Etude de l'auto-organisation dans les algorithmes de patrouille multi-agent fondés sur les phéromones digitales

Arnaud Glad

NANCY
UNIVERSITE

UFR mathématiques et informatique **École doctorale IAEM Lorraine**

Département de formation doctorale en informatique

Etude de l'auto-organisation dans les algorithmes de patrouille multi-agent fondés sur les phéromones digitales

THÈSE

pour l'obtention du

Doctorat de l'université Nancy 2

(spécialité informatique)

par

Arnaud Glad

Composition du jury

Rapporteurs :	Salima Hassas, Professeur, LIESP - Université Lyon 1 Philippe Mathieu, Professeur, LIFL - Université Lille 1
Examinateurs :	Didier Bazalgette, Responsable du domaine scientifique Homme et Systèmes - DGA Olivier Buffet, Chargé de recherche, LORIA - INRIA Jacques Ferber, Professeur, LIRMM - Université Montpellier 2 Abderrafiaa Koukam, Professeur, SET - Université de Technologie de Belfort-Montbéliard Kamel Smaili, Professeur, LORIA - Université Nancy 2
Directeurs de Thèse :	François Charpillet, Directeur de recherches, LORIA - INRIA, Directeur Olivier Simonin, Maître de conférences HDR, LORIA - Université Henri Poincaré Nancy 1, Co-directeur

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Mis en page avec la classe thloria.

Table des matières

Introduction **vii**

1 La complexité dans les systèmes artificiels viii
2 L'auto-organisation dans les systèmes multi-agent réactifs ix
3 Le problème de la patrouille multi-agent ix
4 Plan de la thèse et contributions x

Partie I État de l'art **1**

Chapitre 1
Les systèmes multi-agents réactifs

1.1 Définition générale 3
1.1.1 L'agent 3
1.1.2 L'agent et l'environnement 4
1.1.3 Des agents réactifs et des agents cognitifs 5
1.2 De l'agent au multi-agent 7
1.2.1 La complexité dans les systèmes multi-agents 7
1.2.2 Etude par simulation 8
1.3 L'intelligence en essaim 9
1.3.1 Les champs de potentiels 10
1.3.2 Les algorithmes fourmi et les phéromones digitales 11
1.3.3 Modélisation informatique des mécanismes d'évaporation et de diffusion 12
1.4 Conclusion du chapitre 14

Chapitre 2
La Patrouille multi-agent

2.1 Le problème de la patrouille multi-agent — définition 17
2.1.1 La patrouille vue comme la visite répétitive de l'environnement . . 18
2.1.2 Perception d'intrus à distance 19
2.1.3 Problème considéré 19
2.2 Représentation des environnements discrets 20
2.2.1 Représentation topologique de l'environnement 20
2.2.2 Représentation métrique de l'environnement 22
2.3 Critères de performance pour la patrouille en environnement discret . . . 23
2.3.1 Critères basés sur l'oisiveté 23
2.3.2 Critères basés sur la fréquence de visite 24
2.3.3 Calcul de l'oisiveté optimale 24
2.4 Approches existantes 26
2.4.1 Techniques relevant de la recherche opérationnelle 26
2.4.2 Apprentissage par renforcement 28
2.4.3 Agents cognitifs 29
2.4.4 Algorithmes fourmi 29
2.5 Conclusion du chapitre 32

Partie II EVAP - un algorithme fourmi pour la patrouille

Chapitre 3
Le modèle EVAP

3.1 EVAP - un algorithme fourmi fondé sur le dépôt d'informations 38
3.1.1 Principe du modèle EVAP 38
3.1.2 Définition du modèle 39
3.1.3 Représentation de l'environnement 42
3.1.4 Un premier aperçu du comportement d'EVAP 43
3.2 EVAW - un EVAP sans phéromones 47

3.2.1 Les modèles VAW et EVAW 48
3.2.2 Équivalence entre les marquages 49
3.2.3 Marquage statique contre marquage dynamique 50
3.3 Quelques propriétés du modèle EVAP 51
3.3.1 Borne sur le temps de couverture 51
3.3.2 Propriété de patrouille 53

