrent beaucoup plus d'instances que d'autres, et impliqueront donc d'avantage de calculs,

– selon le nombre d'agents impliqués dans $sr$, on aura plus ou moins d'actions jointes à prendre en compte,
– selon les relations mises en jeu dans $sr$, on aura plus ou moins de valeurs possibles pour *out* (voir le chapitre précédent).

On préfère donc attribuer à chaque cœur un nouveau triplet à calculer, dès que le cœur en question a fini de manipuler le triplet précédent. L'algorithme 8 décrit ce processus de distribution :

**Algorithme 8** : `construire-parallele()`, calcule les transitions et récompenses.

**Entrées** : $M = \langle S,A,T,R\rangle$ un MDP et $T^i$, $R^i$ provenant du cluster d'interactions associé
**pour chaque** $(sr,a,sr') \in S \times A \times S$ **faire**
  `// on attend qu'un cœur soit disponible :`
  **tant que** *tous les cœurs sont bloqués* **faire** attendre;
  $j \leftarrow$ un cœur non-bloqué;
  bloquer $j$;
  $p \leftarrow$ nouveau processus contenant ce bloc d'instructions :
  **début**
    `// on calcule les transitions et les récompenses :`
    **si** $T^i(sr,a,sr')$ *est défini* **alors** $T(sr,a,sr') \leftarrow T^i(sr,a,sr')$;
    **sinon** $T(sr,a,sr') \leftarrow$ transition$(sr,a,sr')$;
    **si** $R^i(sr,a,sr')$ *est défini* **alors** $R(sr,a,sr') \leftarrow R^i(sr,a,sr')$;
    **sinon** $R(sr,a,sr') \leftarrow$ recompense$(sr,a,sr')$;
    débloquer $j$;
  **fin**
  lancer l'exécution de $p$ sur le cœur $j$;
  `// remarque : l'exécution de cette boucle continue une fois le processus` $p$ `lancé.`

On peut, de la même manière, paralléliser `relier` (algorithme 5). La section suivante traite des gain en temps obtenus par cette approche.

### 9.3.2 Gains en temps attendus/obtenus

Nous avons implémenté les algorithmes parallélisés décrits ci-dessus, et observé l'évolution des temps de calcul en fonction du nombre de cœurs disponibles. Pour cela, nous avons exécuté notre solveur sur une machine de test, munie de 48 cœurs, en permettant initialement au solveur d'utiliser 1 de ces cœurs, puis 2, 3, etc. Nous avons réalisé ces tests sur une petite instance du problème de Thalès, nécessitant environ 3 heures pour être résolue avec 1 cœur. Nous nous intéressons ici uniquement au temps nécessaire pour la génération des MDPs associés aux clusters d'interactions. La figure 9.1 montre le temps observé pour la génération des MDPs, en minutes, selon le nombre de cœurs disponibles pour le solveur.

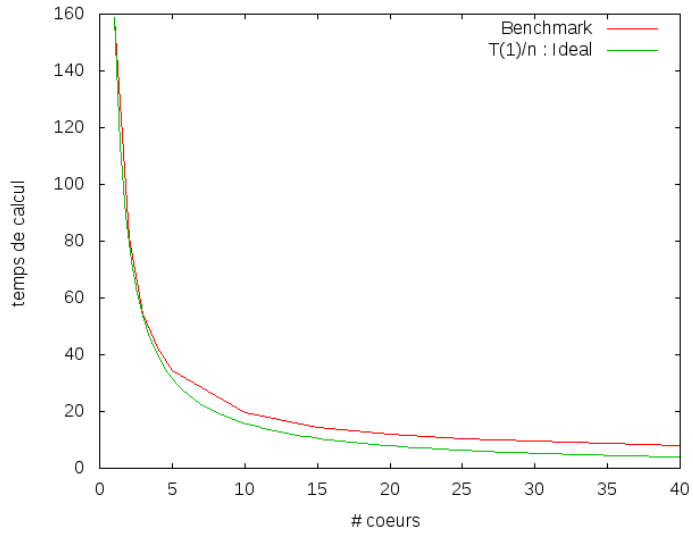


FIGURE 9.1 – Temps nécessaire à la génération des MDPs

Idéalement, on devrait diviser le temps par le nombre de cœurs. Ainsi, avec $T(n)$ le temps de résolution pour $n$ cœurs, la courbe idéale est donnée par $T(n) = T(1)/n$. En situation réelle, nous avons expliqué que la communication entre cœurs pouvait être coûteuse, d'où la fonction réelle $T(n) = C(n) + T(1)/n$, avec $C(n)$ le coût de la communication entre les $n$ cœurs. Afin de mieux distinguer la qualité de cette parallélisation, on peut s'intéresser à l'accélération du calcul, c'est à dire au gain en temps lorsque l'on ajoute un cœur supplémentaire (figure 9.2).

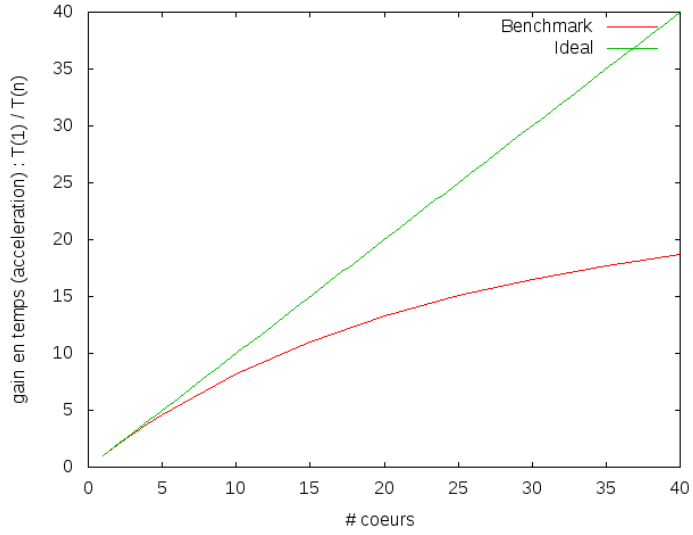


FIGURE 9.2 – Gains en temps

L'accélération se calcule simplement via la formule $gain(n) = T(1)/T(n)$. Une valeur de $x$ pour $n$ cœurs signifie donc que la résolution est $x$ fois plus rapide (idéalement, si on ajoute $n$ cœurs, la résolution doit être $n$ fois plus rapide). Ici, on peut constater que l'on s'écarte

sensiblement de la courbe optimale, lorsque l'on augmente le nombre de cœurs. Ce résultat s'explique par le fait que nos processus doivent accéder aux informations relatives aux clusters d'interactions, pour calculer les transitions et récompenses, et qu'ils se « gênent » probablement entre eux durant ces accès. Il semble possible d'améliorer l'implémentation de nos algorithmes pour éviter cet inconvénient. Afin de chiffrer la perte en qualité, on peut calculer l'efficacité des cœurs durant cette expérimentation (figure 9.3).

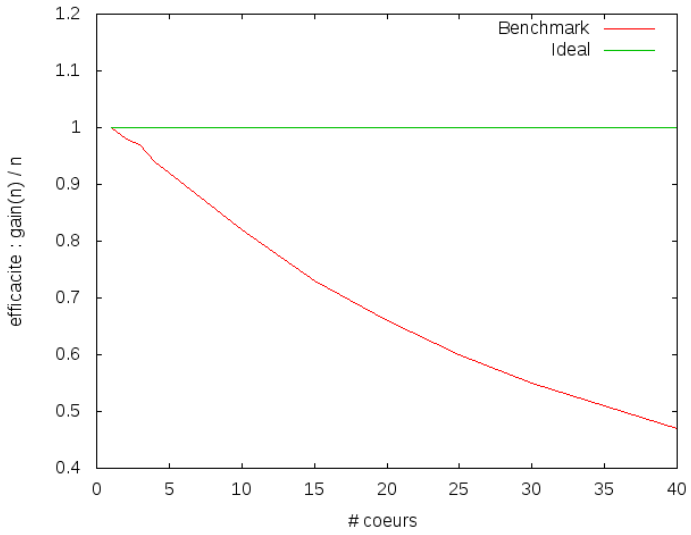


FIGURE 9.3 – Efficacité des coeurs

L'efficacité des cœurs se calcule en divisant le gain par le nombre de cœurs utilisés ($gain(n)/n$). Ainsi, une valeur de 1 signifie que l'on utilise à 100% la capacité de chaque cœur. Ici, on constate que l'efficacité diminue lorsque l'on augmente le nombre de cœurs : arrivé à 40 processus simultanés, on n'utilise plus les cœurs qu'à 50% de leurs capacités. Il semble donc inutile, au vu de notre implémentation, de travailler sur une machine offrant des capacités supérieures.

## Conclusion

Nous disposons maintenant d'algorithmes permettant la résolution du problème généré dans le chapitre précédent. Nous avons notamment vu comment, selon la dimension du problème traité, on pouvait choisir entre deux méthodes de résolution différentes. La première approche calcule des politiques de qualité moyenne, mais permet une résolution suffisamment peu coûteuse en temps pour traiter des problèmes de grande taille. À l'inverse, la seconde approche génère des politiques quasi-optimales, mais nécessite un temps de calcul plus important, la rendant inapplicable aux problèmes de dimension trop importante.

Nous avons également étudié comment optimiser ces algorithmes, notamment via le calcul parallèle. Les deux méthodes de résolution présentées ici reposent avant tout sur la résolution d'un POMDP : il semble donc difficile d'optimiser cet aspect - du moins dans le cadre de ces

travaux - le temps nécessaire à cette résolution dépendant simplement du solveur de POMDP utilisé. Par contre, la phase de pré-traitement des données, permettant de générer le problème à résoudre, peut tirer parti d'une approche parallèle. Ainsi, nous avons montré comment paralléliser ces calculs, puis les gains obtenus par cette approche parallèle.

Nous disposons maintenant d'une méthode « complète » de planification basée interactions, avec non-seulement un modèle pour la représentation du problème (DyLIM), mais également un ensemble d'algorithmes permettant le calcul d'une politique pour chaque agent. Il serait désormais intéressant d'étudier l'efficacité de cette méthode. Le chapitre suivant propose donc une analyse de complexité de nos algorithmes, et sera suivi d'un dernier chapitre décrivant un ensemble de résultats expérimentaux obtenus via notre méthode, sur plusieurs benchmarks représentatifs du domaine et provenant de la littérature.

# Chapitre 10

# Analyse de complexité

**Sommaire**



Les chapitres précédents ont introduit les algorithmes nécessaires au calcul d'une politique, pour un problème décrit via le modèle DyLIM. On s'intéresse désormais à la complexité en temps de ces calculs. Pour cela, on étudie trois grands points dans ce chapitre : la complexité de la phase de pré-traitement (générer les MDPs associés aux clusters d'interactions), la complexité de l'approche de résolution réactive (traiter séparément le POMDP et les MDPs) et, pour finir, la complexité de l'approche de résolution par POMDP augmenté.

## 10.1 Complexité de la génération du problème d'interaction

Nous l'avons mentionné précédemment : générer les MDPs associés aux clusters d'interactions peut s'avérer coûteux, en particulier lorsque l'on traite des problèmes impliquant un grand nombre d'agents. Ainsi, dans cette section, nous étudierons la complexité associée aux deux approches envisagées : l'approche classique, et celle reposant sur le calcul parallèle.

### 10.1.1 Approche classique

On commence donc par étudier la complexité des algorithmes de pré-traitement (générer les MDPs), dans l'approche classique (non-parallèle). On rappelle les paramètres à prendre en compte dans cette étude :

- $|SR|$ le nombre d'états relatifs possibles,
- $|A|$ le nombre d'actions,
- $I$ le nombre d'agents impliqués,
- $m$ le nombre maximal de relations considérées en même temps, au sein d'un état joint relatif (relations approchées),
- $j$ le nombre d'instances représentatives prises en compte, pour chaque état joint relatif.

L'algorithme 3, utilisé pour la génération des MDPs (via les algorithmes `construire` et `relier`), implique dans le pire cas le calcul successif de $K$ transitions (une par triplet ($sr$,$a$,$sr'$)) et des $K$ récompenses associées. Notons $C$ le coût, au pire cas, pour le calcul d'une transition et de la récompense associée. On a alors une complexité, pour cet algorithme, en $O(K \cdot C)$.

#### Évaluation de $K$

Afin d'évaluer $K$, il faut tout d'abord connaître le nombre d'états joints relatifs possibles. On a vu précédemment que, dans le cas des interactions exactes, il y avait $\frac{1-|SR|^I}{1-|SR|}$ états joints relatifs possibles. Ce nombre peut être borné par $O(|SR|^I)$. Dans le cas des interactions approchées, il y a $\sum_{i=0}^{m} \binom{|SR|}{i}$ états joints relatifs possibles. Ce nombre peut être borné par $O(|SR|^m)$. On peut donc évaluer $K$, au pire cas et selon l'approche choisie :

- *interactions exactes* : $K = O(|SR|^{2I} \cdot |A|)$,
- *interactions approchées* : $K = O(|SR|^{2m} \cdot |A|)$.

On remarque notamment que la complexité augmente exponentiellement en le nombre d'agents, lorsque l'on utilise des interactions exactes, mais que cette exponentielle disparaît dans les interactions approchées, le nombre d'interactions considérées étant borné par une constante (la complexité devient alors polynomiale en le nombre de relations $|SR|$).

#### Évaluation de $C$

Il nous faut maintenant évaluer $C$, c'est à dire le coût au pire cas du calcul d'une transition, et de la récompense associée. Tout d'abord, le calcul de la récompense peut se faire en même temps que celui de la transition. On pourra donc se contenter, pour évaluer $C$, d'estimer le coût du calcul d'une transition. Cette transition est générée via l'algorithme 6 qui calcule $p^{stay}$, $p^{out}$ et $p^{in}$ pour chaque valeur admissible de *out*, chaque partie de $sr$ de taille *out* et chaque partie de $sr'$ de taille $|sr| - out$. On évalue donc $C$ par un produit de quatre éléments :

1. *le nombre de valeurs possibles pour out* - au pire cas, la valeur de *out* peut varier de 0 (aucun agent ne quitte l'interaction) à $|sr|$ (tous les agents présents quittent l'interaction).

On a donc $O(|sr|)$ valeurs possibles. De plus, toujours au pire cas, on a $|sr| = O(I)$ pour des interactions exactes et $|sr| = O(m)$ pour des interactions approchées.

2. *le nombre de parties de sr* - il s'agit simplement de choisir *out* relations parmi les $|sr|$ existantes, c'est à dire $\binom{|sr|}{out}$ qui peut être borné au pire cas en $O(|sr|^{out})$. De plus, la valeur de $\binom{|sr|}{out}$ est maximale lorsque $out = (|sr| + 1)/2$, arrondi à l'entier supérieur. Ainsi, on borne le nombre de parties de *sr* en $O(|sr|^{\frac{|sr|+1}{2}})$, c'est à dire $O(I^{\frac{I+1}{2}})$ pour les interactions exactes et $O(m^{\frac{m+1}{2}})$ pour les interactions approchées.

3. *le nombre de parties de sr'* - ils s'agit, là encore, de choisir $|sr| - out$ relations parmi les $sr'$ existantes, donc $\binom{|sr'|}{|sr|-out}$. Là encore, on borne cette valeur par $O(I^{\frac{I+1}{2}})$ pour les interactions exactes et $O(m^{\frac{m+1}{2}})$ pour les interactions approchées.

4. *le calcul d'une transition* - il s'agit des coûts additionnés de $p^{stay}$, $p^{out}$ et $p^{in}$.

Si on se base sur les algorithmes fournis en annexe A, le calcul de $p^{stay}$ est relativement simple : pour chaque instance de *sr* et chaque instance de $sr'$, on calcule la probabilité que chaque relation $sr'_i$ soit générée par au moins un agent venant de *sr*, peu importe l'action exécutée. On a donc, avec $j$ le nombre d'instances considérées pour un état joint relatif donné, une complexité en $O(j^2 \cdot |sr'| \cdot |sr| \cdot |A|)$. Dans le cas des interactions exactes, cela donne $O(j^2 \cdot I^2 \cdot |A|)$ et dans le cas des interactions approchées, on a $O(j^2 \cdot m^2 \cdot |A|)$. De même, le calcul de $p^{out}$ se fait en $O(j^2 \cdot |sr| \cdot |A|)$ et le calcul de $p^{in}$ se fait en $O(j^2 \cdot |A|)$. On a donc, pour le calcul de ($p^{stay}$,$p^{out}$,$p^{in}$) un temps en $O(j^2 \cdot |A| \cdot (1 + I + I^2))$ (ou en $O(j^2 \cdot |A| \cdot (1 + m + m^2))$ pour les interactions approchées) ce qui se simplifie en $O(j^2 \cdot |A| \cdot I^2)$ (ou, respectivement, en $O(j^2 \cdot |A| \cdot m^2)$).

On a donc au pire cas, avec $m$ le nombre maximal de relations considérées dans un état joint relatif, et $j$ le nombre d'instances prises en compte :

- *interactions exactes* : $C = O(I \cdot I^{I+1} \cdot j^2 \cdot |A| \cdot I^2) = O(I^{I+4} \cdot j^2 \cdot |A|)$,
- *interactions approchées* : $C = O(m^{m+4} \cdot j^2 \cdot |A|)$.

### Complexité globale du pré-traitement des données

La complexité globale est, comme nous l'avons montré au début, en $O(K \cdot C)$. On obtient finalement les valeurs suivantes :

- *interactions exactes* : $O(|SR|^{2I} \cdot I^{I+4} \cdot j^2 \cdot |A|^2)$,
- *interactions approchées* : $O(|SR|^{2m} \cdot m^{m+4} \cdot j^2 \cdot |A|^2)$.

On identifie alors un facteur fort, qui est le nombre d'agents $I$ (ou le nombre de relations considérées $m$, dans le cas des interactions approchées). Ainsi, dans le cas des interactions exactes, la complexité de ces algorithmes augmente exponentiellement en le nombre d'agents, et devient rapidement problématique. Dans le cas des interactions approchées par contre, il suffit de fixer une valeur raisonnable pour le paramètre $m$. Prennent alors de l'importance le nombre de relations possibles $|SR|$, puis le nombre d'instances considérées et le nombre d'actions.

### 10.1.2 Approche parallèle

Dans le cas d'une résolution par calcul parallèle, un nouveau paramètre entre en jeu : $N$, le nombre de cœurs disponibles. La conséquence est immédiate, puisque l'on divise le travail entre ces $N$ cœurs. On obtient, pour les interactions approchées par exemple, une complexité en :

$$O\left(\frac{|SR|^{2m} \cdot |A|}{N} \cdot m^{m+4} \cdot j^2 \cdot |A|\right)$$

Nous avons vu, dans la section traitant de parallélisation (chapitre 9, section 9.3), que les choses n'étaient pas aussi simples. La parallélisation étant imparfaite, on ne divise pas vraiment le temps par $N$, mais plutôt par une fonction $f(N)$ dont les valeurs augmentent moins vite que $N$. Quoi qu'il en soit, l'ordre de grandeur donné ici reste correcte.

Nous avons vu précédemment que la complexité était en $O(K \cdot C)$, avec $K$ le nombre de triplets ($sr$,$a$,$sr'$) à traiter et $C$ le coût de traitement d'un triplet. Ici, nous avons choisi de séparer $K$ d'un côté, qui est divisé par $N$, et $C$ d'un autre côté, qui ne l'est pas. Cela a peu d'intérêt d'un point de vue mathématique, mais permet de mettre en évidence une réalité algorithmique : on ne diminue pas, à priori, le coût de traitement d'un triplet. Ainsi, la complexité donnée ici est vraie pour $1 \leq N \leq K$ (tout cœur supplémentaire serait inutile).

La complexité donnée pour $C$ reste cependant théorique. Celle-ci peut, en effet, être grandement allégée via une bonne implémentation, beaucoup de calculs étant répétés. Le calcul des transitions par exemple, d'une partie de $sr$ vers une partie de $sr'$, sera probablement répété de nombreuses fois (plusieurs états joints relatifs ayant des parties en commun). Cette complexité permet malgré tout d'identifier les paramètres critiques.

## 10.2 Complexité de la résolution par POMDP+MDP

Nous nous intéressons maintenant à la complexité des algorithmes de résolution. Nous étudierons pour commencer la complexité de l'approche réactive, impliquant de résoudre séparément le POMDP décrivant la tâche de l'agent, et les MDPs décrivant l'évolution de ses interactions. Deux points en particulier sont à étudier ici : la complexité de la résolution en elle-même, et le coût de maintien des états de croyance, lorsque l'on exécute la politique de l'agent.

### 10.2.1 Coût de la résolution

Le calcul de la politique, selon cette approche, se fait en deux temps : on résout tout d'abord le POMDP, puis l'ensemble des MDPs associés aux clusters d'interactions. La résolution du POMDP est une opération coûteuse, la complexité étant PSPACE-complète. Toutefois, la dimension de ce POMDP est en général suffisamment petite pour permettre une résolution en un temps raisonnable. La complexité de résolution des MDPs demande un peu plus de réflexion. Au vu de l'algorithme 7, deux éléments sont à prendre en compte :

– $B$ : le nombre de passes, c'est à dire le nombre d'exécutions de la boucle « tant que »,

– $P$ : la complexité d'une passe, c'est à dire le temps nécessaire pour résoudre chaque MDP, propager les valeurs et calculer les deltas.

Le coût total sera alors, simplement, en $O(B \cdot P)$.

**Nombre de passes**

On calcule, pour tout $MDP^i$ et tout état $s \in S^i$ (avec $\bar{S}^i$ l'ensemble d'états $S^i$ privé des états abstraits représentant les transitions vers d'autres MDPs), la fonction de valeurs :

– $\forall s \in \bar{S}^i, V^i(s) = \max_a R(s,a) + \gamma \sum_{s' \in \bar{S}^i} T(s,a,s').V^i(s') + \gamma \sum_{j \neq i} T(s,a,j). \max_{s' \in S^j} V^j(s')$,

– $\forall j \neq i, V^i(j) = \max_{s \in S^j} V^j(s)$.

Ainsi, $\gamma$ étant un réel positif ou nul strictement inférieur à 1, $V$ sera un réel fini (à condition que la récompense maximale atteignable soit également finie). On montre pour commencer que l'algorithme 7, utilisé pour calculer ces valeurs, converge vers un point fixe en un nombre fini d'itérations (c'est-à-dire un nombre fini de passes dans la boucle « répéter... jusqu'à ») :

1. Pour rappel, le principe de cet algorithme est d'alterner l'exécution du Value-Iteration sur chaque MDP, puis la propagation des valeurs, jusqu'à ce que la plus grande modification de valeur entre deux passes soit inférieure à un delta donné.
2. À l'issue de la première passe, toutes les fonctions de valeur ont convergé. Pour chaque MDP, la valeur maximale est positive ou nulle, car la valeur (ou récompense) initiale des états abstraits (les états représentant une transition vers un autre MDP) est nulle.
3. En conséquence, après propagation des valeurs maximales, on aura augmenté ou laissé identiques les valeurs (ou récompenses) des états abstraits, mais on n'aura pas modifié les valeurs/récompenses des autres états.
4. Donc, une fois que l'on aura fait re-converger le Value-Iteration, la valeur de chaque état aura augmenté ou sera restée la même. La valeur maximale de chaque MDP aura donc, de même, augmenté ou sera restée la même.
5. Ainsi, chaque passe fait soit augmenter les valeurs des états, soit stagner. Comme les valeurs de chaque état sont des réels finis, celles-ci ne peuvent pas augmenter indéfiniment. L'algorithme converge donc, nécessairement, vers un point où toutes les valeurs stagnent.

L'algorithme 7 s'arrête lorsque l'évolution de la valeur maximale de chaque MDP $i$ est inférieure à un $\epsilon$ donné (car si on propage les mêmes valeurs maximales qu'à la passe précédente, on calculera les mêmes valeurs pour chaque état, donc l'algorithme aura convergé). Cet algorithme revient donc à calculer les valeurs maximales à l'instant 1, puis à l'instant 2, etc., c'est-à-dire à résoudre pour chaque instant $t$ et chaque MDP $i$ l'équation :

$$V_{t+1}^{max}(i) = \max_{s \in S^i} \max_a \left[ R^i(s,a) + \gamma \sum_{s' \in \bar{S}^i} T^i(s,a,s').V^i(s,a,s') + \gamma \sum_{j \neq i} T^i(s,a,j).V_t^{max}(j) \right]$$

On crée l'ensemble $E^i = S^i \times A$. On peut alors définir, pour tout couple $e = (s,a) \in E^i$, la fonction $f(i,e) = R^i(s,a) + \gamma \sum_{s' \in \bar{S}^i} T^i(s,a,s').V^i(s')$. On obtient alors l'équation suivante :

$$V_{t+1}^{max}(i) = \max_{e \in E^i} f(i,e) + \gamma \sum_{j \neq i} T(i,e,j).V_t^{max}(j)$$

On retrouve donc la forme de l'équation de Bellman. Ainsi, l'algorithme 7 est similaire dans sa complexité à l'algorithme de Value-Iteration. On sait alors que, si on peut borner la fonction $f$ par une valeur $f^{max}$, le nombre de passes sera en $O(\frac{1}{1-\gamma} \cdot \ln \frac{f^{max}}{\epsilon(1-\gamma)})$, ce résultat étant issu de la littérature MDP. On peut borner cette fonction $f$ de la façon suivante :

- on a $f(i,e) = R^i(s,a) + \gamma \sum_{s' \in \bar{S}^i} T^i(s,a,s').V^i(s')$,
- on peut borner $R^i(s,a) \leq R^{max}$,
- de même, $V^i(s') \leq \frac{R^{max}}{1-\gamma}$,
- ainsi, au pire cas, $f(i,e) \leq R^{max} + \gamma \frac{R^{max}}{1-\gamma} = \frac{R^{max}}{1-\gamma}$

On a donc, finalement, un nombre de passes pour l'algorithme 7 borné en :

$$O\left(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)^2}\right)$$

**Complexité d'une passe**

La complexité d'une passe de calcul semble composée de trois éléments (résolution des MDPs, propagation des valeurs et calcul des deltas). En réalité, la propagation des valeurs et le calcul des deltas peut se faire simultanément à la résolution des MDPs, et ce de façon transparente pour la complexité. Il suffit donc, pour déterminer la valeur de $P$, de calculer la complexité de résolution des MDPs. Trois paramètres vont entrer en jeu ici :

- $S^i$ l'ensemble des états joints relatifs du $MDP^i$,
- $A$ l'ensemble des actions individuelles,
- $\gamma$ le facteur d'atténuation utilisé dans le Value-Iteration (voir algorithme 1, page 28).

On va donc résoudre séparément chaque $MDP^i$. La complexité d'un Value-Iteration est simple à calculer : on fait le produit du nombre de passes (en $O(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)})$ avec $R^{max}$ la plus grande récompense atteignable) par la complexité d'une passe (en $O(|A| \cdot |S|^2)$, avec $S$ les états du MDP). Ainsi, résoudre un $MDP^i$ se fait en $O(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)} \cdot |A| \cdot |S^i|^2)$. On doit résoudre chaque $MDP^i$, l'un après l'autre. La complexité globale s'obtient donc par une somme de ces complexités individuelles, c'est-à-dire en $O(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)} \cdot |A| \cdot \sum_i |S^i|^2)$. On peut finalement borner cette valeur par $O(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)} \cdot |A| \cdot (\sum_i |S^i|)^2)$. Ici, $|A| \cdot (\sum_i |S^i|)^2$ décrit le nombre de triplet $(sr,a,sr')$ possibles, calculé dans la section précédente (ce nombre était représenté par la valeur $K$). On obtient donc, finalement, une complexité en $O(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)} \cdot K)$. Il faut différencier cette complexité, selon le type d'interactions manipulées :

- *interactions exactes* : $O\left(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)} \cdot |SR|^{2I} \cdot |A|\right)$,
- *interactions approchées* : $O\left(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)} \cdot |SR|^{2m} \cdot |A|\right)$.

On obtient donc, là encore, une complexité exponentielle en le nombre d'agents pour les interactions exactes, et polynomiale en le nombre de relations possibles pour les interactions approchées. On prendra également en compte le paramètre $\gamma$, qui lorsqu'il tend vers 1 entraînera une croissance exponentielle du temps de résolution (en général, on constate qu'une valeur oscillant entre 0,1 et 0,01 est suffisante).

#### Complexité globale de la résolution

Comme nous l'avons montré initialement, la complexité totale du processus de résolution est en $O(B \cdot P)$ avec $B$ le nombre de passes et $P$ la complexité d'une passe. On obtient donc une complexité totale en $O(\frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)^2} \cdot \frac{1}{1-\gamma} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)} \cdot |SR|^{2?} \cdot |A|)$, le symbole $\ll ? \gg$ variant selon le type d'interactions. Or, $(1-\gamma)$ est compris entre 0 et 1, donc $(1-\gamma)^2 \leq (1-\gamma)$. On peut donc à nouveau borner cette complexité par $O(\frac{1}{(1-\gamma)^2} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)^2} \cdot |SR|^{2?} \cdot |A|)$. On obtient finalement, selon le type d'interactions manipulées, la complexité suivante :

- *interactions exactes* : $O\left(\frac{1}{(1-\gamma)^2} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)^2} \cdot |SR|^{2I} \cdot |A|\right)$,
- *interactions approchées* : $O\left(\frac{1}{(1-\gamma)^2} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)^2} \cdot |SR|^{2m} \cdot |A|\right)$.

Ainsi, là encore, le facteur fort est le $|SR|^{2?}$, impliquant une complexité exponentielle en le nombre d'agents lorsque l'on manipule des interactions exactes, polynomiale sinon.

### 10.2.2 Coût de maintien des états de croyance

Nous avons analysé, dans la section précédente, la complexité de résolution du problème. Qu'en est-il du coup d'exécution de la politique ainsi calculée ? En effet, contrairement aux problèmes totalement observables dans lesquels on peut se contenter d'appliquer la politique, il faut ici maintenir un état de croyance à jour, durant l'exécution du problème. Cette mise à jour peut alors s'avérer coûteuse, selon le problème traité. On a ici deux états de croyance à mettre à jour : l'état de croyance individuel (pour l'exécution du POMDP) et l'état de croyance relationnel (pour l'exécution des MDPs, via l'approche Q-MDP décrite en section 9.1.3, chapitre 9).

#### Mise-à-jour de l'état de croyance individuel

On rappelle que l'on calcule l'état de croyance $b_t$ à partir de $b_{t-1}$, lorsque l'agent a exécuté l'action $a$ puis reçu l'observation $o$ au sujet de son état individuel. On utilise pour cela l'équation :

$$\forall s' \in S, b_t^{ind}(s') = \frac{\sum_s b_{t-1}^{ind}(s) \cdot T(s,a,s') \cdot O(s,a,s',o)}{\sum_{s''} \sum_s b_{t-1}^{ind}(s) \cdot T(s,a,s'') \cdot O(s,a,s'',o)}$$

On a donc une complexité en $O(|S|+|S|^2)$, que l'on peut approcher par $O(|S|^2)$. Cette complexité, polynomiale en le nombre d'états, permet une mise-à-jour rapide de l'état de croyance.

**Mise-à-jour de l'état de croyance relationnel**

Plusieurs notations avaient été définies, pour la mise à jour de l'état de croyance relationnel :

- $S(sr) = \{s \in S | \forall sr_i \in sr, (s,-) \in sr_i\}$ désigne l'ensemble des états individuels dans lesquels l'agent peut se trouver, lorsqu'il observe l'état joint relatif $sr$,
- $b^{ind}(sr) = \sum_{s \in S(sr)} b^{ind}(s)$ donne la probabilité qu'a l'agent d'être dans un état pour lequel l'état joint relatif $sr$ est possible,
- $a$ désigne l'action exécutée par l'agent et $o$ l'observation reçue sur ses interactions,
- $T^{sr}(sr,a,sr')$ est la probabilité de transition issue du MDP contenant l'état $sr$.

On rappelle ensuite l'équation de mise-à-jour de l'état de croyance relationnel :

$$b_t^r(sr') = \frac{OR(sr',o) \sum_{sr} b^{ind}(sr) \cdot b_{t-1}^r(sr) \cdot T^{sr}(sr,a,sr')}{\sum_{sr''} OR(sr'',o) \sum_{sr} b^{ind}(sr) \cdot b_{t-1}^r(sr) \cdot T^{sr}(sr,a,sr'')}$$

La complexité de cette mise-à-jour est donc en $O(\bigcup_{i=0}^{|C|} S^i \cdot |S| + (\bigcup_{i=0}^{|C|} S^i)^2 \cdot |S|)$, avec $\bigcup_{i=0}^{|C|} S^i$ l'ensemble des états joints relatifs. Au pire cas, cette complexité est donc en :

- *interactions exactes* : $O(|SR|^{2I} \cdot |S|)$,
- *interactions approchées* : $O(|SR|^{2m} \cdot |S|)$.

Ainsi, là encore, la mise à jour de l'état de croyance peut être polynomiale en le nombre de relations et en la taille de l'espace d'états, si on utilise des interactions approchées. Si on utilise des interactions exactes par contre, la mise à jour de l'état de croyance nécessitera un temps exponentiel en le nombre d'agents, ce qui peut rapidement poser problème.

**Complexité global du maintient des états de croyance**

On obtient finalement une complexité globale en :

$$O(|S|^2 + |SR|^{2m} \cdot |S|)$$

Cette complexité passe à $O(|S|^2 + |SR|^{2I} \cdot |S|)$ si les interactions sont exactes. On a de plus l'avantage de séparer les états $S$ et les relations $SR$, ce qui nous permet de manipuler deux ensembles de taille raisonnable. On dispose donc d'une méthode rapide de mise-à-jour des états de croyance : cette propriété est indispensable, la mise-à-jour se faisant en temps réel, durant l'exécution du problème.

## 10.3 Complexité de la résolution par POMDP augmenté

La section précédente nous a permis d'analyser la complexité en temps pour la résolution (puis l'exécution) via la méthode réactive. De la même façon, on s'intéresse maintenant à la complexité de l'approche par POMDP augmenté.

### 10.3.1 Coût de la résolution

La résolution du problème, via l'approche par POMDP augmenté, se fait en deux temps : générer le POMDP, puis résoudre celui-ci. Il suffit donc de calculer $c_1$ la complexité en temps pour générer le POMDP augmenté, ainsi que $c_2$ la complexité de résolution. La complexité totale de l'approche par POMDP augmenté est alors obtenue par la somme $(c_1 + c_2)$.

#### Génération du POMDP augmenté

Afin de construire le POMDP augmenté, il suffit de prendre un-à-un tous les couples $(s,sr)$ (avec $s$ un état individuel et $sr$ un état joint relatif), toutes les actions $a$ et tous les couples $(s',sr')$. On calcule alors la probabilité de transition pour chacun de ces triplets $((s,sr),a,(s',sr'))$, ainsi que la récompense et la fonction d'observation associées. La calcul de ces transitions, récompenses et observations se faisant en temps constant, la complexité pour ce processus de génération est en $O(K.|S|^2)$, avec $K$ le nombre de triplets $(sr,a,sr')$ possibles (voir section 10.1.1). On obtient donc finalement, selon le type d'interactions considérées :

- *interactions exactes* : $O(|SR|^{2I} \cdot |A| \cdot |S|^2)$,
- *interactions approchées* : $O(|SR|^{2m} \cdot |A| \cdot |S|^2)$.

Ainsi, encore une fois, le complexité est exponentielle en le nombre d'agents ou polynomiale, selon le type d'interactions manipulées.

#### Résolution du POMDP augmenté

La complexité pour la résolution exacte d'un POMDP est PSPACE-complète. La valeur exacte pour cette complexité est en $O(|S|^2 \cdot |A|^{1+|\Omega|\cdot\frac{|\Omega|^{h-1}-1}{|\Omega|-1}})$, que l'on peut borner en $O(|S|^2 \cdot |A|^{|\Omega|^h})$. Le nombre d'observations $|\Omega|$ a donc une influence forte sur cette complexité, mais c'est l'horizon $h$ qui est le paramètre le plus coûteux. Ainsi, il est impossible de calculer une politique optimale pour un POMDP à horizon infini. On peut toutefois calculer une politique approchée, sacrifiant ainsi une part d'optimalité afin de maintenir des temps de résolution raisonnables. Diverses approches existent pour cela, dont celle sur laquelle repose l'algorithme SARSOP (approche « point-based ») utilisé pour la résolution de notre POMDP augmenté. Cet algorithme repose sur le principe suivant [Hsu *et al.*, 2007] :

1. L'espace des états de croyance possibles $B$ est, potentiellement, de taille infinie.
2. À partir d'un état de croyance initial $b_0$, on ne pourra atteindre qu'une partie $R(B)$ de l'espace des états de croyance.
3. Si l'agent suit une politique optimale $\pi^*$, il ne pourra atteindre qu'une partie $R^*(B)$ de l'espace des états de croyance.
4. On peut calculer un ensemble fini de points $C(\delta)$ permettant de « couvrir » $R^*(B)$, c'est à dire tel que tout état de croyance $b \in R^*(B)$ soit à une distance inférieure à un $\delta$ donné d'au moins un des éléments de $C(\delta)$.

5. Il devient alors possible de résoudre le POMDP à horizon infini, de façon approchée, sur l'ensemble $C(\delta)$ au lieu de l'espace $R^*(B)$.
6. Si on veut imposer que, pour tout état, l'erreur sur la fonction de valeur soit inférieure à un $\epsilon$ donné, il faut poser $\delta = \frac{(1-\gamma)^2\epsilon}{2\gamma R^{max}}$.

Finalement, si on suppose que l'on dispose de l'ensemble $C(\delta)$ avec $\delta$ la valeur définie ci-dessus, le POMDP peut être résolu en :

$$O\left(|C(\delta)|^2 + |C(\delta)| \log_\gamma \frac{(1-\gamma)\epsilon}{2R^{max}}\right)$$

Cette résolution génère un ensemble d'alpha-vecteurs : ceux-ci on été calculés à partir des éléments de $C(\delta)$, mais permettent ensuite de déterminer l'action à exécuter pour tout état de croyance $b \in B$ (d'où l'erreur $\epsilon$ admise lors de la résolution). Ici, l'élément le plus coûteux pour la complexité en temps est la taille de $C(\delta)$. Il faut donc être capable non-seulement de calculer efficacement cet ensemble de points, devant couvrir $R^*(B)$, mais également de minimiser la taille de cet ensemble. L'efficacité des méthodes existantes varie grandement selon la structure du problème résolu : plus il est facile de « discrétiser » l'espace $B$, et plus on pourra calculer un ensemble $C(\delta)$ de petite taille. Le nombre d'observations $|\Omega|$ est déterminant pour cette approche (car plus il y a d'observations possibles, et plus on peut générer d'états de croyance différents).

#### Complexité totale

On a finalement deux étapes complètement dissociées lors de la résolution par POMDP augmenté : générer le POMDP, puis résoudre celui-ci. Générer le POMDP augmenté peut se faire en temps polynomial si on manipule des interactions approchées, ou en temps exponentiel en le nombre d'agents si les interactions manipulées sont exactes. La résolution par contre, dépendra essentiellement de la structure du problème, ainsi que du nombre d'observations possibles.

### 10.3.2 Coût de maintien de l'état de croyance

Il parait plus simple de maintenir l'état de croyance via cette approche, que par l'approche réactive. En effet, on n'a ici qu'un seul état de croyance à maintenir, donnant une distribution de probabilités sur l'ensemble $S \times \bigcup_{i=0}^{|C|} S^i$ des états globaux. Ainsi, si on se base sur la section précédente, la complexité pour maintenir cet état de croyance est en $O(|S \times \bigcup_{i=0}^{|C|} S^i|^2)$, c'est-à-dire (selon le type d'interactions manipulées) en :

- *interactions exactes* : $O(|S|^2 \cdot |SR|^{2I})$,
- *interactions approchées* : $O(|S|^2 \cdot |SR|^{2m})$.

Il y a toutefois une difficulté à prendre en compte : la complexité en $O(|S|^2 \cdot |SR|^{2m})$ est nettement plus importante que celle en $O(|S|^2 + |SR|^{2m})$ de l'approche réactive (de même pour les interactions exactes). En effet, cette approche bénéficiait de la séparation de $S$ et de $SR$. Ici, en traitant ces deux ensembles comme un tout, on souffre d'un accroissement combinatoire pouvant rapidement rendre la mise-à-jour de l'état de croyance très coûteuse.