Chapitre 4
Evaluation des performances d'EVAP pour la patrouille multi-agent

4.1 CLInG, un algorithme fondé sur la propagation d'information 57
4.1.1 Présentation 57
4.1.2 Comportement et hypothèses sur l'environnement 58
4.2 Hypothèses de simulation et détails techniques 59
4.2.1 Hypothèses de simulation 59
4.2.2 Mesure de performances 60
4.3 Etude de la couverture 61
4.3.1 Observations expérimentales 61
4.3.2 Etude du comportement exploratoire des agents 63
4.3.3 Les algorithmes fourmi et la couverture 66
4.4 Etude de la patrouille (phase d'organisation) 67
4.4.1 Comparaison des patrouilles d'EVAP, de LRTA* et de CLInG . . 68
4.4.2 Impact du nombre d'agents sur la patrouille 71
4.4.3 Conclusion sur la patrouille par les algorithmes fourmis 72
4.5 Etude de la robustesse d'EVAP 73
4.5.1 Robustesse à la perte d'agents 73
4.5.2 Passage à l'échelle 75
4.6 Conclusion sur la patrouille d'EVAP 78

Chapitre 5
Outils et notations pour l'étude expérimentale des comportements cycliques

5.1 Description formelle de l'évolution des algorithmes fourmis 80
5.1.1 Représentation de l'état du système 80
5.1.2 L'évolution du système comme une chaîne de Markov 81
5.2 Définition des comportements cycliques 82
5.2.1 Sous-graphes transients 83

5.2.2 Sous-graphes récurrents 84
5.2.3 Cycles d'états du système 85
5.2.4 Preuve de convergence des algorithmes fourmi pour la patrouille vers des structures répétitives 85
5.3 Détection des sous-graphes récurrents dans les algorithmes fourmis 87
5.3.1 Détection de cycles dans les systèmes déterministes avec *the Tortoise and the Hare* 88
5.3.2 Détection de cycles d'états du système dans un système dynamique discret non déterministe 89
5.3.3 Détection des sous-graphes récurrents 90
5.3.4 Notre algorithme de recherche de sous-graphes 92
5.3.5 Problèmes pratiques et performances 93

Chapitre 6
Etude de l'auto-organisation en cycles

6.1 Stabilité et instabilité des cycles d'états du système 98
6.1.1 Les sous-graphes d'un point de vue agents 98
6.1.2 Instabilité des cycles de longueurs différente 99
6.1.3 Stabilité des cycles agent de même longueur 101
6.2 Améliorer la convergence vers des cycles 105
6.2.1 Visualisation 105
6.2.2 Des motifs particuliers 105
6.2.3 EVAW+ : réduire les temps de convergence 107
6.3 Étude expérimentale 109
6.3.1 Cadre expérimental 109
6.3.2 Résultats avec des grilles simples 109
6.3.3 Topologies irrégulières 113
6.3.4 Oisiveté des cycles 114
6.3.5 Robustesse des comportements cycliques 115
6.4 Conclusion sur l'auto-organisation en cycles 115

Chapitre 7
Influence des modèles d'exécution dans l'étude de systèmes auto-organisés

7.1 Modèles d'exécution 119
7.1.1 Gestion de l'indéterminisme 119
7.1.2 Représentation et granularité du temps 120

7.1.3 Hypothèses d'ordonnancement 121
7.2 Implémentation robotique 122
7.3 Effet des modifications du modèle d'exécution 124
7.3.1 Hypothèses de simulation 124
7.3.2 Propriété de patrouille 124
7.3.3 Auto-organisation en cycles 125
7.3.4 Des robots et des cycles 127
7.4 Conclusion 128

Conclusion de la thèse

1 Synthèse des contributions 131
2 Perspectives 134
2.1 Amélioration du comportement exploratoire d'EVAP 134
2.2 Extension d'EVAP à d'autres problèmes 134
2.3 Algorithmes fourmi et programmation dynamique : des phéromones pour apprendre 134
2.4 Résolution de problèmes par des approches auto-organisées 135

Bibliographie **143**

Introduction

1 La complexité dans les systèmes artificiels

Les progrès technologiques réalisés au cours des dernières décennies conduisent les systèmes artificiels créés par l'homme vers une complexité toujours croissante. Cette augmentation de la complexité pose un problème majeur dans la gestion et la conception de ces systèmes. En effet, les approches classiques pour la conception et la gestion de tels systèmes — nous parlons ici des approches centralisées, top-down et des modèles macroscopiques analytiques — montrent rapidement leurs limites tant au niveau du contrôle du système que dans leur capacité prédictive.