Malgré tout, il faut garder à l'esprit que cette complexité est un coût au pire cas. Dans les faits, une bonne implémentation permet de réduire drastiquement cette complexité. Par exemple, l'équation de mise-à-jour suppose une somme sur l'ensemble des états, mais on peut en réalité se contenter d'une somme sur les états de probabilité non-nulle dans $b_{t-1}$. On réduit alors considérablement le nombre de calculs à réaliser : les expérimentations effectuées montrent que l'on peut, en général, mettre à jour cet état de croyance en un temps suffisamment rapide pour qu'il soit négligeable durant l'exécution du problème. Ce nous permettra de nous attaquer à des applications réelles, telles que celle de Thales.

## Conclusion

Ce chapitre nous a permis de vérifier l'applicabilité des algorithmes présentés dans cette thèse, notamment d'un point de vue complexité en temps. Nous avons ainsi étudié la complexité des deux phases principales de résolution du problème, que sont la génération de données exploitables à partir du modèle DyLIM (phase de pré-traitement), puis le calcul d'une politique à partir de ces données (phase de résolution). Nous avons, entre autres, montré comment l'ensemble des algorithmes étudiés ici souffraient d'une complexité exponentielle en le nombre d'agents, dès lors que l'on manipulait des interactions exactes. Lorsque l'on s'autorise à manipuler des interactions approchées par contre, la complexité diminue et les algorithmes deviennent exploitables.

Ainsi, les algorithmes de pré-traitement ont une complexité en $O(|SR|^{2m} \cdot m^{m+4} \cdot j^2 \cdot |A|^2)$, avec $j$ le nombre d'instances représentatives considérées pour chaque état relatif, et $m$ le nombre maximal d'interactions prises en compte au même moment. Cette complexité permet des temps de calcul raisonnables, dès l'instant où on se limite à des petites valeurs pour $m$. Nous avons de plus montré qu'une approche par calcul parallèle permettait de diviser ces temps de calcul par le nombre de cœurs disponibles, permettant ainsi de traiter très rapidement les données.

Nous avons ensuite étudié la complexité des deux algorithmes possibles, pour la phase de résolution. l'approche par POMDP+MDP suppose une complexité en temps polynomiale, donnée par $O(\frac{1}{(1-\gamma)^2} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)^2} \cdot |SR|^{2m} \cdot |A|)$. On aura alors à maintenir à jour un état de croyance individuel, ainsi qu'un état de croyance relationnel, le tout se faisant en $O(|S|^2 + |SR|^{2m} \cdot |S|)$. Ainsi, là encore, il suffit de limiter les valeurs de $m$ pour permettre une résolution efficace.

L'approche par POMDP augmenté pour finir, nécessite de générer le POMDP puis de le résoudre. Générer le POMDP est peu coûteux, et peut se faire en $O(|SR|^{2m} \cdot |A| \cdot |S|^2)$. Résoudre ce POMDP par contre, peut devenir coûteux si le problème est trop complexe : il est difficile ici de faire une analyse de complexité, le coût de la résolution dépendant principalement de la structure du problème (plus il est facile de discrétiser l'espace des états de croyance, et plus il sera simple de résoudre le POMDP). Il faudra donc choisir entre la méthode par POMDP+MDP et la méthode par POMDP augmenté, selon le problème traité.

Le chapitre suivant présente un ensemble de Benchmarks, pour la validation de nos algorithmes (ainsi que de notre modèle). Nous autoriserons, dans les tests réalisés, la manipulation des interactions approchées (les interactions exactes rendant la plupart des problèmes de di-

mension ≪ réelle ≫ incalculables en un temps réaliste). Nous verrons notamment l'impact de la méthode de résolution employée (POMDP+MDP ou POMDP-augmenté) sur les temps de calcul ainsi que sur la qualité des politiques produites.

# Chapitre 11

# Expérimentations

**Sommaire**



Ce chapitre présente un ensemble d'expérimentations réalisées afin de valider notre approche. Ainsi, cette étude s'articule autour de trois aspects :

- *le positionnement* de notre approche relativement à l'existant (via un ensemble de tests relatifs à la qualité des politiques produites par nos algorithmes),
- *les performances* des algorithmes proposés, via une analyse des temps de calcul, afin de s'assurer qu'il est possible de passer à l'échelle sur des problèmes de taille « réelle »,
- *l'applicabilité* du modèle DyLIM, via une étude du problème de Thales, afin de valider notre approche par la résolution d'un problème complexe.

Le protocole expérimental suivi durant ces tests est décrit en annexe B.

## 11.1 Étude comparative : validation sur Benchmarks

Pour commencer, nous étudions dans cette section la qualité des politiques produites, et ce sur deux benchmarks. Nous commencerons par étudier le problème de la navigation dans des

couloirs, ce problème étant devenu un benchmark classique au sein de la communauté [Melo et Veloso, 2011].

### 11.1.1 Le problème de navigation

On s'intéresse à un problème de navigation en milieu fermé. Les figures 11.1 à 11.3 représentent trois environnements différents sur lesquels nos tests ont été réalisés.

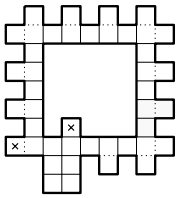

FIGURE 11.1 – ISR

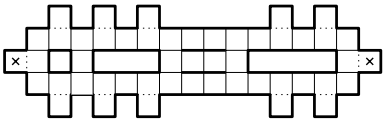

FIGURE 11.2 – MIT

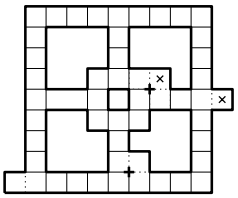

FIGURE 11.3 – PENTAGON

Le principe de ces benchmarks est le suivant :

- *Description du problème* : deux agents sont initialement placés sur une case tirée au hasard, telle que les agents ne soient pas sur la même case et qu'ils ne soient pas non plus sur les cases contenant un symbole 'X'. Leur but est alors de se rendre sur ces cases 'X' (un agent sur chaque 'X'). Une fois arrivé sur un des objectifs, l'agent termine sa mission.
- *États, actions, observations* : on représente l'état d'un agent par un triplet `(x,y,d)` avec `(x,y)` les coordonnées de l'agent et `d` sa direction (orienté vers le nord, l'est, le sud ou l'ouest). À tout moment, chaque agent peut exécuter une action au choix (avancer, tourner à gauche, tourner à droite ou ne rien faire). Finalement, les agents reçoivent des observations au sujet de leur environnement : ils n'observent pas directement leur état, mais uniquement les murs avoisinants (y'a t'il un mur au nord, à l'est, dans l'angle nord-est, etc. ?). Ces observations sont reçues avec une probabilité de 100% (il s'agit pourtant bien d'observabilité partielle, puisqu'on ne peut pas en déduire l'état exact de l'agent).
- *Interactions possibles* : les agents doivent faire en sorte d'éviter les collisions. Ainsi, les états relatifs seront les mêmes que pour le problème de Thales : voisin en `nord-D`, `est-D`, `nord-est-D`, etc. avec `D` la distance (couche 0, 1, 2, etc.), ainsi que la relation particulière `collision`. L'agent n'a pas une connaissance parfaite sur son voisinage : il se contente d'observer celui-ci, et de percevoir pour chaque direction (nord, est, nord-est, etc.) si des voisins sont présents (sans savoir combien) et, si oui, à quelle distance est le plus proche.
- *Transitions, récompenses* : lorsque l'agent choisit de tourner ou de ne rien faire, l'action réussit systématiquement. Lorsque l'agent avance par contre, il y a une probabilité qu'il dérive sur sa gauche ou sa droite, avec une probabilité de 0,05% pour chaque côté. Si l'agent avance en direction d'un mur, il reste sur place et reçoit une pénalité de -1. Chaque fois qu'un agent atteint un objectif, il reçoit une récompense de +10 et est immobilisé. Pour finir, un agent entrant en collision avec son voisin (ou atteignant un objectif déjà occupé)

reçoit une pénalité de -100. Cette pénalité n'est reçue qu'au moment de la collision (tant que les agents restent immobiles dans l'état collision, ils ne cumulent pas les pénalités).

Nous avons simulé l'exécution de ces benchmarks, afin d'estimer la qualité de nos algorithmes.

### 11.1.2 Résultats obtenus

Nous comparerons, dans la section suivante, les résultats obtenus par notre approche avec d'autres approches existantes dans la littérature. Pour cela, nous commençons dans cette section par présenter le détail des ADR obtenues (voir annexe B pour la définition de l'ADR) sur les instances ISR, MIT et PENTAGON du problème de navigation. Le tableau 11.1 décrit les résultats des tests réalisés.

| Instance | POMDPs indépendants | POMDP + MDP | POMDP augmenté | MMDP |
|---|---|---|---|---|
| ISR | -14.8 | 8.49 | 11.28 | 12.84 |
| MIT | -14.5 | 9.18 | 11.57 | 12.96 |
| PEN | -14.94 | 8.39 | 10.72 | 13.11 |

TABLE 11.1 – récompenses décomptées moyennes (ADR)

Ces expérimentations ont été réalisées conformément au protocole décrit en annexe B, en autorisant l'usage des interactions approchées. Pour chaque instance du problème de navigation, nous avons comparé les résultats obtenus selon quatre approches :

- *POMDPs indépendants* : il s'agit ici de calculer, pour chaque agent, une politique uniquement basée sur le POMDP décrivant la composante individuelle du problème. Cette approche permet d'ignorer complètement l'impact qu'ont les interactions sur l'agent lors du calcul de la politique. On obtient alors une borne inférieure pour l'ADR.
- *POMDP + MDP* : dans cette approche, on calcule la politique à partir du premier algorithme de résolution présenté dans le chapitre 9. La politique intégrant l'impact des interactions, l'ADR doit être supérieure à celle obtenue via les POMDPs indépendants.
- *POMDP augmenté* : il s'agit, là encore de calculer la politique à partir de l'algorithme correspondant (la deuxième approche de résolution présentée chapitre 9). Cette approche est, théoriquement, plus performante que l'approche par POMDP + MDP.
- *MMDP* : cette dernière approche consiste à résoudre le MMDP sous-jacent au problème (c'est à dire le MMDP obtenu en fournissant aux agents une observabilité complète sur leur état et sur l'état des autres agents), puis à simuler la politique ainsi calculée en offrant une observabilité totale aux agents. On obtient alors une borne supérieure pour l'ADR, l'observabilité totale permettant d'intégrer au mieux l'impact des interactions sur l'agent.

La première approche (*POMDP+MDP*) offre déjà des résultats satisfaisants. En comparaison, la borne inférieure est nettement plus basse : l'approche *POMDPs individuels* ne prenant pas les interactions en compte, les agents entrent régulièrement en collision, d'où l'ADR négative. L'intégration des interactions dans l'approche *POMDP+MDP* semble donc réussie, l'ADR posi-

tive reflétant l'absence de collisions. Cette intégration n'est toutefois pas optimale, puisque l'on ne considère ici que l'impact immédiat des interactions sur les décisions individuelles (et non l'impact à long terme). Imaginons le scénario suivant : l'agent s'approche d'un objectif $A$, mais s'aperçoit que cet objectif est déjà occupé par un voisin. Si on considère l'impact immédiat des interactions, l'agent ne doit pas se rendre sur cet objectif $A$ (car cela entraînerait une collision). Il n'a toutefois pas non plus intérêt à faire demi-tour pour se diriger vers l'objectif $B$ puisqu'il considère qu'à l'instant suivant, l'interaction aura disparue, et il qu'il pourra alors se rendre sur l'objectif $A$. Dans ce genre de situation, l'agent s'immobilise et la situation stagne ainsi, avec un seul des deux objectifs atteint.

La seconde approche (*POMDP augmenté*) est encore plus performante, les résultats étant cette fois quasiment égaux à la borne supérieure. Cette borne supérieure n'est probablement pas atteignable, puisqu'elle est obtenue par des agents bénéficiant d'une observabilité totale. Lorsque l'exécution du problème se fait en conditions réelles (donc avec observabilité partielle), comme c'est le cas pour les approches *POMDP+MDP* et *POMDP augmenté*, chaque agent ignore la position de départ des autres agents et pourra donc, parfois, se diriger vers un objectif puis s'apercevoir qu'un voisin est déjà sur l'objectif en question. L'agent devra alors faire demi-tour pour se diriger vers l'autre objectif possible, d'où une perte de temps et donc une ADR plus basse. Malgré tout, les interactions étant intégrées à long terme, les deux objectifs sont systématiquement atteints. Cette intégration semble donc excellente, mais cette approche ne pourra pas toujours être appliquée, comme nous le verrons par la suite (passage à l'échelle). Ici, les temps de résolution ont tous été inférieurs à 2 minutes, en incluant le temps nécessaire pour construire le problème d'interactions et le POMDP augmenté, ainsi que le temps de résolution avec SARSOP [Kurniawati *et al.*, 2008].

### 11.1.3 Résolution via les algorithmes existants

Nous avons montré, dans le chapitre 3 de ce document (partie I), que la littérature du domaine proposait deux approches principales pour la planification multiagent basée interactions :

- le modèle DPCL (ainsi que les algorithmes associés), permettant de considérer les agents comme complètement indépendants, hormis un certain nombre de tâches à réaliser sur lesquelles ils doivent se mettre d'accord,
- le modèle IDMG (ainsi, là encore, que les algorithmes associés), permettant d'intégrer le voisinage dans le raisonnement de l'agent, à condition que ce voisinage soit complètement observable (DEC-SIMDP, l'extension de DPCL, reposant sur les mêmes hypothèses).

Le modèle DPCL ne peut, quoi qu'il arrive, pas traiter les problèmes étudiés ici, les interactions impliquées étant trop imprévisibles pour pouvoir être représentées par un ensemble de tâches. En effet, formaliser ces interactions via des tâches à réaliser suppose que l'on peut prévoir à l'avance l'ensemble des interactions qui auront lieu (tâche 1 : tel agent croise tel autre agent à tel endroit, tâche 2 : tel agent croise...). Afin de prévoir cet ensemble, il faut pouvoir prévoir, là encore de manière exacte, les trajectoires que suivront chaque agent. Une telle prévision est

ici impossible : le succès des actions étant incertain, on ne peut pas garantir que les agents iront là où on le veut. Il serait peut-être possible de couvrir tous les cas possibles, en générant un ensemble de tâches de dimension exponentielle en le nombre d'agents, mais le problème serait impossible à résoudre au vu de sa dimension. En réalité, l'approche DPCL est plus adaptée à des problèmes où les interactions sont peu nombreuses et prévisibles. Nous ne chercherons donc pas à comparer cette approche avec la notre de façon plus poussée.

L'approche IDMG par contre, permet de traiter ces problèmes, dès l'instant où on ajoute une hypothèse d'observabilité complète sur le voisinage. Ces benchmarks sont d'ailleurs tirés d'un article traitant de DEC-SIMDP, l'extension de IDMG. Ainsi, en se basant sur l'ADR obtenue pour estimer la qualité d'une politique (voir annexe B), nous avons montré dans le chapitre 3 (pages 53 et suivantes) que l'approche DEC-SIMDP parvenait à calculer sur le problème MIT une politique quasiment idéale, obtenant une ADR égale à presque 100% de l'ADR optimale. On obtient ce pourcentage en transposant les ADR obtenues sur une échelle allant de 0 à 1, avec 0 la borne inférieure correspondant à l'ADR obtenue par des agents indépendants exécutant chacun un POMDP et 1 la borne supérieure correspondant à l'ADR obtenue par des agents bénéficiant d'une observabilité totale sur le problème, et exécutant donc le MMDP sous-jacent.

L'approche IDMG effectue toutefois un certain nombre d'hypothèses fortes, pour réussir à atteindre ces scores. Pour commencer, on suppose une observabilité complète sur le voisinage. Plus précisément, on définit un ensemble de zones d'interaction, chaque zone correspondant à un ensemble d'états contigus : si un agent et son voisin sont dans la même zone d'interaction, alors l'état du voisin est complètement observable. Si au contraire le voisin n'est pas dans une zone d'interaction, ou est dans une zone différente de celle de l'agent, alors son état n'est pas observable. Afin d'obtenir ces scores très élevés, on suppose qu'il n'existe qu'une seule zone d'interaction, occupant une grande partie de l'environnement. Dans le cas du problème ISR par exemple, cette zone contient 86% des états possibles (l'agent ne peut donc pas observer l'état de son voisin lorsque celui-ci se trouve dans les 14% restant). Si on réduit cette surface d'observabilité et que l'on pose une hypothèse plus réalise, l'ADR obtenue chute drastiquement (toujours dans le cas du problème ISR, si on considère un ensemble de 11 zones réduites, chacune couvrant 7% de l'environnement, l'ADR obtenue chute à environ 40% de l'ADR optimale). Le tableau 11.2 compare ainsi la qualité des politiques calculées pour l'approche IDMG bénéficiant d'une observabilité étendue, puis pour cette même approche lorsqu'on réduit l'observabilité, et finalement pour notre approche.

| Instance | IDMG (observabilité réduite) | IDMG (observabilité étendue) | DyLIM |
|---|---|---|---|
| ISR | 39,3% | 99,8% | 94,4% |
| MIT | 27,9% | 99,6% | 94,9% |
| PEN | 11,8% | 99,8% | 91,5% |

Table 11.2 – Pourcentage obtenu de l'ADR optimale

On constate ici que DyLIM permet de calculer des politiques quasiment optimales, même si celles-ci sont de qualité légèrement inférieure à celles calculées par IDMG. Pourtant, dans les tests réalisés avec notre approche, on ne fourni aux agents qu'une observabilité partielle sur le voisinage, contrairement aux tests réalisés avec IDMG dans lesquels on fourni une observabilité totale, à condition que le voisin se situe dans la zone d'interaction. Ainsi, si on compare nos résultats à ceux obtenus en limitant l'observabilité pour IDMG, notre approche devient nettement plus intéressante. Finalement, si on considère dans IDMG que l'observabilité sur le voisinage est partielle, peut importe où se trouve le voisin, alors on ne peut plus appliquer les algorithmes de résolution associés, et on doit se contenter de calculer des POMDPs indépendants.

L'approche DEC-SIMDP/IDMG souffre d'une dernière limitation, comparativement à notre approche : la méthode de résolution est centralisée, afin de traiter les interactions (contrairement à notre approche, où on peut se contenter d'une prévision sur le comportement des voisins). Ainsi, en raison de cette centralisation, la complexité combinatoire des algorithmes employés est importante, d'où une difficulté à passer à l'échelle en termes de nombre d'agents impliqués. À l'inverse, notre approche propose une résolution complètement décentralisée, chaque agent étant responsable du calcul de sa propre politique. Ainsi, comme notre analyse de complexité l'a démontré, le nombre d'agents n'influe pas sur les temps de calcul, la complexité dépendant uniquement de la taille du voisinage considéré. En raison de ces limitations, les benchmarks décrits dans la section suivante (traitant du passage à l'échelle) ne peuvent être résolus via l'approche IDMG.

## 11.2 Étude de performances : passage à l'échelle

Nous étudions maintenant la façon dont nos algorithmes se comportent, lorsque l'on augmente la taille du problème. Nous commencerons donc par présenter le benchmark utilisé durant ces tests, puis nous étudierons l'influence du nombre d'agents et du nombre de voisins considérés dans le voisinage, sur les temps de calcul et sur la qualité des politiques produites.

### 11.2.1 Choix du benchmark

Nous nous sommes basés sur un benchmark dérivé du problème de navigation ISR.

#### Description du problème

La figure 11.4 présente l'environnement open-ISR, utilisé durant nos tests. Les agents évoluent donc dans une version étendue du problème ISR présenté précédemment. On constate entre autres que le couloir de droite a été élargi, permettant ainsi des interactions plus complexes entre les agents. Un agent pourra par exemple être « cerné » par ses voisins, ce qui était auparavant impossible (la finesse des couloirs offrant, dans la version normale d'ISR, une relative protection aux agents). Cet environnement de test respecte les règles fixées pour les problèmes de navigation présentés précédemment. On rappelle notamment que les agents sont orientés, que chaque

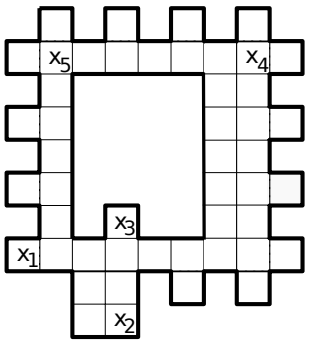


FIGURE 11.4 – Environnement de test : open-ISR.

agent doit se rendre sur un objectif au choix pour gagner une récompense (les objectifs étant représentés par des symboles 'x$_i$') et qu'un déplacement ne coûte rien à l'agent mais qu'une collision apporte une pénalité forte au groupe. On a de plus ajouté des interactions supplémentaires (au delà des simples interactions de « voisinage » géographique) : lorsqu'un agent constate qu'un voisin occupe un objectif $x_i$ donné, cela génère une interaction `objectif-x`$_i$`-complet`. On a donc $I$ interactions supplémentaires, une par objectif possible.