Les systèmes dont nous parlons ici peuvent être assimilés à des systèmes complexes, c'est-à-dire à des systèmes décentralisés caractérisés par *un grand nombre d'entités* et dont le comportement *émerge* des interactions entre les entités les constituant. La principale problématique liée aux systèmes complexes provient de leur nature *non linéaire* et *imprévisible* ainsi que des phénomènes émergents [1] qui peuvent apparaître. S'ils peuvent atteindre des régimes réguliers et stables dans le temps, leur évolution est principalement conditionnée par les interactions locales entre les entités. Ainsi, une perturbation minime dans le système peut altérer les boucles de rétroaction [2] et engendrer une réaction brutale et imprévisible, allant d'une réorganisation partielle du système à un changement de régime (notons toutefois que ce changement n'est pas nécessairement négatif).

Comprendre, contrôler, influencer ou simplement maintenir de tels systèmes est devenu un enjeu scientifique majeur. La science s'est traditionnellement efforcée de modéliser ce type de systèmes à travers des approches globales, macroscopiques, prenant le système comme un "tout" et ignorant — au niveau microscopique — l'entité, pourtant au centre du fonctionnement du système. La non linéarité de ces systèmes met en difficulté les approches analytiques classiques (en termes explicatifs et prédictifs) dès lors qu'ils sortent d'un régime stable. L'histoire a montré, à de nombreuses reprises et dans de nombreux domaines, qu'il ne s'agit pas vraiment d'une bonne idée [Mandelbrot and Hudson, 2006] (on pensera notamment aux crashs réguliers du système boursier, un système complexe hautement hétérogène).

La fin des années 80 marque l'émergence du paradigme multi-agent qui considère l'entité comme élément central de l'étude et qui permet de modéliser fidèlement des systèmes complexes pour les étudier. Une des sources d'inspiration de ce paradigme provient de l'observation de systèmes naturels, en particulier des colonies d'insectes sociaux comme les fourmis [Deneubourg *et al.*, 1990], les termites [Grassé, 1959] ou les abeilles [Seeley and Buhrman, 2001]. Ces systèmes complexes vivants sont en effets soumis à des contraintes fortes de la part de l'environnement et ont dû s'adapter en mettant en place des mécanismes robustes de décision décentralisés, de régulation et d'auto-réparation permettant la survie de milliers d'individus.

1. On parle de phénomène émergent ou d'auto-organisation dès lors que ce phénomène, provenant des interactions entre les entités, ne peut être prédit par des méthodes analytiques ou déduit depuis les spécifications des composants du système.

2. Dans une boucle de rétroaction, l'action d'une entité modifie le *monde* qui influencera, par sa modification, les actions futures des entités. Il existe de ce fait deux types de boucles de rétroaction ; les boucles de rétroaction positives qui amplifient un phénomène — lorsque les actions des entités les poussent à continuer ces mêmes actions — et les boucles de rétroaction négatives qui atténuent un phénomène — lorsque les actions entités les poussent à arrêter ces mêmes actions.

2 L'auto-organisation dans les systèmes multi-agent réactifs

Les systèmes distribués artificiels s'inspirant des systèmes biologiques, et en particulier des insectes sociaux, définissent une catégorie de systèmes multi-agents (dits *réactifs* ou *collectifs*). Ils reposent sur des agents aux comportements simples et aux capacités cognitives et de communication très limitées [Ferber, 1995]. La puissance de modélisation et l'expressivité des systèmes multi-agent a permis, d'abord, de reproduire les comportements observés chez certains insectes sociaux, chose jusqu'alors inaccessible aux modèles mathématiques. Les expérimentations réalisées à travers ces modèles informatiques permettant la validation ou la confirmation des hypothèses des biologistes (cette utilisation des systèmes multi-agent est d'ailleurs toujours d'actualité) [Deneubourg *et al.*, 1990; Bourjot *et al.*, 2003].