Nous réaliserons, dans cette section, des tests impliquant 2, 3 puis 4 agents. Nous ne pourrons plus calculer de MMDP sous-jacent, comme nous l'avions fait précédemment pour obtenir une borne supérieure à l'ADR obtenue. En effet, ces problèmes sont d'une taille trop importante pour permettre le calcul d'un tel MMDP (un test à 3 agents par exemple, impliquerait un MMDP à $192^3 = 7077888$ états joints). Nous nous contenterons donc de calculer la borne inférieure avec les POMDPs indépendants, puis de comparer à cette borne nos différentes approches. Nous utiliserons pour cela nos algorithmes de résolution parallèle (voir chapitre 9, section 9.3).

Pour finir, on constate que 5 objectifs sont indiqués sur la figure 11.4 (de $x_1$ à $x_5$). En réalité, on ne considère que les $I$ premiers objectifs. Ainsi, si le test réalisé implique $I = 3$ agents, seuls les objectifs $x_1$ à $x_3$ seront présents (les cases contenant les objectifs $x_4$ et $x_5$ étant alors considérées comme des cases « normales »). Attention cependant : cette numérotation ne signifie pas que l'agent $k$ doit se rendre sur l'objectif $x_k$ : chaque agent doit se rendre sur un des objectifs présents, peu importe lequel, le but final étant que chaque agent soit sur un objectif différent. Les agents ne connaissent donc pas, à priori, les objectifs choisis par les autres agents.

#### Avantages de ce problème

Nous avons choisi ce benchmark pour plusieurs raisons. Pour commencer, il a l'avantage de permettre des interactions complexes. Les agents étant initialement distribués aléatoirement dans l'environnement (sans connaître les positions initiales des autres agents), n'importe lequel de ces agent peut potentiellement se retrouver en interaction avec n'importe quel autre agent, et ce à n'importe quel endroit. De plus, les couloirs ayant été élargis, on peut imaginer des interactions impliquant jusqu'à 4 agents simultanément. Les situations de conflit (deux agents face à face devant passer dans un même couloir, par exemple) pourront être résolues via des

interactions complexes, les agents pouvant maintenant choisir de se croiser (alors que dans ISR, la seule solution était systématiquement qu'un agent recule jusqu'à pouvoir laisser passer son voisin). Ces interactions complexes impliqueront de plus de nombreuses possibilités à prendre en compte. Par exemple, que se passe t'il si l'agent que je croise dans un couloir échoue dans son action `avancer` et glisse vers l'emplacement où je me trouve ?

Ce benchmark a également l'avantage de permettre des tests impliquant un nombre variable d'agents. Il suffit, à chaque fois que l'on veut augmenter la complexité du problème, d'ajouter un objectif sur une des cases disponibles, et on peut alors ajouter un agent. On peut de même faire facilement varier la taille du voisinage considéré : un voisinage réduit ne conduit pas à une impossibilité pour l'agent de traiter le problème. Ici, un voisinage de taille 1 permettra uniquement à l'agent d'éviter les collisions immédiates (voisin en face de lui), tandis qu'un voisinage de taille 2 ou 3 permettra de prendre en compte les agents sur ses côtés et donc d'éviter les collisions par « dérapage » (échec de l'action `avancer`). Intuitivement, on voit ici qu'un voisinage d'une taille supérieure à 3 n'aurait que peu d'intérêt : cela permettrait de considérer les voisins présents derrière l'agent, mais celui-ci étant orienté et ne pouvant pas reculer, il ne pourra de toute façon pas générer de collisions avec de tels voisins.

Pour finir, on constate que les objectifs ont été disposés dans les zones permettant les interactions les plus complexes (principalement en bas à gauche et en haut à droite de la figure). L'objectif est ici de générer de telles interactions, et donc d'augmenter la difficulté du problème, lorsque plusieurs agents se dirigeront vers un même objectif.

### 11.2.2 Influence du nombre d'agents

On s'intéresse maintenant à l'influence du nombre d'agents sur les algorithmes employés. On analysera tout d'abord les temps de calcul, selon le nombre d'agents $I$ et le nombre maximal $k$ d'instances représentatives considérées pour un état joint relatif donné (voir la définition 30, page 94 pour la notion d'instance ainsi que l'annexe A, page 185 pour le choix des instances représentatives). On s'intéressera ensuite à la qualité des politiques produites, toujours selon ces deux paramètres. Pour l'instant, on considérera la totalité du voisinage durant le calcul de la politique (on pose $m = I - 1$, avec $m$ la taille du voisinage considéré).

#### Temps de calcul

Afin d'analyser l'impact du nombre d'agents sur les temps de calcul, on a réalisé un ensemble de tests sur le problème open-ISR. On a donc mesuré le temps de résolution à 2, 3 puis 4 agents : les valeurs relevées incluent le temps nécessaire au calcul des MMDPs associés à chaque cluster, ainsi que le temps nécessaire au calcul de la politique, après la phase de génération du problème. Ainsi, les courbes présentées en figure 11.5 présentent ces temps de calcul, en fonction de $I$ le nombre d'agents. On a superposé, pour permettre leur comparaison, les courbes pour différentes valeurs de $k$ (le nombre d'instances considérées). On a donc effectué ces tests pour $k = 10$, 50, 100, 500 et 1000, la dernière courbe ($k$ infini) correspondant au cas où on considère la totalité

des instances pour chaque état joint, peut importe leur nombre. L'échelle utilisée pour la courbe des ordonnées (temps de calcul, en secondes) est une échelle log. On a donc bien une croissance exponentielle des temps de calcul, lorsque l'on augmente le nombre d'agents impliqués.

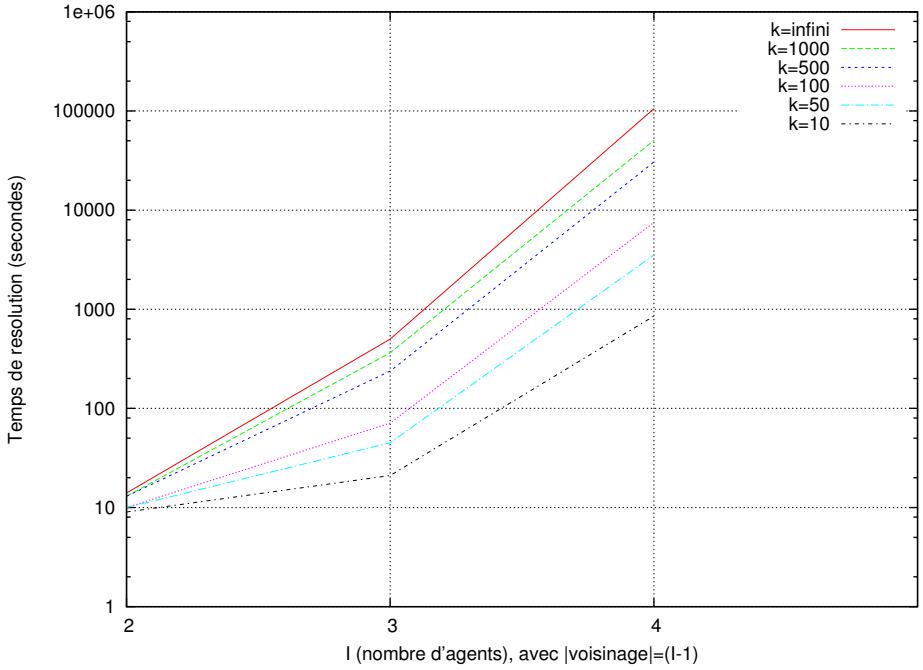


FIGURE 11.5 – Temps de résolution, selon le nombre d'agents.

Au delà de cette croissance exponentielle, prévisible au vu de l'analyse de complexité réalisée précédemment, on peut observer l'influence forte du paramètre $k$. Ainsi, L'instance impliquant 4 agents se résout en une trentaine d'heures pour un $k$ infini, alors qu'il suffit d'approximativement 15 minutes pour $k = 10$. Il serait donc intéressant d'étudier les ADR obtenues selon ces différentes valeurs de $k$, afin de déterminer quelle valeur est suffisante, et ainsi minimiser les temps de calcul.

### Qualité des politiques produites

On s'intéresse maintenant à la qualité des politiques produites, en termes d'ADR obtenue. Pour cela, on a simulé l'exécution des politiques précédemment calculées. Les courbes données en figure 11.6 présentent ces résultats, c'est à dire l'ADR obtenue en fonction de la valeur de $k$. On a superposé 3 courbes : la première correspond au cas $I = 2$, la suivante à $I = 3$ et la dernière $I = 4$. Il y a ici peu d'intérêt à comparer ces trois courbes entre elles : il s'agit en effet de trois problèmes distincts. Leur superposition sur une même figure est uniquement là pour faciliter la lecture des résultats.

L'axe des ordonnées (ADR obtenue) suit cette fois une échelle normale, tandis que l'axe des abscisses (valeur de $k$) suit une échelle log. Ainsi, la vitesse de progression de l'ADR ralentit lorsque $k$ augmente. Il faut, par exemple, passer de $k = 500$ à $k = 1000$ pour obtenir un gain en

ADR similaire à celui obtenu en passant de $k = 50$ à $k = 100$. Ainsi, augmenter la valeur de $k$ au delà de $k = 1000$ n'a quasiment plus d'impact sur la qualité des politiques produites.

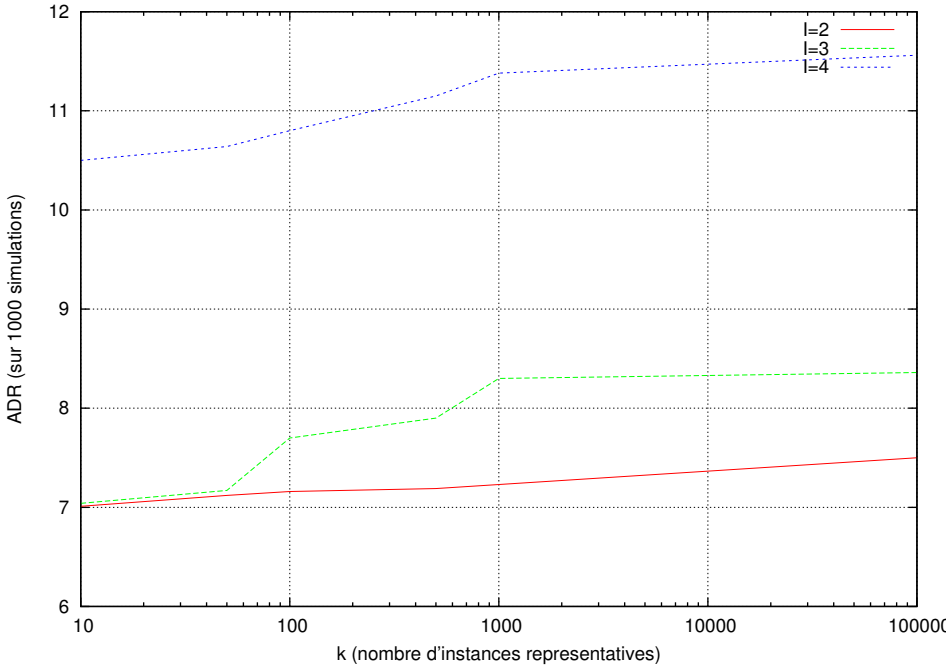


FIGURE 11.6 – ADR, selon le nombre d'instances représentatives.

On constate ici que les gains en ADR sont relativement faibles, lorsque l'on augmente la valeur de $k$. En utilisant des POMDPs indépendants pour résoudre ces instances de open-ISR, nous avons obtenu les valeurs suivantes :

- $I = 2$ : ADR=-9,38.
- $I = 3$ : ADR=-17,21.
- $I = 4$ : ADR=-3,4.

L'ADR obtenue par cette approche est meilleure pour $I = 4$ que pour $I = 2$ ou 3 : cela s'explique par la répartition des objectifs dans l'environnement. En effet, l'objectif 4 étant relativement excentré par rapport aux objectifs 1 à 3, moins de collisions sont générées. Quoi qu'il en soit, on constate que l'ADR obtenue pour $k = 10$ est déjà largement supérieure à l'approche par POMDP indépendants, et ce pour les trois instances testées ($I = 2$, 3 ou 4).

Ainsi, au vu de cet écart, le gain apporté par des valeurs supérieures de $k$ est négligeable. On a donc un impact très fort du paramètre $k$ sur les temps de résolution, et très faible sur la qualité des politiques produites. On pourra donc se contenter, à la résolution, d'un petit nombre d'échantillons, une valeur de $k = 10$ semblant ici suffisante.

### 11.2.3 Influence de la taille du voisinage

Nous nous intéressons désormais à l'influence de la taille du voisinage considéré, sur les temps de résolution et sur la qualité des politiques produites. Les tests réalisés ici concernent

une instance à 4 agents du problème open-ISR, dans laquelle on a fait varier la taille du voisinage considéré de $m = 0$ (aucun voisin pris en compte) à $m = 3$ (totalité des voisins pris en compte).

### Temps de calcul

Nous commençons donc par analyser l'influence de la taille du voisinage, sur les temps de résolution (figure 11.7).

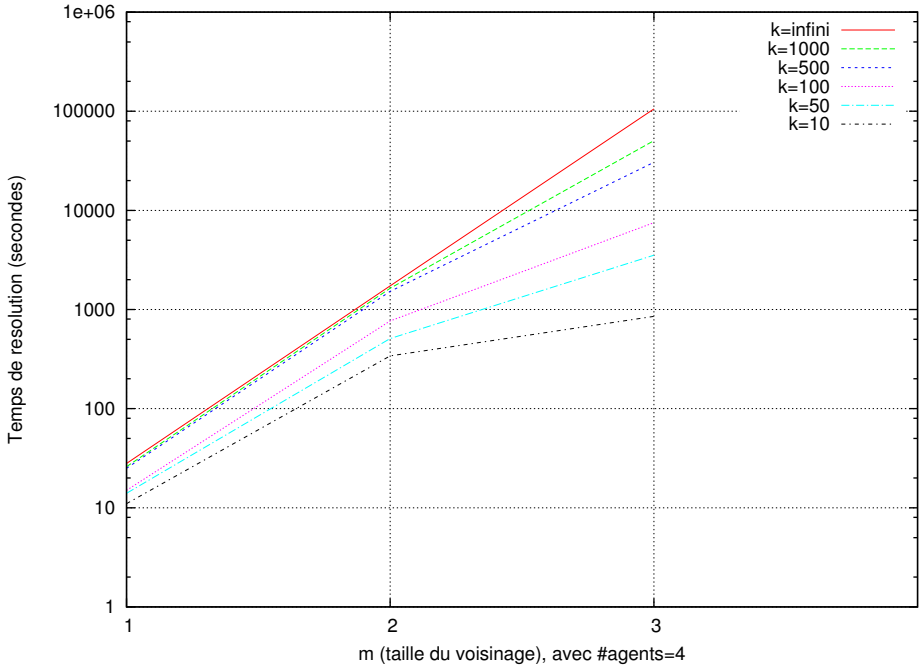


FIGURE 11.7 – Temps de résolution à 4 agents, selon la taille du voisinage.

On constate ici, conformément à ce qui était attendu suite à notre étude de complexité, une croissance exponentielle en la taille du voisinage. En théorie, le nombre d'agents impliqués dans le problème ne devrait avoir aucune influence sur le temps de résolution à condition qu'il y ait au moins $I = m + 1$ agents, ce qui se démontre facilement :

- la résolution du problème implique seulement de raisonner sur l'ensemble des états joints relatifs possibles,
- si on n'augmente pas la taille du voisinage, alors le nombre d'états joints relatifs n'augmente pas non plus, peut importe le nombre d'agents,
- la seule différence, lorsque $I$ augmente, repose donc dans la fonction de transition (la probabilité des états joints relatifs impliquant un nombre d'agents proche de $m$ va augmenter en fonction de $I$),
- cela ne doit, en toute logique, pas impacter les temps de calcul.

Pourtant, on constate ici que les temps de résolution pour un voisinage de taille 2 par exemple, sont légèrement supérieurs à ceux constatés dans la section précédente, pour 3 agents (et donc un voisinage de taille 2 également). Cela se justifie simplement : on a expliqué que l'on ajoutait

au problème une nouvelle interaction possible par objectif supplémentaire. Ainsi, il y a plus d'interactions possibles dans une instance à 4 agents (et donc 4 objectifs) que dans une instance à 3 agents, peu importe la taille de voisinage considérée, d'où le coût en temps supérieur. On constate par contre, lorsque l'on augmente le nombre d'agents sans rajouter d'objectif, que les temps de calcul ne changent effectivement pas (contrairement à la valeur des politiques calculées).

On constate également, là encore, une forte influence du paramètre $k$ (nombre d'instances considérées). Ainsi, les valeurs basses de $k$ permettent une résolution nettement plus rapide, en particulier lorsque l'on considère un voisinage de grande taille. Il serait donc, là encore, intéressant d'étudier l'impact de ce paramètre sur la qualité des politiques produites.

### Qualité des politiques produites

On s'intéresse désormais à la qualité des politiques produites, lorsque l'on augmente la taille du voisinage considéré. Ainsi, la figure 11.8 présente les résultats obtenus en termes d'ADR, sur une instance d'open-ISR impliquant 5 agents. Le cas $m = 0$ signifie que les agents ne prennent en compte aucun voisin : on retombe alors sur la borne inférieure, donnée par les POMDPs indépendants.

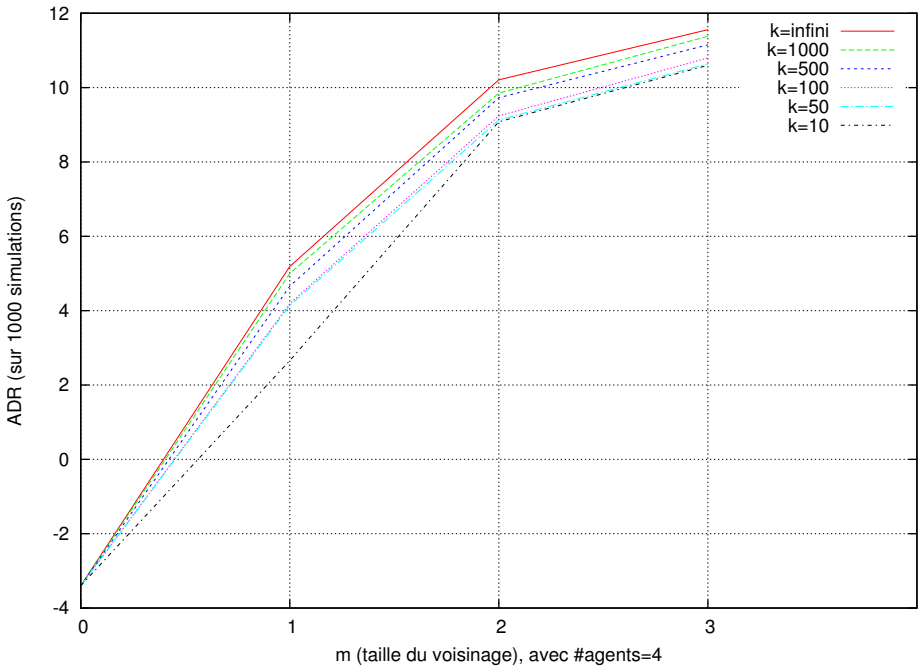


FIGURE 11.8 – ADR, selon la taille du voisinage.

Ici, plusieurs éléments sont remarquables. Pour commencer, on constate à nouveau que l'influence du paramètre $k$ sur la qualité des politiques produites est minime. Ainsi, là encore, les valeurs basses de $k$ semblent les plus intéressantes.

Un second élément apparaît ici : l'ADR obtenue semble converger vers un palier, au fur et à mesure que l'on augmente $m$. Ainsi, on gagne environ 8 points sur l'ADR en passant d'un

voisinage de taille zéro à un voisinage de taille 1, puis 5 points en passant de 1 à 2 et à peine plus d'un point en passant à un voisinage de taille 3. Il serait donc intéressant, pour confirmer cette hypothèse, de résoudre une instance de open-ISR dans laquelle on considérerait un voisinage de taille supérieure à 3.

### Voisinage suffisant

Nous terminons donc cette section par une série d'expérimentations réalisées sur une instance de open-ISR impliquant 5 agents. Nous nous contenterons ici d'observer l'évolution de l'ADR pour le cas où $k = 10$, ayant montré précédemment le faible impact de $k$ sur la qualité des politiques produites. La figure 11.9 présente les résultats ainsi obtenus.

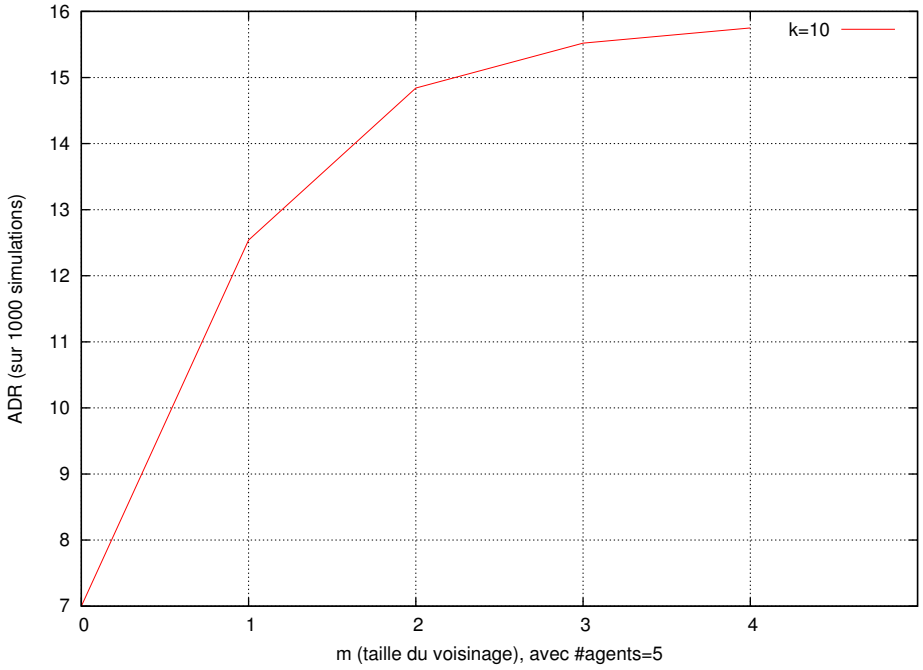


FIGURE 11.9 – saturation de l'ADR, à partir d'un voisinage 3.

On constate ici que l'ADR obtenue, pour un voisinage de taille 4, est quasiment égale à celle obtenue pour un voisinage de taille 3. L'intuition exposée précédemment semble donc se confirmer : l'ADR converge vers une valeur donnée, lorsque l'on augmente la taille du voisinage. La raison intuitive de ce phénomène est assez simple : lorsque la taille du voisinage est bornée, l'agent considère en priorité les interactions les plus critiques. Ainsi, passé un certain seuil, augmenter la taille du voisinage permet seulement à l'agent de prendre en compte des interactions n'ayant quasiment aucun impact sur ses transitions ni sur ses récompenses, d'où la convergence de l'ADR obtenue.

## 11.3 Étude d'applicabilité : le problème de Thales

Nous terminerons ces expérimentations par une analyse du problème de Thales. L'objectif de cette section est donc de montrer comment ce problème peut être traité via notre approche.

### 11.3.1 Problème traité

Nous nous intéressons désormais à la formalisation du problème de Thales, tel qu'il a été défini dans la partie II, chapitre 5. Ainsi, l'objectif est de permettre à un groupe de véhicules autonomes de former puis de maintenir un convoi, durant un déplacement en environnement extérieur. Cet environnement pouvant potentiellement évoluer au cours de la mission, les agents devront modifier en conséquence la structure du convoi, afin de toujours maintenir une structure optimale. Plusieurs points sont à préciser ici, quant aux choix que nous avons effectués relativement à la spécification du problème :

- on suppose qu'un agent leader existe dans le convoi, qui aura pour mission de guider les autres agents au sein de l'environnement. Ce leader ne sera pas contrôlé par notre approche, mais par un opérateur extérieur. Ainsi, les agents auront simplement à évoluer autour de ce leader afin de maintenir la structure du convoi, sans se soucier de la trajectoire suivie par le groupe. Pour cela, ils pourront choisir d'accélérer (pour dépasser le leader) ou de se déporter sur leur gauche ou leur droite. On suppose qu'un agent n'a pas le droit de rouler dans le sens inverse du leader, mais il peut ralentir pour se laisser dépasser.
- Les agents se déplacent au sein d'un canal de navigation. La dimension de ce canal peut évoluer au cours du temps, variant d'un simple chemin à une large zone ouverte. En dehors de ce canal, les agents peuvent malgré tout se déplacer, mais sont considérablement ralentis. Si un agent s'éloigne trop du leader, il est considéré perdu, et quitte la mission.
- Toute collision entre agents ralentit considérablement les deux agents concernés, et toute collision avec le leader provoque un échec de la mission.
- Pour finir, un agent peut observer avec certitude la largeur du canal de navigation, mais n'a qu'une vision locale des positions des autres agents ainsi que de sa propre position relativement au leader.

Bien entendu, au delà de ces points particuliers, on respecte également les contraintes propres au domaine : incertitude sur l'exécution des actions, absence de communication et planification complètement distribuée. Ce problème correspond parfaitement à notre approche, puisque l'on manipule des interactions locales (aussi bien géographiquement que temporellement) et imprévisibles. Nous pouvons donc décrire ce problème via notre modèle. L'aspect « réel » de ce problème implique toutefois certaines contraintes fortes. En particulier : il faut garantir le succès de la mission. Cela passe, dans notre cas, par deux points :

1. s'assurer, en modélisant le problème, qu'aucun agent ne puisse choisir de quitter la mission,
2. prendre en compte un voisinage de taille suffisamment grande pour éviter toute collision.

La section suivante montre la façon dont on peut modéliser ce problème, via DyLIM.

### 11.3.2 Formalisation du problème

Voyons maintenant comment formaliser ce problème via DyLIM. Il suffit ici d'instancier $\langle\langle S,A,T,R,\Omega,O\rangle;\langle SR,\Omega R,OR,C\rangle\rangle$ avec $C = \{C^1,\ldots,C^{|C|}\}$ et $C^i = (S^i,T^i,R^i)$.

#### Choix de paramètres

Avant de formaliser le problème de Thales, il convient de décider de certains paramètres. Les valeurs données pour ces différents éléments sont des choix que nous avons effectué pour l'implémentation du problème, mais n'ont pas d'influence sur l'applicabilité de notre approche (hormis peut-être un impact sur la dimension du problème à traiter, et donc sur les temps de résolution). Ainsi, on pourra modifier ces valeurs selon le problème que l'on traite.

1. *Taille de l'environnement* - nous avons expliqué que les agents devaient se positionner relativement au leader. Ainsi, il suffit de considérer une fenêtre de navigation autour de ce leader : on considère que tout agent en dehors de cette fenêtre sera trop éloigné du leader pour l'observer, et aura donc quitté le convoi. On définit donc l'environnement comme un damier de 100 ≪ cases ≫ (10 de large et 10 de long), au centre duquel se trouve le leader. On considère alors que l'axe $y$ représente les côtés de l'agent, tandis que l'axe $x$ représente la direction dans laquelle il se déplace.

2. *Largeur du canal de navigation* - nous avons expliqué que la dimension du canal de navigation pouvait varier, au fur et à mesure que le convoi se déplace. On suppose ici que, au mieux, la totalité de l'environnement est dans le canal et qu'au pire, seules les ≪ lignes ≫ du centre ($y = -1$ et $y = 1$) appartiennent à ce canal. De plus, comme le leader se maintient au centre du canal, on suppose qu'il y a autant de place pour circuler à sa gauche qu'à sa droite. Ainsi, avec $l$ le nombre de lignes occupées sur le damier par le canal, on a de chaque côté du leader $(l/2)$ lignes appartenant au canal. Ainsi, avec $(x,y)$ les coordonnées de l'agent, celui-ci est hors du canal si $|y| > (l/2)$.

3. *Transitions et récompenses* - on définira les probabilités de transition de telle sorte qu'un canal de navigation très petit ou très large soit rarement observé, mais que l'on oscille entre des dimensions moyennes. De plus, afin d'augmenter la difficulté du problème, on fera en sorte que les actions puissent échouer, notamment en faisant dériver l'agent sur les côtés. En ce qui concerne les récompenses, on attribuera une petite récompense lorsque tout va bien, mais une pénalité importante pour les situations que l'on veut éviter à tout prix (principalement, agent quittant le convoi, et collision avec le leader).

4. *Nombre d'agents et taille du voisinage* - ici, nous avons choisi de manipuler 10 agents, plus le leader que l'on ne contrôle pas. En réalité, le nombre d'agents n'a aucun impact sur l'implémentation du problème ou sur les temps de résolution (comme nous l'avons montré dans les expérimentations précédentes), mais il faut par contre prévoir un environnement suffisamment grand pour permettre aux agents d'évoluer au sein de celui-ci. Ainsi, 10 agents évoluant sur un damier de 100 cases semble un chiffre raisonnable. Quoi qu'il en

soit, nous considérerons un voisinage de 3 agents, ce choix étant basé sur les expériences menées sur open-ISR (ce problème étant très similaire à celui de Thales, en ce qui concerne les interactions entre agents).

Une fois ces valeurs définies, on peut instancier les deux composantes.

**Instanciation du problème individuel $\langle S,A,T,R,\Omega,O\rangle$**

Le problème individuel consiste, pour l'agent, à éviter toute collision avec le leader, à rester dans le canal de navigation et à ne pas trop s'éloigner du leader (pour ne pas quitter le problème). On instancie pour cela les ensembles $S$, $A$ et $\Omega$ ainsi que les fonctions $T$, $R$ et $O$.

*L'ensemble des états individuels* $S$ : il est inutile de raisonner sur les coordonnées absolues de l'agent, celui-ci ayant seulement à se positionner vis-à-vis du leader (qui se maintient lui-même au centre du canal de navigation). Ainsi, un état doit simplement décrire la position de l'agent, relativement au leader, ainsi que la largeur actuelle du canal de navigation. On en déduit donc l'ensemble $S = \{(x,y,l)|x,y \in [-5,0[\cup]0,5],2 \leq l \leq 10\}$. On génère ainsi 900 états individuels, couvrant 9 largeurs possibles pour le canal de navigation. La coordonnée (0,0) correspond alors à la position du leader, mais on supposera par sécurité (afin d'éviter tout risque de collision) que celui-ci occupe les 4 cases $(-1,-1)$, $(-1,1)$, $(1,-1)$ et $(1,1)$.

*L'ensemble des actions possibles* $A$ : on suppose que l'agent peut `accélérer` (ce qui le fera avancer d'une case relativement au leader, sa vitesse revenant à la normale après un pas de temps), `ralentir` (ce qui le fera reculer d'une case relativement au leader), ne `rien` faire (c'est-à-dire rouler à vitesse normale) et se déporter sur sa `gauche` ou sa `droite`.

*L'ensemble des observations possibles* $\Omega$ : une observation décrira la position de l'agent relativement au leader et au canal de navigation. On aura donc $\Omega = \{(l,p,c)|2 \leq l \leq 10, p \in \{fg,fd,ag,ad\},c \in \{0,1\}\}$, avec $l$ la largeur du canal, $p$ la position relative (en face-gauche, face-droite, arrière-gauche ou arrière-droite du leader) et $c = 1$ si l'agent évolue dans le canal.

*La fonction de transition* $T$ : on définit les transitions entre états, selon l'action exécutée. À chaque pas de temps, la largeur du canal de navigation peut changer :

- pour $l = 10$, la probabilité que le canal rétrécisse (vers $l = 9$) est de 0,4 ;
- pour $6 \leq l < 10$, la probabilité que $l$ varie de 1 est de 0,2 ;
- pour $2 < l < 6$, la probabilité que $l$ varie de 1 passe à 0,3 ;
- pour $l = 2$, la probabilité que $l$ augmente de 1 est de 0,5.

Il faut également considérer, après chaque action, l'évolution de la position de l'agent relativement au leader.

- Action `rien` : l'agent avançant à la même vitesse que le leader, sa position relative ne change pas. Si il évolue en dehors du canal, sa coordonnée $y$ peut augmenter ou diminuer de 1, avec une probabilité de 0,05 pour chaque côté.
- Action `accélérer` : si l'agent évolue au sein du canal, sa coordonnée $x$ augmente de 1. Sinon, non-seulement sa coordonnée $x$ augmente de 1, mais sa coordonnée $y$ peut également augmenter ou diminuer de 1, avec une probabilité de 0,1 pour chaque côté.

– Action `ralentir` : la coordonnée $x$ de l'agent diminue de 1.
– action `gauche` (ou, respectivement, `droite`) : la coordonnée $y$ de l'agent diminue (ou, respectivement, augmente) de 1 avec probabilité 0,95 et de 2 avec probabilité 0,05.

On posera donc pour tout état $s = (x,y,l)$, tout autre état $s' = (x',y',l')$ et toute action $a$, la probabilité $T(s,a,s') = P_{deplacemet}((x,y),a,(x',y')) \times P_{largeur}(l,l')$ avec $P_{deplacement}$ la probabilité que l'agent passe de la position décrite dans l'état $s$ à celle décrite dans l'état $s'$ lorsqu'il exécute $a$, et $P_{largeur}$ la probabilité que le canal de navigation évolue d'une largeur $l$ à une largeur $l'$.

*La fonction de récompense* $R$ : ici, la récompense dépend principalement de l'état. Ainsi, pour tout état tel que l'agent est en dehors du canal de navigation ($|y| > (l/2)$), la récompense est de -5. Pour tout autre état, la récompense est de 0. Il y a également deux cas particuliers : si l'agent se trouve sur un des quatre états correspondant au leader, la mission échoue, et si l'agent dérive sur sa gauche alors qu'il est en $y = -5$ ou qu'il dérive sur sa droite alors qu'il est en $y = 5$, alors il transite vers un état « echec » qu'il ne pourra plus quitter et qui apporte systématiquement une pénalité de -5 (sans mettre fin à la mission).

*La fonction d'observation* $O$ : l'observation $(l,p,c)$ reçue donne une information exacte en ce qui concerne la largeur du canal. En ce qui concerne la position $p$, l'information est exacte mais ne permet pas de déduire directement les coordonnées $(x,y)$ de l'agent. De plus, pour tout état $s = (x,y,l)$, on observera $(l,p,0)$ si $|y| > (l/2)$, et $(l,p,1)$ sinon.

Il faut maintenant instancier le problème d'interactions, après quoi nous disposerons d'un problème complet pouvant être traité via nos algorithmes.

### Instanciation du problème d'interactions $\langle SR,\Omega R,OR,C\rangle$

Le problème d'interaction consiste à éviter les collisions entre agents, et maintenir la structure de convoi voulue. On a décrit, dans les chapitres précédents (voir notamment le chapitre 6, page 94), les interactions possibles au sein de ce problème (voir le rappel en figure 11.10).

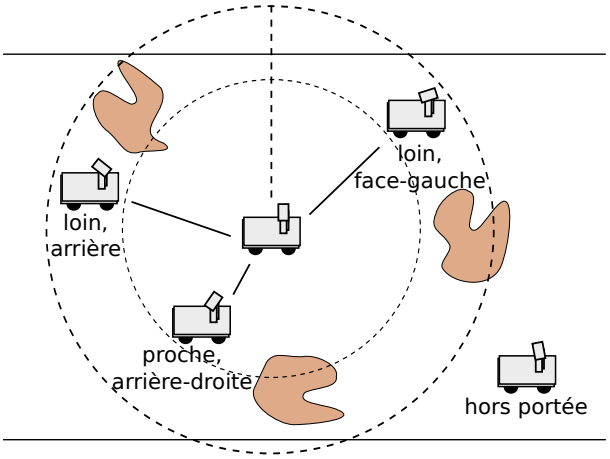


FIGURE 11.10 – Rappel des interactions dans le problème de Thales

Ainsi, un voisin peut être perçu `proche` ou `loin` de l'agent, et ce dans 8 directions possibles (face, gauche, droite, derrière ainsi que les 4 angles). On précise également qu'un voisin en interaction `proche` va « cacher » un agent en interaction `loin`, et que les voisins trop éloignés ne seront pas perçus. On en déduit donc l'ensemble $SR$, qui contient un état relatif pour chacune de ces 17 interactions possibles (les 16 que l'on vient de décrire, plus l'interaction particulière `collision`) et associe à chaque interaction les couples (état_de_l'_agent,état_du_voisin) correspondants. On se référera au chapitre 7, pages 107 et suivantes, pour une description plus détaillée des interactions mises en jeu.

L'ensemble $\Omega R$ et la fonction $OR$ associée sont ici relativement simples. Pour toute interaction possible, on ajoute une observation à $\Omega R$. La fonction $OR$ nous permet alors de recevoir, de façon certaine, les observations associées aux interactions présentes avec les voisins (en ne prenant en compte que les $k$ interactions les plus importantes, $k$ étant la taille de voisinage considéré pour la résolution). L'observabilité partielle sur le voisinage est ici prise en compte dans le fait qu'une même interaction peut correspondre à plusieurs états réels possibles pour le voisin, cette incertitude augmentant au fur et à mesure que le voisin s'éloigne de l'agent.

Il reste finalement à définir $C$, l'ensemble des clusters d'interactions. Ici, on se contentera de deux clusters, l'un associé à la situation de l'agent lorsqu'il ne perçoit pas de voisins ($C^0$) et l'autre pour les états dans lesquels un voisinage est présent ($C^1$). Le premier cluster est simple à définir : il ne contient qu'un seul état « interaction vide », de récompense nulle. Détaillons donc la formalisation du second cluster $C^1 = (S^1,T^1,R^1)$ :

- $S^1$ : le cluster $C^1$ contient l'ensemble des états joints relatifs possibles (hormis celui pour lequel aucun voisin n'est présent), impliquant 1 à $k$ voisins. On peut donc définir l'ensemble $S^1 = \{(sr_1,\ldots,sr_i)|i \in \{1,2,\ldots,k\}, sr_1,\ldots,sr_i \in SR\}$, tel que cet ensemble ne contienne aucun état joint relatif $(sr_1,\ldots,sr_i)$ pour lequel on aurait $sr \in (sr_1,\ldots,sr_i)$ et $sr' \in (sr_1,\ldots,sr_i)$ où $sr$ et $sr'$ décriraient deux interactions concernant la même direction (face, gauche, etc.) mais à deux distances différentes (car pour une direction donnée, un agent proche va « cacher » un agent éloigné).
- $T^1$ : les transitions ne dépendent pas de l'état joint relatif, hormis pour l'interaction particulière `collision` pour laquelle seules les actions `gauche` et `droite` deviennent exécutable.
- $R^1$ : en cas de collision, l'agent reçoit une pénalité de -100. À l'inverse, l'agent reçoit une récompense positive lorsqu'il est dans une bonne interaction. Sinon, la récompense est nulle.

Ainsi, il reste simplement à décider de ce qu'est une bonne interaction. Lorsque le canal est étroit (largeur inférieure à 6), on décidera qu'une bonne interaction consiste à être devant ou derrière un autre agent. La récompense est alors de +10 si l'agent n'a des voisins que devant ou derrière lui, +5 sinon. Au contraire, lorsque le canal est large ($l \geq 6$), toute interaction telle qu'aucun agent n'est présent à distance « proche » sera jugée positive, et apportera une récompense de +10.

### 11.3.3 Faisabilité

Nous avons pu constater, dans la section précédente, qu'il était possible de décrire ce problème complexe via notre approche. La question se pose désormais de la faisabilité, en termes de complexité des structures ainsi générées. On rappelle pour commencer la complexité en temps de nos algorithmes, selon les deux phases de la résolution (pré-traitement des données, pour générer l'ensemble des MDPs associés aux clusters d'interactions, puis calcul d'une politique à partir de ces données, via l'approche par POMDP+MDPs) avec $k$ la taille du voisinage considéré :

1. phase de pré-traitement : $O(|SR|^{2k} \cdot k^{k+4} \cdot j^2 \cdot |A|^2)$,
2. phase de calcul de la politique : $O\left(\frac{1}{(1-\gamma)^2} \cdot \ln \frac{R^{max}}{\epsilon(1-\gamma)^2} \cdot |SR|^{2k} \cdot |A|\right)$.

Il faut également considérer l'étape de résolution du POMDP correspondant au problème individuel de l'agent, mais cette étape ne devrait pas poser de problème au vu de la dimension des structures manipulées ($|S| = 900$, $|A| = 5$ et $|\Omega| = 72$).

Du point de vue « problème d'interactions », la complexité ne dépend que du nombre d'interactions possibles et de la taille du voisinage considéré. Ici, ces dimensions sont similaires aux problèmes étudiés dans la première partie de ce chapitre (open-ISR notamment), dont nous avons montré qu'ils pouvaient être résolus en un temps raisonnable. Ce problème de Thales peut donc être effectivement traité via notre approche. On pourra manipuler, par exemple, 10 agents (plus un leader non contrôlé), mais nous nous contenterons d'un voisinage de taille 3.

L'intérêt principal est ici de voir comment notre approche permet de représenter de tels problèmes. Toutefois, le lecteur désireux de connaître les résultats obtenus sur ce problème trouvera en annexe C une étude plus détaillée des simulations réalisées.

## Conclusion

Ce chapitre nous a permis d'étudier, via un ensemble de benchmarks classiques du domaine, la qualité de notre modèle et des algorithmes associés. Nous avons notamment montré en quoi la taille du voisinage considéré était un paramètre critique pour le temps de résolution du problème (croissance exponentielle). Ce paramètre n'ayant pas besoin de prendre des valeurs trop élevées (dans nos exemple, une valeur de 3 était suffisante), nos algorithmes semblent pouvoir passer à l'échelle et traiter des problèmes de dimension « réelle ». Nous avons de plus montré qu'en jouant sur le nombre d'instances considérées pour chaque état joint relatif, il était possible de diminuer considérablement les temps de calcul, sans trop affaiblir la qualité des politiques produites.

Notre approche offre donc de très bon résultats, alors même que l'on relâche certaines hypothèses nécessaires à l'application des autres approches basées interactions (les approches DPCL ou IDMG notamment). Ainsi, nous avons montré que, tout en considérant des interactions imprévisibles (et non-pas limitées à un ensemble de tâches) et en supposant une observabilité partielle sur le voisinage, on parvient à obtenir des politiques d'une qualité similaire à celles obtenues via les approches existantes.

Nous avons ensuite démontré l'applicabilité de notre approche au problème concret fourni par Thales. Nous avons notamment vu comment représenter formellement ce problème (instanciation de DyLIM), ce qui nous a permis de déduire la dimension des structures ainsi générées. Nous en avons conclu que ce problème pouvait effectivement être traité via notre approche, ce qui a ensuite été vérifié par simulation.

# Conclusion de la partie III

À l'issue de cette troisième et dernière partie, nous disposons désormais d'un ensemble d'algorithmes permettant, à partir du modèle DyLIM, de construire une politique pour chaque agent impliqué dans le problème. Notre approche de résolution, complètement décentralisée, passe par deux étapes principales : générer, à partir du modèle, des données exploitables, puis calculer une politique pour l'agent en fonction de ces données.

Nous avons présenté plusieurs façons de traiter le problème, selon l'objectif recherché. Ainsi, nous avons montré deux algorithmes principaux : le premier repose sur une prise en compte « réactive » des interactions. Cet algorithme permet un calcul relativement rapide de la politique, et génère des comportements de bonne qualité. Le second algorithme quant à lui, propose une prise en compte « prédictive » des interactions. Ainsi, cet algorithme permet d'optimiser la qualité des comportements produits, mais cela se fait au détriment des temps de calcul, qui deviennent plus importants.

Nous avons également présenté d'autres paramètres sur lesquels il est possible de jouer. Entre autres, nous avons vu comment on pouvait relâcher certaines contraintes sur les interactions (en ne considérant, par exemple, que certaines des interactions observées), afin d'alléger les temps de calcul. Ces temps ont d'ailleurs été analysés dans une étude de complexité, mettant en relief les paramètres les plus coûteux. Nous avons ainsi montré que les algorithmes proposés nécessitaient un temps exponentiel en la taille du voisinage considéré.

Finalement, nous avons validé ces algorithmes sur un ensemble de problèmes expérimentaux. Nous avons tout d'abord analysé les temps de calcul et la qualité des politiques produites sur un ensemble de benchmarks tirés de la littérature. Nous avons ensuite montré comment, en utilisant notre méthode, il était possible de traiter le problème fourni par Thales. Nous terminerons ce document par une conclusion reprenant les principaux points exposés dans cette thèse, afin d'en avoir une vision plus globale, puis par une courte discussion autour des perspectives de recherche soulevées par ce travail.

# Conclusion et perspectives

Ce document nous a permis, en trois parties, d'effectuer une étude complète autour de la planification multiagent orientée interactions, afin de proposer une approche pour la résolution de tels problèmes. La communauté DEC-POMDP est actuellement très active autour de cette thématique, le raisonnement à base d'interactions étant particulièrement prometteur pour le passage à l'échelle des algorithmes de résolution. Malheureusement, les approches existantes souffrent de limitations trop fortes, notamment en termes d'applicabilité. Le modèle DyLIM que nous proposons ici, utilisé conjointement à nos algorithmes de résolution, offre donc une première façon de traiter ce type de problèmes sans en restreindre l'applicabilité. Nous proposons désormais de conclure cette étude via une vision plus globale de nos travaux, ainsi qu'une discussion autour de nos prochaines perspectives de recherches.

# 1 Conclusion

**Résumé du document -** nous avons proposé, dans ce document, une étude de la planification multiagent sous incertitude (usuellement traitée via le modèle DEC-POMDP), et des différentes approches existantes pour le calcul de politiques. Nous avons notamment montré que l'on pouvait classer ces approches parmi deux grandes catégories : les approchés basées interactions, et les approches « classiques ». Nous avons ensuite vu que les approches classiques partageaient l'hypothèse implicite d'une interaction permanente entre les agents, d'où la dimension exponentielle en le nombre d'agents des problèmes à résoudre. À l'inverse, les approches basées interactions visent à casser cette complexité. On peut, pour commencer, interdire certaines interactions (en supposant par exemple une indépendance permanente sur les transitions), mais l'applicabilité de ces approches est alors fortement limitée. D'autres approches, plus récentes, proposent non-pas d'interdire certaines interactions, mais de supposer que celles-ci n'existent que localement. Le problème, là encore, est que ces approches reposent sur des hypothèses fortes (notamment, observabilité totale sur le voisinage) afin de permettre la manipulation d'interactions locales, et souffrent ainsi d'une applicabilité limitée.

Il n'existait donc, à priori, aucune approche permettant le passage à l'échelle (notamment en ce qui concerne le nombre d'agents) tout en conservant l'applicabilité d'un DEC-POMDP « classique ». Il y avait pourtant un besoin fort à ce niveau : de nombreux problèmes réels, issus de la robotique notamment, impliquent un grand nombre d'agents et pourraient bénéficier d'une telle approche. Dans notre cas, l'entreprise Thales nous a fourni une application concrète à traiter, de gestion de convoi pour un groupe de véhicules autonomes. Ce problème implique de nombreux agents, mais les interactions entre ceux-ci sont locales. Pourtant, les approches basées interactions existantes sont inapplicables, ce problème ne vérifiant pas les hypothèses nécessaires (par exemple, l'observabilité sur le voisinage est ici partielle).

Nous avons donc proposé un nouveau modèle, DyLIM (pour Dynamic Local Interaction Model), capable de décrire un problème de type DEC-POMDP (sans ajouter d'hypothèse limitative) tout en intégrant une représentation explicite des interactions entre agents. Nous nous sommes pour cela basé sur le comportement d'un humain évoluant au sein d'une foule. Ainsi,

on représente d'une part le problème individuel de l'agent, et d'autre part les interactions possibles avec le voisinage et l'impact de ces interactions sur l'agent. On peut alors représenter le problème individuel de façon très précise (tout comme l'humain se fera une représentation précise du chemin à emprunter pour atteindre son but), tout en autorisant une description plus approximative du voisinage (là encore comme l'humain, qui se contente de prendre en compte les gens devant lui pour ne pas se cogner, mais qui ignore les autres gens présents dans la foule). On « casse » alors la complexité combinatoire des problèmes traités en ne prenant en compte que les éléments nécessaires à la prise de décision.

Nous avons ensuite proposé plusieurs algorithmes capables de calculer une politique à partir d'un problème décrit avec DyLIM. Ainsi, notre approche résout le problème individuel de l'agent, mais prend en compte durant cette résolution l'impact du voisinage et l'évolution probable de celui-ci. Nous avons alors réalisé une étude de complexité à propos de ces algorithmes, et montré que les temps de résolution ne dépendaient pas du nombre d'agents impliqués, mais uniquement de la taille du voisinage considéré. Nous avons finalement validé cette approche via une série d'expérimentations : on constate alors que l'on obtient des résultats similaires aux approches existantes sur les benchmarks tirés de la littérature, mais aussi que l'on peut traiter des problèmes qu'aucune autre approche n'est à même de résoudre (le problème de Thales notamment), notre modèle étant aussi expressif qu'un DEC-POMDP, tout en permettant le passage à l'échelle grâce à l'usage des interactions durant la résolution du problème.

**Discussion autour de la contribution -** notre approche permet le passage à l'échelle du processus de résolution, principalement pour deux raisons : la taille du problème qui ne dépend pas du nombre d'agents impliqués, et la méthode de résolution complètement décentralisée. Ces deux points représentent un avantage considérable lors de la résolution, relativement aux approches classiques, et sont rendus possibles par l'usage des états relatifs. Le premier point (taille ne dépendant pas du nombre d'agents) a été largement discuté dans ce document : on manipule un ensemble d'états joints relatifs, dont le nombre dépend uniquement de la taille de voisinage considérée, et non du nombre d'agents impliqués. Le second point (résolution décentralisée) est plus implicite : le but, lors d'une résolution centralisée, est d'avoir le contrôle sur les actions jointes, afin d'optimiser le comportement du groupe. Ici, les actions exécutées par le voisinage n'ont aucun impact sur l'agent, seul compte l'état de ce voisinage.

Ainsi, les MDPs associés aux clusters d'interactions forment un modèle simplifié du voisinage, suffisant pour la prise de décision. La manipulation de ce modèle est rendue possible par l'aspect « local » des interactions. On utilise en effet un opérateur de choix sur les actions des voisins (on suppose par exemple que ceux-ci vont toujours choisir l'action optimale au vu de nos connaissances, ou encore qu'ils adopteront un comportement en moyenne). Cette heuristique introduit une erreur sur les comportements calculés, mais nous permet pourtant de calculer des politiques satisfaisantes. Pourquoi, dans ce cas, ne pas utiliser une telle heuristique pour calculer le comportement de notre agent ? En fait, il faut voir ici que l'on a séparé le problème individuel du problème d'interactions. Ainsi, si le problème individuel doit être résolu à long terme, via

une méthode d'optimisation (résolution du POMDP) et non une approche heuristique, il n'en va pas de même pour le problème d'interactions.

Les interactions sont, au sein des problèmes traités, locales. Cette localité est non-seulement physique (tel agent, avec tel voisin), mais également temporelle : un agent va soudainement entrer en interaction avec un voisin puis, après quelques pas de temps, quitter cette interaction. Ainsi, si l'agent fait une erreur en prévoyant le comportement de son voisin, l'impact à long terme de cette erreur sera négligeable, puisque l'interaction aura pris fin à court terme. Bien sur, une telle approche ne fournit pas un résultat optimal. Toutefois, l'approximation est suffisamment précise pour permettre un comportement de bonne qualité (comme nous avons pu le montrer dans les expérimentations réalisées).

Le fait de raisonner directement sur les interactions et leur évolution probable est donc un avantage, aussi bien pour la modélisation (limitation du nombre d'états joints) que pour l'aspect algorithmique. Nous avons ainsi constaté que de nombreux modèles existants pouvaient être vus comme des instances de DyLIM. Le modèle IDMG par exemple, peut être vu comme une restriction de DyLIM dans laquelle les interactions seront exactes et complètement observables.

**Point de vue final -** le modèle DyLIM, utilisé conjointement à nos algorithmes, répond aux objectifs que nous nous étions initialement fixés. Ainsi, nous sommes effectivement capables de traiter des problèmes complexes, tels que l'exemple de Thales, et de calculer des politiques de bonne qualité pour ces problèmes en un temps raisonnable, même lorsque de nombreux agents sont impliqués. Pour autant, il reste plusieurs points pouvant être retravaillés, comme nous le verrons dans la section suivante, traitant de nos perspectives de recherche.

Le travail proposé dans cette thèse peut être vu comme un tout. Nous avons en effet construit notre modèle à partir de notre analyse de l'état de l'art, afin de tirer parti des techniques les plus prometteuses (raisonnement basé interactions) tout en palliant aux problèmes des approches existantes (hypothèses contraignantes). Nous avons ensuite construit des algorithmes spécifiques à ce modèle, toujours dans l'idée de pallier aux problèmes des techniques existantes (principalement l'explosion combinatoire empêchant le passage à l'échelle). Ce travail réalisé en parallèle, sur le modèle et les algorithmes, nous a permis d'améliorer les deux aspects : ainsi, nous avons non-seulement cherché à produire de bons algorithmes au vu de notre modèle, mais aussi à inclure dans le modèle tous les éléments nécessaires à une résolution efficace.

Malgré tout, l'approche proposée reste très ouverte quant à son aspect algorithmique. Ainsi, on pourra imaginer développer une toute autre façon de calculer la politique de l'agent, via une résolution centralisée par exemple qui permettrait d'optimiser le comportement joint. On pourra alors tirer parti des spécificités de DyLIM (notamment la façon dont les interactions sont explicitement décrites), afin de faciliter ce processus de résolution. Les algorithmes proposés ici représentent donc une façon parmi d'autres de calculer une politique basée sur notre modèle, mais ne doivent pas être considérés comme la seule approche possible.

## 2 Perspectives

Il reste, à l'issue de cette thèse, plusieurs voies sur lesquelles nous souhaiterions continuer à travailler. Il y a non-seulement des points théoriques que l'on pourrait étudier, mais également certains aspects applicatifs.

**Perspectives théoriques -** la rédaction de ce document nous a permis d'identifier plusieurs points sur lesquels notre approche pourrait être améliorée, notamment en ce qui concerne la gestion des interactions. Nous nous sommes en particulier posé la question de la taille de voisinage à considérer. Dans un problème de navigation par exemple, nous avons constaté qu'un voisinage de taille 3 était suffisant, mais imaginons d'autres interactions pour lesquelles un voisinage de taille 4 par exemple serait nécessaire. Imaginons maintenant que les agent doivent à la fois gérer un problème de navigation, et des interactions nécessitant 4 voisins. Selon notre approche, on posera $k = 4$, ce qui sera inutile durant les phases de navigation. Pourquoi, dans ce cas, ne pas permettre des voisinages dont la taille ne serait pas fixée en fonction du problème, mais en fonction des interactions ? Ainsi, selon la situation, l'agent prendra 3 ou 4 voisins en considération, afin de toujours minimiser la complexité de la résolution. Cela ne devrait, à priori, pas remettre en question notre modèle ni nos algorithmes (moyennant quelques ajustements). Nous envisageons donc de tester l'apport de cette idée à des problèmes complexes autres que la navigation en convoi.

La question se pose également du type de voisins considérés. Pour l'instant, on raisonne sur leurs actions possibles en considérant que les agents sont homogènes, ou au moins qu'ils disposent des mêmes possibilités d'action. Que se passerait-t'il, dans le cas d'un voisinage hétérogène ? Il faudrait notamment étudier l'impact qu'a cette hétérogénéité sur la composante d'interactions : à priori, la « qualité » que l'on associe à une interaction donnée ne dépend pas du type de voisin impliqué. Par contre, cette hétérogénéité influera sur l'évolution de ces interactions. Nous envisageons donc de proposer un algorithme modifié, capable de prendre en compte ce type de situation. De la même manière, on peut étudier l'impact d'un agent piloté par l'humain, évoluant au sein du convoi. Comment considérer un tel agent ? Peut-on se contenter d'estimer que les voisins sont hétérogènes, et que certains d'entre eux n'ont pas toujours un comportement rationnel ? Certains travaux récents, dans notre équipe, étudient la façon dont un agent peut estimer le comportement d'un humain, et agir en conséquence [Karami *et al.*, 2010]. Il faudrait donc, là encore, étudier les modifications à apporter à nos algorithmes afin de gérer ces aspects.

Pour finir, un point intéressant à étudier serait l'apport de la communication au modèle. Nous avons supposé qu'une communication gratuite et illimitée représentait une hypothèse trop optimiste. Par contre, il est courant en robotique que les agents puissent communiquer, dans la limite d'une bande passante donnée, et moyennant un coût de communication connu. Dans ce cas, peut-on envisager d'intégrer la communication au modèle, comme une décision de l'agent ? La description explicite des interactions pourrait-elle être un avantage pour la communication également, en permettant à l'agent de savoir avec qui communiquer par exemple ? Certains travaux ont déjà abordé cette question [Messias *et al.*, 2011], dans le cas du raisonnement basé

indépendances. Il faudrait donc commencer par étudier ces travaux, ce qui fait de ce point un objectif à plus long terme.

**Perspectives applicatives -** nous avons, à court terme, plusieurs perspectives applicatives. Nous envisageons tout d'abord de modéliser le problème de Dassault, comme nous avons modélisé celui de Thales. On rappelle que ce problème consiste à contrôler un ensemble d'avions évoluant sur un aéroport (au sol), afin que chacun ait pu décoller avant une heure donnée, le tout en respectant les règles de navigation en aéroport. Ainsi, l'agent pilotant l'avion doit choisir quelles actions exécuter, sauf dans certains cas particuliers (situation d'urgence) où il faut respecter un protocole donné, c'est-à-dire exécuter une série d'actions imposées. Certains travaux récents de notre équipe [Côté *et al.*, 2011] étudient ce point particulier.

Ainsi, une première façon de représenter ce problème serait de considérer chaque segment de route sur lequel un avion peut évoluer comme un état, les actions consistant alors à passer d'un segment à l'autre. Les interactions entre ces avions décriraient donc leur proximité, aussi bien géographiquement (deux avions ne pouvant pas rouler trop prêt l'un de l'autre) que temporellement (deux avions ne peuvent pas décoller en même temps). Il existe une période idéale entre deux décollages : on pourrait donc associer une grande récompense aux interactions correspondant à cette période, puis une récompense de plus en plus petite à mesure que l'on s'en éloigne. On associerait de même une pénalité aux interactions telles que les agents décollent trop prêt l'un de l'autre, ou telles que la totalité du groupe ne sera pas partie à l'heure prévue.

Nous envisageons également, toujours à court terme, d'implémenter le problème de Thales sur un « véritable » simulateur, gérant la physique des véhicules manipulés, ceci afin de valider notre approche. Une telle implémentation n'est pas triviale, en raison notamment de la difficulté qu'il y a à discrétiser le monde dans lequel les agents évoluent. Nous envisagerons ensuite une implémentation robotique, ce qui nous permettra de tester notre modèle en « situation réelle ».

# Annexes

# Annexe A

# Algorithmes de calcul des transitions ($\mathtt{p}^{\mathtt{out}}$, $\mathtt{p}^{\mathtt{stay}}$ et $\mathtt{p}^{\mathtt{in}}$)

Cette annexe contient les algorithmes utilisés pour le calcul de $p^{out}$, $p^{stay}$ et $p^{in}$, tels qu'ils sont implémentés dans notre solveur. Le calcul de ces trois éléments est notamment mentionné dans la partie III, chapitre 8, section 8.2.2. Il convient toutefois de faire un certain nombre de rappels, avant de présenter les algorithmes en question :

1. *$sr_i$ décrit la relation entre l'agent et son voisin $i$* : on pourra donc noter $instance_i$ l'état de l'agent $i$, tel que $instance_0$ décrive l'état de l'agent et que les $instance_1$, ..., $instance_n$ décrivent les états des $n$ voisins. Ainsi, $\forall i, 1 \leq i \leq n, relation(instance_0, instance_i) = sr_i$.

2. *On dispose d'un ordre de préférence sur les relations* : on pourra en extraire la relation de préférence maximale $sr_{max}$, au sein de $sr$. On peut alors garantir qu'aucun voisin de l'agent n'est présent dans un état qui entraînerait une relation ayant une préférence supérieure à celle de $sr_{max}$ (sinon, cette relation ferait partie de l'état joint relatif). Cela nous sera utile dans l'algorithme *nobody*.

3. *on manipule des relations approchées* : la version s'appliquant aux relations exactes est en effet bien plus simple, puisqu'il suffit de déterminer l'état exact de chaque voisin, puis de calculer les probabilités de transition individuelles et finalement de renvoyer le produit de ces probabilités individuelles.

On peut maintenant introduire l'algorithme 9, décrivant le calcul de $p^{stay}$. L'idée ici est de travailler avec un **ensemble représentatif** d'instances de $sr$, c'est à dire un ensemble contenant suffisamment d'instances pour décrire le panel d'états joints pouvant entraîner l'état joint relatif $sr$. Le chapitre traitant des résultats expérimentaux indique le nombre d'instances considérées dans chaque benchmark, et l'influence de ce nombre sur la qualité des résultats. On calcule ensuite, pour chaque instance, la probabilité de transiter vers $sr'$, après quoi on effectue une moyenne de ces probabilités. On notera que l'algorithme de $p^{stay}$ ne s'applique que si on vérifie $|sr| = |sr'|$. On introduira finalement l'algorithme 10, pour le calcul de $p^{in}$.

**Algorithme 9** : $p^{stay}$ la probabilité que les agents de $sr$ transitent vers $sr'$.

**Entrées** : $sr$ et $sr'$ les états joints relatifs (départ, arrivée), $a$ l'action individuelle

**Sorties** : $p^{stay}(sr'|sr,a)$

// choix (au hasard) de $k$ instances de $sr$ :

$start \leftarrow randomChoice(instances(sr),k)$;

// pour chaque instance de $sr$, on génère les instances de $sr'$ accessibles :

**pour chaque** $instance \in start$ **faire**

  $end(instance) \leftarrow \emptyset$;

  **pour chaque** $instance' \in instances(sr')$ **faire**

    **si** $T^{POMDP}(instance_0,a,instance'_0) > 0$ **alors**

    $end(instance) \leftarrow end(instance) \cup instance'$;

// on calcule maintenant la probabilité de transition, de $sr$ à $sr'$ :

$proba \leftarrow 0$;

// comme on ne connaît pas l'instance actuelle de $rs$, on fait la moyenne sur l'ensemble des instances possibles :

**pour chaque** $instance \in start$ **faire**

  $p\_start \leftarrow 0$ et $total \leftarrow 0$;

  // on calcule une moyenne sur les $instance'$, pondérée par les $T^{POMDP}(instance_0,a,instance'_0)$. En effet, plus la probabilité que l'agent transite vers un état $instance'_0$ est forte, plus la transition associée aura de poids dans la moyenne :

  **pour chaque** $instance' \in end(instance)$ **faire**

    $p\_end \leftarrow T^{POMDP}(instance_0,a,instance'_0)$;

    $total \leftarrow total + p\_end$;

    // on ne connaît pas les actions exécutées par les autres agents, on se contente donc là-encore d'une moyenne sur les $a' \in A^{POMDP}$ :

    **pour chaque** *agent* $i$, $1 \leq i \leq |sr'|$ **faire**

      // l'agent en $instance'_i$ peut venir de $instance_1$ OU de $instance_2$ OU... d'où la somme sur les $j$ :

      $p\_sP \leftarrow \sum_{j=1}^{|sr|} \frac{\sum_{a' \in A^{POMDP}} T^{POMDP}(instance_j,a',instance'_i)}{|A^{POMDP}|}$;

      // il faut un agent sur $instance'_1$ ET un agent sur $instance'_2$ ET... d'où le produit sur les $i$ :

      $p\_end \leftarrow \frac{p\_end * p\_sP}{|sr|}$;

    $p\_start \leftarrow p\_start + p\_end$;

  // on applique $moyenne = \frac{\sum_x P(x).valeur(x)}{\sum_x P(x)}$ :

  $proba \leftarrow proba + \frac{p\_start}{total}$;

// on retourne finalement la transition en moyenne :

**retourner** $\frac{proba}{|start|}$;

**Algorithme 10** : $p^{in}$ la probabilité, sachant $sr$ l'état joint relatif initial, que des agents venus de l'extérieur génèrent l'état joint relatif $sr'$.

**Entrées** : $sr$ et $sr'$ des états joints relatifs, $a$ l'action individuelle de l'agent

**Sorties** : $p^{in}(sr'|sr,a)$

// cas particulier - lorsqu'on s'intéresse à 0 agents, ceux-ci transitent vers un $sr'$ ≪ vide ≫ dans 100% des cas :

**si** $|sr'| = 0$ **alors retourner** *1*;

// on calcule l'ensemble des positions où l'agent peut initialement se trouver, sachant l'état joint relatif de ses voisins :

$origines \leftarrow \{s \in S | \exists instance \in randomChoice(instances(sr),k)\ tq.\ s = instance_0\}$;

// on va calculer, pour chaque état initial possible de l'agent, l'atteignabilité de $sr'$ par ses voisins. Il suffira alors de renvoyer l'atteignabilité moyenne :

**pour chaque** $s \in origines$ **faire**

  // on calcule tout d'abord l'ensemble *nobody* des états dans lesquels on est certain qu'aucun voisin ne se trouve (sachant l'état initial $s$, dans lequel a lieu l'interaction jointe $sr$) :

  $nobody \leftarrow nobody(s,sr)$;

  // on calcule ensuite l'atteignabilité de $sr'$, sachant *nobody* :

  $p \leftarrow 0$;

  $total \leftarrow 0$;

  // on génère pour cela les instances de $sr'$ accessibles depuis $s$ :

  $cibles \leftarrow \emptyset$;

  **pour chaque** $instance' \in instances(sr')$ **faire**

    **si** $T^{POMDP}(s,a,instance'_0) > 0$ **alors** $cibles \leftarrow cibles \cup instance'$;

  // puis on fait une moyenne pondérée sur ces instances :

  **pour chaque** $instance' \in cibles$ **faire**

    $poids \leftarrow T^{POMDP}(s,a,instance'_0)$;

    $total \leftarrow total + poids$;

    // il faut un voisin en $instance'_1$ ET un voisin en $instance'_2$ ET... d'où le produit des probabilités (via la méthode *incoming*) :

    $poids\_voisins \leftarrow \prod_{j=1}^{|instance'|} incoming(instance'_j|nobody)$;

    $p \leftarrow p + (poids \times poids\_voisins)$;

  $proba(s) \leftarrow p/total$;

// on renvoie finalement l'atteignabilité moyenne :

**retourner** $\frac{\sum_{s \in origines} proba(s)}{|origines|}$;

L'algorithme 10 fait appel à deux autres blocs de calculs : `nobody` et `incoming`. On utilise `nobody` pour identifier l'ensemble des états dans lesquels on est certain qu'aucun agent ne se trouve. Cela permet de limiter l'incertitude, lors des calculs de transition. Dans le calcul de $p^{in}$ par exemple, cela nous permettra de limiter le nombre d'états dans lesquels les agents non-observés peuvent se situer. Inversement, dans le calcul de $p^{out}$, cela permet de limiter le nombre d'états vers lesquels les agents « sortant » peuvent transiter. L'algorithme 11 décrit ces calculs.

**Algorithme 11** : *nobody* l'ensemble des états dans lesquels on peut garantir qu'aucun voisin de l'agent n'est présent.

**Entrées** : $s$ l'état initial de l'agent et $sr$ l'état joint relatif de ses voisins
**Sorties** : ensemble d'états où l'on est sur qu'aucun agent ne se trouve
// on extrait la relation de préférence maximale, au sein de $sr$ :
$preference\_max \leftarrow \max_{sr_i \in sr} preference(sr_i)$;
// on calcule l'ensemble des instances associées à l'état $s$ :
$instances \leftarrow \{s \in S | \exists inst \in instances(sr)\ tq.\ s = inst_0\}$;
// puis l'ensemble des états pouvant entraîner une interaction avec $s$ :
$zone\_interaction \leftarrow \{s' \in S | \exists sr_i \in SR\ tq.\ relation(s,s') = sr_i\}$;
$nobody \leftarrow \emptyset$;
// soit $k$ le nombre max de relations considérées au même moment, le cas où l'interaction n'est pas saturée :
**si** $|sr| < k$ **alors**
    // si un des états pouvant entraîner une interaction n'est présent dans aucune instance de $sr$, c'est qu'il ne contient aucun agent :
    **pour chaque** $s' \in zone\_interaction$ **faire**
        // ici, toute instance est de la forme $instance(sr) = (s,inst_1,inst_2,\dots)$
        **si** $\nexists inst \in instances\ tq.\ s' \in \{inst_1,\dots,inst_{|inst|}\}$ **alors** $nobody \leftarrow nobody \cup s'$;
// on traite ensuite le cas où l'interaction est saturée (donc où certains états relatifs peuvent être « ignorés »:
**sinon**
    **pour chaque** $s' \in zone\_interaction$ **faire**
        // l'interaction étant saturée, on peut ignorer des relations. Il faut donc également vérifier si la relation à une priorité forte (telle que cette relation ne serait pas ignorée) :
        $sr_i \leftarrow relation(s,s')$;
        $pref \leftarrow preference(sr_i)$;
        **si** $pref > preference\_max$ *et* $\nexists inst \in instances\ tq.\ s' \in \{inst_1,\dots,inst_{|inst|}\}$
        **alors** $nobody \leftarrow nobody \cup s'$;
**retourner** *nobody*;

L'algorithme `incoming` quant à lui, décrit le « cœur » du calcul de $p^{in}$, puisqu'il permet de calculer la probabilité qu'un agent hors interaction transite vers un état $s'$ donné. On suppose une distribution uniforme des agents sur les états non-observés, et on effectue une moyenne sur l'ensemble des actions que ces agents peuvent effectuer. L'algorithme 12 décrit cette approche, pour le calcul de `incoming`.

**Algorithme 12** : *incoming* probabilité qu'un agent venu de l'extérieur transite vers $s'$, sachant qu'il ne peut pas être dans *nobody*.

**Entrées** : $s'$ l'état cible considéré et *nobody* les états « vides »

**Sorties** : la probabilité $P(s'|nobody)$

`// on calcule l'ensemble des états d'où peut provenir l'agent :`

$candidats \leftarrow \{s \in S | s \notin nobody \text{ et } \exists a \in A,\ T^{POMDP}(s,a,s') > 0\}$;

`// on calcule la probabilité de transition en moyenne, pour toute action :`

$$p \leftarrow \sum_{s \in candidats} \frac{\sum_{a \in A} T^{POMDP}(s,a,s')}{|A|};$$

`// on ne connaît pas l'état actuel du voisin concerné, donc on retourne la moyenne sur les états de départ possibles :`

**retourner** $\frac{p}{|S|-|nobody|}$;

On a donc décrit le calcul de $p^{stay}$ et de $p^{in}$. On introduit maintenant le calcul de $p^{out}$, c'est-à-dire la probabilité qu'ont certains voisins de quitter l'interaction. On manipule des relations approchées : on cherche donc la probabilité qu'ont ces voisins de transiter soit vers un état sans interaction (sachant $sr'$), soit vers une des interactions déjà présentes dans $sr'$ (l'interaction étant déjà présente, le voisins en question serait « absorbé » par celle-ci). Le principe de cet algorithme est le suivant :

- $sr$ désigne ici l'état joint relatifs des agents devant quitter l'interaction (on « oublie » les autres agents),
- on sélectionne les instances représentatives de $sr$,
- pour chaque instance, on calcule la probabilité qu'ont les voisins de quitter l'interaction (transition hors interaction, où dans une interaction déjà présente dans $sr'$),
- en renvoie la moyenne de ces probabilités.

Ce processus, permettant le calcul de $p^{out}$, est décrit dans l'algorithme 13.

**Algorithme 13** : $p^{out}$ la probabilité que les agents de $sr$ quittent l'interaction.

**Entrées** : $sr$ et $sr'$ des états joints relatifs, $a$ l'action individuelle de l'agent

**Sorties** : $p^{out}(sr'|sr,a)$

**si** $|sr'| = 0$ **alors retourner** *1*;

// sélection des instances de $sr$ et $sr'$ :

$origines \leftarrow randomChoice(instances(sr),k)$;

**pour chaque** $inst \in origines$ **faire**

  $cibles(inst) \leftarrow \emptyset$;

  **pour chaque** $inst' \in instances(sr')$ **faire**

    **si** $T^{POMDP}(inst_0,a,inst'_0) > 0$ **alors** $cibles(inst) \leftarrow cibles(inst) \cup inst'$;

// probabilité, pour chaque instance de $sr$, de quitter l'interaction :

**pour chaque** $instance \in origines$ **faire**

  // on extrait l'ensemble des positions ou l'agent peut terminer :

  $E \leftarrow \{s' \in S | \exists inst' \in cibles(instance)\ tq.\ s' = inst'_0\ et\ T^{POMDP}(instance_0,a,s') > 0\}$;

  $total \leftarrow 0$;

  // puis on calcule, pour chaque $s'$, la probabilité que les agents de $sr$ ne génèrent aucune interaction (afin de faire la moyenne pondérée) :

  **pour chaque** $s' \in E$ **faire**

    // on calcule tout d'abord l'ensemble *nobody* des états vers lesquels on est certain qu'aucun voisin ne va transiter (sachant l'état final $s'$, dans lequel aura lieu l'interaction jointe $sr'$) :

    $nobody \leftarrow nobody(s',sr')$;

    // on calcule ensuite la probabilité de transiter soit vers $sr'$, soit hors interaction, depuis $sr$ et sachant *nobody* :

    $poids \leftarrow T^{POMDP}(instance_0, a, s')$;

    $total \leftarrow total + poids$;

    // on détermine pour cela l'ensemble des états vers lesquels les agents peuvent transiter :

    **pour chaque** $1 \leq j \leq |instance|$ **faire**

      $candidats(instance_j) \leftarrow \{s'' \in S | s'' \notin nobody, T^{POMDP}(instance_j,a,s'') > 0\}$;

    // chaque agent de $sr$ doit quitter l'interaction (d'où le produit), mais on ignore les actions exécutées (d'où la moyenne), et n'importe quel état cible est acceptable (d'où la somme) :

    $poids\_voisins \leftarrow \prod_{j=1}^{|instance|} \frac{\sum_{a \in A}\left[\sum_{s'' \in candidats(instance_j)} T^{POMDP}(instance_j,a,s'')\right]}{|A|}$;

    $proba(s') \leftarrow poids \times poids\_voisins$;

  $transition(instance) \leftarrow \sum_{s' \in E} proba(s')/total$;

// on renvoie finalement la probabilité moyenne :

**retourner** $\sum_{instance \in origines} transition(instance)/|origines|$;

Si on cherche à calculer des récompenses, et non plus des transitions, il faudra définir $r^{stay}$, $r^{out}$ et $r^{in}$ à la place de $p^{stay}$, $p^{out}$ et $p^{in}$. Les fonctions `nobody` et `incoming` quant à elles resteront inchangées. On définit ces fonctions de la façon suivante :

- $r^{out}$ : on calcule la récompense obtenue par l'agent et ses voisins, durant la transition. Ainsi, les *nouveaux* voisins ne doivent pas être pris en compte dans ce calcul. On peut donc poser, pour toute transition, $r^{out} = 0$.
- $r^{stay}$ : ici, le calcul sera identique à celui de $p^{stay}$, à une modification prêt. À la fin de l'algorithme de $p^{stay}$, on calcule $p\_sP \leftarrow \sum_{j=1}^{|sr|} \frac{\sum_{a' \in A^{POMDP}} T^{POMDP}(instance_j, a', instance'_i)}{|A^{POMDP}|}$. Il faudra simplement remplacer dans cette ligne l'appel à $T^{POMDP}$ par un appel à $R^{POMDP}$.
- $r^{in}$ : là encore, on pourra reprendre l'algorithme de $p^{in}$, en intégrant lors du calcul des poids des voisins (fin de l'algorithme) un appel à $R^{POMDP}$.

# Annexe B

# Protocole expérimental

Cette annexe décrit le protocole expérimental observé durant l'exécution de l'ensemble de nos tests. Nous avons implémenté un solveur, utilisant le modèle DyLIM et les algorithmes décrits dans ce document, ainsi qu'un simulateur pour tester les politiques ainsi calculées. Trois éléments sont ici décrits : (1) les hypothèses admises durant l'exécution de ces benchmarks, (2) la méthode employée pour estimer le degré de réussite d'un test en particulier et (3) le déroulement « standard » de ces tests.

### 1 - Hypothèses admises durant l'exécution des benchmarks

Nous avons, dans la partie II de ce document (modélisation du problème avec DyLIM), mis en avant un certain nombre d'hypothèses devant être respectée afin de ne pas limiter l'applicabilité du modèle. Afin de ne pas « tricher » durant la réalisation de nos tests, nous avons fait en sorte que chaque benchmark utilisé nécessite la vérification de chacune de ces hypothèses. Ainsi, les problèmes testés impliquent les éléments suivants :

- *L'exécution des politiques est décentralisée* : chaque agent est donc responsable de sa prise de décision. Il n'y a aucun élément central, au sein du simulateur, qui puisse prendre des décisions au niveau du groupe. Au contraire, chaque agent doit calculer sa propre politique et appliquer celle-ci en se basant sur ses observations.
- *Il n'y a pas de communication possible* : là encore, les agents n'ont aucun moyen d'échanger de l'information. Ils peuvent uniquement observer les autres agents présents dans leur voisinage.
- *L'observabilité est partielle* : les agents n'ont qu'une vision partielle de leur environnement, mais aussi de l'état des autres agents (même en ce qui concerne les voisins en interaction avec l'agent).
- *Les interactions manipulées sont complexes* : ainsi, l'agent peut être en interaction avec n'importe lequel des autres agents, et ce en n'importe quel état. De plus, ces interactions peuvent influer aussi bien sur les récompenses que sur les transitions ou observations.

Ainsi, au vu de ces quatre hypothèses, les problèmes testés sont tous des problèmes complexes, ne pouvant pas être résolus via les approches existantes.

**2 - Estimation du degré de réussite d'un test**

Il est difficile d'estimer la valeur des politiques calculées, sur ce genre de problème. En particulier, les méthodes existantes pour le calcul à priori de la valeur d'une politique jointe sont très coûteuses, notamment lorsque le nombre d'agents est élevé. Ainsi, ce type de méthode n'étant pas applicable aux problèmes de grande taille, il est courant de simuler l'exécution des politiques afin de calculer la somme des récompenses obtenues. Bien sur, si on souhaite obtenir une valeur représentative de la qualité de la politique, il faut simuler l'exécution de celle-ci un grand nombre de fois, et calculer la moyenne des récompenses ainsi obtenues. On parle alors de **récompense moyenne décomptée** (ou ADR, pour *Average Discounted Reward*).

Le calcule de l'ADR est relativement simple : on commence par simuler une première fois l'exécution de la politique. À chaque instant $t$, chaque agent $i$ exécute une action choisie selon sa politique et génère une récompense $r_{i,t}$. On peut donc calculer la récompense jointe $r_t = \sum_i r_{i,t}$. On arrête l'exécution lorsque $t = h$, avec $h$ l'**horizon de simulation** choisi. On peut alors calculer la récompense décomptée $dr = \sum_{t=1}^{h} \gamma^{t-1} r_t$ de cette simulation. On exécute ainsi un grand nombre de fois la simulation, en calculant à chaque fois la valeur $dr$ associée. L'ADR est alors une simple moyenne de ces récompenses décomptées. Si on calcule cette moyenne sur un nombre suffisamment grand de simulations, l'ADR ainsi calculé converge vers la valeur réelle de la politique jointe. Durant nos expérimentations, sauf information contraire, nous avions fixé une valeur de $\gamma = 0{,}95$ pour le facteur d'atténuation et de $h = 30$ pour l'horizon de planification. Nous avons de plus simulé l'exécution de chaque problème exactement 1000 fois : ce nombre de simulations est suffisant pour voir l'ADR converger vers une valeur stable.

**3 - Déroulement « standard » d'un test donné**

On commence systématiquement par résoudre le problème (c'est-à-dire calculer la politique individuelle de chaque agent). On exécute ensuite la simulation autant de fois que nécessaire, sans re-calculer les politiques entre chaque simulation. Chaque simulation suit le scénario suivant :

- chaque agent commence par recevoir une observation au sujet de son environnement, et de son voisinage,
- les agents mettent à jour leurs états de croyance (individuels et relationnels),
- chaque agent choisi son action selon sa politique individuelle et ses états de croyance,
- les actions sont exécutées simultanément, le simulateur transite vers un nouvel état tiré selon la distribution de probabilités donnée par la fonction de transition,
- on comptabilise les récompenses associées à cette transition,
- on génère les observations de chaque agent, au vu de la transition réalisée,
- on reprend à la première étape, tant que l'on n'a pas atteint $t = h$.

Ainsi, on respecte bien les hypothèses de base (exécution décentralisée, observation partielle). L'exécution de cette simulation peut être très rapide, le seul élément coûteux étant la mise à jour des états de croyance. Cette étape pouvant également être exécutée rapidement si l'implémentation est bonne, on peut exécuter les 1000 simulations en un temps raisonnable.

# Annexe C

# Simulation du problème de Thales

## Simulation expérimentale

Afin de prouver l'applicabilité de notre approche, nous avons choisi d'implémenter et simuler ce problème, puis d'analyser la façon dont le convoi adapte sa structure à la situation. La figure C.1 présente une vision symbolique de notre simulateur. L'agent leader du convoi est représenté en vert, entouré des autres agents. La partie blanche représente le canal de navigation des agents, la partie beige représente la bordure entre ce canal et la zone n'appartenant pas à la mission. Cet environnement évolue au fur et à mesure de l'exécution de la mission.

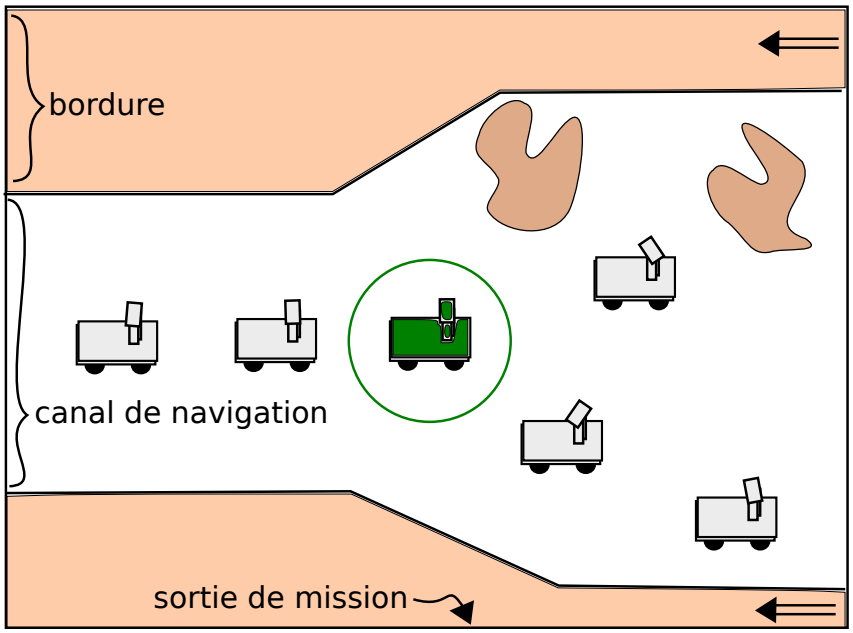


FIGURE C.1 – exemple d'environnement de simulation du problème de Thales.

Le concept de cette simulation est le suivant : l'environnement défile à l'écran, de la droite vers la gauche. Ainsi, le leader qui roule au centre du canal de navigation apparaît immobile à l'écran. En observant l'évolution des récompenses cumulées durant l'exécution du problème, on

peut analyser la capacité du groupe à s'adapter à l'environnement changeant, tout en maintenant le convoi. On se base ici sur la formalisation donnée précédemment (chapitre 11, section 11.3).

Cette simulation peut être paramétrée sous différents axes : on peut jouer sur les capacités des agents (à quel point peuvent-ils accélérer et ralentir ? quelle est leur capacité à se déporter sur les cotés ? Quelle est la portée de leur capteurs ?), mais aussi sur la difficulté de l'environnement (notamment la vitesse à laquelle la largeur du canal de navigation évolue).

## Résultats obtenus

(les expérimentations sont en cours de réalisation, et seront ajoutées à une version ultérieure du manuscrit)

# Bibliographie


[Aberdeen, 2003] Aberdeen, D. (2003). A (revised) survey of approximate methods for solving partially observable markov decision processes. *National ICT Australia, Canberra, Australia, Tech. Rep.*

[Allen, 2009] Allen, M. (2009). *Interactions in decentralized environments.* Thèse de doctorat, University of Massachusetts.

[Amato *et al.*, 2009] Amato, C., Dibangoye, J. et Zilberstein, S. (2009). Incremental policy generation for finite-horizon dec-pomdps. *In Proceedings of the nineteenth International Conference on Automated Planning and Scheduling (ICAPS-09)*, pages 2–9.

[Astrom, 1965] Astrom, K. (1965). Optimal control of markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10(1):174–205.

[Becker *et al.*, 2004a] Becker, R., Zilberstein, S. et Lesser, V. (2004a). Decentralized markov decision processes with event-driven interactions. *In Proceedings of the third international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-04)*, pages 302–309. IEEE Computer Society.

[Becker *et al.*, 2003] Becker, R., Zilberstein, S., Lesser, V. et Goldman, C. (2003). Transition-independent decentralized markov decision processes. *In Proceedings of the second international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-03)*, pages 41–48. ACM.

[Becker *et al.*, 2004b] Becker, R., Zilberstein, S., Lesser, V. et Goldman, C. (2004b). Solving transition independent decentralized markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, pages 423–455.

[Bellman, 1957] Bellman, R. (1957). Dynamic Programming. *Princeton University Press.*

[Bernstein *et al.*, 2005] Bernstein, D., Hansen, E. et Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. *WS4*, page 52.

[Bernstein *et al.*, 2000] Bernstein, D., Zilberstein, S. et Immerman, N. (2000). The complexity of decentralized control of markov decision processes. *In Proceedings of the sixteenth conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 32–37. Morgan Kaufmann Publishers Inc.

[Bertsekas, 1995] Bertsekas, D. (1995). Dynamic programming and optimal control, vol. 1, 2. *Athena Scientific.*

[Beynier, 2006] Beynier, A. (2006). Une contribution à la résolution des Processus Décisionnels de Markov Décentralisés avec contraintes temporelles. *These de doctorat, Université de Caen-Basse Normandie.*

[Boussard *et al.*, 2008] Boussard, M., Bouzid, M. et Mouaddib, A. (2008). Vector valued markov decision process for robot platooning. *In Proceeding of the eighteenth European Conference on Artificial Intelligence (ECAI-08)*, pages 925–926. IOS Press.

[Boutilier, 1996] Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. *In Proceedings of the sixth conference on Theoretical Aspects of Rationality and Knowledge (TARK-96)*, pages 195–210. Morgan Kaufmann Publishers Inc.

[Boutilier *et al.*, 1999] Boutilier, C., Dean, T. et Hanks, S. (1999). Decision-theoretic planning : Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research (JAIR)*, 11:94.

[Boutilier *et al.*, 1995] Boutilier, C., Dearden, R. et Goldszmidt, M. (1995). Exploiting structure in policy construction. *In Proceedings of the fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1104–1113.

[Canu et Mouaddib, 2011a] Canu, A. et Mouaddib, A.-I. (2011a). Collective decision-theoretic planning for planet exploration. *In International Conference on Tools with Artificial Intelligence (ICTAI).*

[Canu et Mouaddib, 2011b] Canu, A. et Mouaddib, A.-I. (2011b). Collective decision under partial observability : a dynamic local interaction model. *In Proceedings of the International Joint Conference on Computational Intelligence (IJCCI-11).*

[Capitán *et al.*, 2011] Capitán, J., Spaan, M., Merino, L. et Ollero, A. (2011). Decentralized multi-robot cooperation with auctioned pomdps. *In Sixth Annual Workshop on Multiagent Sequential Decision Making in Uncertain Domains (MSDM-2011)*, page 24.

[Cassandra, 1998] Cassandra, A. (1998). *Exact and approximate algorithms for partially observable Markov decision processes.* Thèse de doctorat, Brown University.

[Cassandra *et al.*, 1994] Cassandra, A., Kaelbling, L. et Littman, M. (1994). Acting optimally in partially observable stochastic domains. *In Proceedings of the twelfth National Conference on Artificial Intelligence (AAAI-94).*

[Chades *et al.*, 2002] Chades, I., Scherrer, B. et Charpillet, F. (2002). A heuristic approach for solving decentralized-pomdp : Assessment on the pursuit problem. *In Proceedings of the 2002 ACM symposium on Applied computing*, pages 57–62. ACM.

[Côté *et al.*, 2011] Côté, N., Bouzid, M. et Mouaddib, A.-I. (2011). Integrating the human recommendations in the decision process of autonomous agents : A goal biased markov decision process. *In Fall Symposium AAAI.*

[Dibangoye *et al.*, 2009] Dibangoye, J., Mouaddib, A.-I. et Chaib-draa, B. (2009). Point Based Incremental Pruning Heuristic for Solving Finite-Horizon DEC-POMDPs. *In Pro-*

*ceedings of the eighth international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-09)*.

[Ghavamzadeh *et al.*, 2006] Ghavamzadeh, M., Mahadevan, S. et Makar, R. (2006). Hierarchical multi-agent reinforcement learning. *Proceedings of the fifth international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-06)*, 13(2):197–229.

[Gmytrasiewicz et Doshi, 2005] Gmytrasiewicz, P. et Doshi, P. (2005). A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research (JAIR)*, 24(1):49–79.

[Goldman et Zilberstein, 2003] Goldman, C. et Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. *In Proceedings of the second international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-03)*, pages 137–144. ACM.

[Goldman et Zilberstein, 2004] Goldman, C. et Zilberstein, S. (2004). Decentralized control of cooperative systems : Categorization and complexity analysis. *Journal of Artificial Intelligence Research (JAIR)*, 22:143–174.

[Goldman et Zilberstein, 2005] Goldman, C. et Zilberstein, S. (2005). Goal-oriented dec-mdps with direct communication. *Computer Science Technical Report TR-04-44, University of Massachusetts at Amherst*.

[Goodrich *et al.*, 2001] Goodrich, M., Olsen, D., Crandall, J. et Palmer, T. (2001). Experiments in adjustable autonomy. pages 1624–1629.

[Guestrin *et al.*, 2002] Guestrin, C., Koller, D. et Parr, R. (2002). Multiagent planning with factored mdps. *Advances in neural information processing systems*, 2:1523–1530.

[Hansen *et al.*, 2004] Hansen, E., Bernstein, D. et Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. *In Proceedings of the National Conference on Artificial Intelligence (AAAI-04)*, pages 709–715. Menlo Park, CA ; Cambridge, MA ; London ; AAAI Press ; MIT Press ; 1999.

[Howard, 1960] Howard, R. (1960). Dynamic programming and Markov processes.

[Hsu *et al.*, 2007] Hsu, D., Lee, W. et Rong, N. (2007). What makes some pomdp problems easy to approximate. *Advances in Neural Information Processing Systems (NIPS)*, page 28.

[Karami *et al.*, 2010] Karami, A.-B., Jeanpierre, L. et Mouaddib, A.-I. (2010). Human-robot collaboration for a shared mission. *In Proceedings of the 5th ACM/IEEE International Conference on Human Robot Interaction, HRI 2010, Osaka, Japan, March 2-5*, pages 155–156.

[Kok *et al.*, 2005] Kok, J., Hoen, P., Bakker, B. et Vlassis, N. (2005). Utile coordination : Learning interdependencies among cooperative agents. *In Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 29–36. Citeseer.

[Kumar et Zilberstein, 2009] Kumar, A. et Zilberstein, S. (2009). Constraint-based dynamic programming for decentralized pomdps with structured interactions. *In Proceedings*

*of the eighth international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-09)*, pages 561–568, Budapest, Hungary.

[Kurniawati *et al.*, 2008] Kurniawati, H., Hsu, D. et Lee, W. (2008). SARSOP : Efficient point-based POMDP planning by approximating optimally reachable belief spaces. *In Proceedings of the Robotics : Science and Systems conference (RSS-08)*.

[Littman, 1996] Littman, M. (1996). *Algorithms for Sequential Decision Making*. Thèse de doctorat, Brown University.

[Madani *et al.*, 1999] Madani, O., Hanks, S. et Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. *In Proceedings of the National Conference on Artificial Intelligence (AAAI-99)*, pages 541–548. JOHN WILEY & SONS LTD.

[Marecki *et al.*, 2008] Marecki, J., Gupta, T., Varakantham, P., Tambe, M. et Yokoo, M. (2008). Not all agents are equal : Scaling up distributed POMDPs for agent networks. *In Proceedings of the seventh international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-08)*, pages 485–492. International Foundation for Autonomous Agents and Multiagent Systems.

[Melo et Veloso, 2011] Melo, F. et Veloso, M. (2011). Decentralized mdps with sparse interactions. *Artificial Intelligence*.

[Messias *et al.*, 2011] Messias, J. V., Spaan, M. T. J. et Lima, P. U. (2011). Exploiting sparse dependencies for communication reduction in multiagent planning under uncertainty. *In Decision Making in Partially Observable, Uncertain Worlds : Exploring Insights from Multiple Communities (Workshop at IJCAI)*.

[Mostafa et Lesser, 2011] Mostafa, H. et Lesser, V. (2011). A compact mathematical formulation for problems with structured agent interactions. *In Sixth Annual Workshop on Multiagent Sequential Decision Making in Uncertain Domains (MSDM-2011)*, page 55.

[Mouaddib *et al.*, 2007] Mouaddib, A., Boussard, M. et Bouzid, M. (2007). Towards a formal framework for multi-objective multiagent planning. *In Proceedings of the sixth international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-07)*, pages 1–8. ACM.

[Nair et Tambe, 2005] Nair, R. et Tambe, M. (2005). Hybrid bdi-pomdp framework for multiagent teaming. *Journal of Artificial Intelligence Research (JAIR)*, 23(1):367–420.

[Nair *et al.*, 2003] Nair, R., Tambe, M., Yokoo, M., Pynadath, D. et Marsella, S. (2003). Taming decentralized POMDPs : Towards efficient policy computation for multiagent settings. *In Proceedings of the eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, volume 18, pages 705–711. Citeseer.

[Nair *et al.*, 2005] Nair, R., Varakantham, P., Tambe, M. et Yokoo, M. (2005). Networked distributed pomdps : A synthesis of distributed constraint optimization and pomdps. *In Proceedings of the twentieth National Conference on Artificial Intelligence (AAAI-05)*.

[Oliehoek *et al.*, 2010] Oliehoek, F., Spaan, M., Dibangoye, J. et Amato, C. (2010). Heuristic search for identical payoff bayesian games. *In Proceedings of the nineth international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-10)*, pages 1115–1122. International Foundation for Autonomous Agents and Multiagent Systems.

[Oliehoek *et al.*, 2008] Oliehoek, F., Spaan, M., Whiteson, S. et Vlassis, N. (2008). Exploiting locality of interaction in factored Dec-POMDPs. *In Proceedings of the seventh international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-08)*.

[Papadimitriou et Tsitsiklis, 1987] Papadimitriou, C. et Tsitsiklis, J. (1987). The complexity of markov decision processes. *Mathematics of operations research*, pages 441–450.

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds, and schools : A distributed behavioral model. *Computer Graphics*, 21(4):25–34.

[Russell et Norvig, 2009] Russell, S. et Norvig, P. (2009). *Artificial intelligence : a modern approach*. Prentice hall.

[Seuken et Zilberstein, 2007] Seuken, S. et Zilberstein, S. (2007). Memory-bounded dynamic programming for DEC-POMDPs. *In Proceedings of the twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2009–2016.

[Spaan et Melo, 2008] Spaan, M. et Melo, F. (2008). Interaction-driven Markov games for decentralized multiagent planning under uncertainty. *In Proceedings of the seventh international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-08)*.

[Szer et Charpillet, 2005] Szer, D. et Charpillet, F. (2005). An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. *In Proceedings of the sixteenth European Conference on Machine Learning (ECML-05)*, pages 389–399.

[Szer *et al.*, 2005] Szer, D., Charpillet, F. et 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 576–590. Citeseer.

[Varakantham *et al.*, 2009] Varakantham, P., Kwak, J., Taylor, M., Marecki, J., Scerri, P. et Tambe, M. (2009). Exploiting coordination locales in distributed POMDPs via social model shaping. *In Proceedings of the nineteenth International Conference on Automated Planning and Scheduling (ICAPS-09)*.

[Varakantham *et al.*, 2007] Varakantham, P., Marecki, J., Yabu, Y., Tambe, M. et Yokoo, M. (2007). Letting loose a spider on a network of pomdps : Generating quality guaranteed policies. *In Proceedings of the sixth international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS-07)*, pages 1–8. ACM.

[Witwicki et Durfee, 2010] Witwicki, S. et Durfee, E. (2010). Influence-based policy abstraction for weakly-coupled dec-pomdps. *In Proceedings of the twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

[Wooldridge et Jennings, 1995] Wooldridge, M. et Jennings, N. (1995). Intelligent agents : Theory and practice. *The knowledge engineering review*, 10(02):115–152.

# Planification multiagent sous incertitude orientée interactions : modèle et algorithmes

## Résumé

Cette thèse adresse le problème de la planification multiagent, lorsque l'environnement est partiellement observable et que le résultat des actions est soumis à une incertitude. Un état de l'art est proposé autour des techniques existantes (le modèle DEC-POMDP et ses dérivés) et montre que ces approches, souffrant d'une explosion combinatoire, sont insuffisantes pour traiter des problèmes de taille « réelle ». On présente alors un nouveau modèle, permettant de contourner le problème de l'explosion combinatoire. Ce modèle décrit d'une part un problème individuel (lorsque l'on ne prend en compte que l'existence d'un seul agent), et d'autre part l'influence des voisins sur cet agent. Ainsi, on reprend l'approche classique des SMA, visant à faire émerger un comportement de groupe à partir des interactions locales, mais on pallie aux faiblesses de ces approches en adoptant un raisonnement rationnel sur l'impact des interactions observées et à venir, grâce à une approche de type MDP.

On propose finalement un ensemble d'algorithmes pour le calcul d'une politique de comportement basée sur ce modèle. On décrit ainsi plusieurs approches, permettant de considérer les interactions immédiates uniquement, ou également celles à venir, de prendre en compte plus ou moins de voisins dans le raisonnement de l'agent, etc. La complexité de ces algorithmes est exponentielle en le nombre de voisins considérés simultanément : on peut donc calculer des politiques de bonne qualité pour des problèmes de taille réelle, dès l'instant où on se limite à un voisinage de taille raisonnable.

## Abstract

This thesis deals with partially observable multiagent decision-making problems. First of all, a state of the art describes the existing approaches (DEC-POMDP and its sub-models) : because of the exponential complexity, they can not deal with real-world problems. Then, we introduce a new model, to avoid this combinatorial complexity. Our model is made of two parts : the first one describes an individual problem (how the agent evolves, while ignoring the other agents) and the second one describes an interaction problem (how the neighbors influence the agent). Such an approach comes from MultiAgent Systems, where the group behavior is emerging from local interactions, but we avoid the weakness of these approaches by adopting a rational reasoning over the current and future interactions, with an MDP.

Finally, we give several algorithms to compute a policy based on our model. We show how to deal with current interactions only or to predict the futur interactions too, how to consider a given neighborhood size while computing the policy, etc. The time complexity to compute a policy with these algorithms is exponential in the neighborhood size, so we are able to compute good policies for real-size problems, by choosing a small enough neighborhood.

**Mots-clés indexation Rameau :** Markov, processus de ; Intelligence artificielle répartie ; Robots mobiles ; Planification

**Discipline :** Informatique et applications

Groupement de Recherche en Informatique, Image, Automatique et Instrumentation de Caen
CNRS UMR 6072, Université de Caen Basse-Normandie BP 5186, 14032 Caen Cedex, FRANCE