Dans un second temps, l'exploitation des mécanismes identifiés a permis de résoudre des problèmes particuliers [Drogoul, 1993], proches des tâches effectuées par les insectes. Il s'agit d'une démarche opposée à la précédente, une démarche de conception des systèmes plutôt que de reproduction à visée explicative. L'approche SMA réactive reste cependant limitée à des catégories de problèmes relativement réduites. En effet, si la puissance de ces approches ne fait pas de doutes et que de nombreux processus semblent relever de ce type d'approches (comme par exemple, l'embryogénèse, les phénomènes grégaires — *herding* — chez les humains [Raafat *et al.*, 2009], les systèmes boursiers ou encore la construction de nid chez les insectes [Grassé, 1959]), la complexité de la création et du contrôle de boucles de rétroaction — et donc des phénomènes auto-organisés — dépasse de loin ce que les méthodologie actuelles permettent [3] et il est difficile de s'éloigner des tâches classiques du domaine (comme le flocking [Reynolds, 1987] — déplacement en formation sans leader — ou le foraging [Deneubourg *et al.*, 1990] — recherche et récupération de nourriture —).

Nous nous retrouvons donc devant un ensemble de modèles et principes plus ou moins génériques répondant à un ensemble relativement limité de types de problèmes. Un enjeu majeur du domaine consiste donc à formaliser les principes à l'origine des processus responsables de l'auto-organisation afin de donner au domaine un socle théorique solide permettant de concevoir des systèmes complexes pour la résolution de problèmes plus variés et de plus grande complexité.

3 Le problème de la patrouille multi-agent

Nous proposons dans cette thèse de nous intéresser à la problématique de la compréhension et de la formalisation des phénomènes d'auto-organisation au travers de l'étude d'un modèle multi-agent réactif bio-inspiré pour le problème de la patrouille.

Le problème de la patrouille multi-agent se définit comme la visite répétitive, par un groupe d'agents, de tous les nœuds d'un environnement discret représenté par un graphe. Dans ce problème on cherche à réduire le délai entre deux visites consécutives d'un même nœud [Almeida *et al.*, 2004].

Suivant les travaux initiés par Wagner [Wagner *et al.*, 1999] qui proposa l'utilisation d'al-

3. C'est vrai en particulier lorsqu'il s'agit de la composition de plusieurs boucles de rétroaction qui s'influencent les unes les autres.

gorithmes stigmergiques [4] pour la tâche de couverture d'un graphe (un problème proche de celui de la patrouille), nous proposons dans cette thèse le modèle de patrouille EVAP qui tire son inspiration du comportement des fourmis.

Dans la nature, certaines espèces de fourmi marquent leur passage par le dépôt d'une substance chimique, les phéromones, que peuvent percevoir les autres fourmis pour en tirer de l'information.

Dans le cadre d'EVAP, les agents, qui ne peuvent communiquer directement entre eux et qui ne possèdent ni mémoire ni carte de l'environnement, laissent derrière eux une trace de *phéromones digitales* — le pendant informatique des phéromones animales — servant à matérialiser leur passage. Les phéromones s'évaporant au fil du temps, il se crée un gradient que descendent les agents pour se rendre, localement vers les zones les plus anciennement visitées [5]. Outre le fait de réaliser la tâche de patrouille et de présenter des propriétés intéressantes pour cette tâche (comme la robustesse à l'ajout ou la perte d'agents, l'imprévisibilité du comportement des agents ou encore la capacité de passer à l'échelle), le système présente la capacité surprenante, considérant la simplicité du comportement des agents, de s'auto-organiser en cycles, une structure stable et particulièrement efficace en ce qui concerne la tâche de patrouille.

Nous nous proposons donc d'étudier ce phénomène émergent, à la fois sous des aspects expérimentaux et théoriques, afin de pouvoir, d'une part, mesurer ce phénomène (vitesse de convergence, qualité de la patrouille) et d'autre part, formaliser et comprendre ce phénomène d'auto-organisation.

4 Plan de la thèse et contributions

Chapitre 1 Ce chapitre présente le paradigme multi-agent de modélisation et de simulation. Les systèmes multi-agents se définissent comme un moyen de représenter et d'étudier les systèmes fondés sur les interactions des entités les composant (c'est-à-dire, les systèmes complexes). Nous présentons un rapide état de l'art du domaine dans lequel nous parlerons en particulier des modèles réactifs et du lien qu'entretiennent les systèmes multi-agents avec les systèmes complexes.

Chapitre 2 La patrouille multi-agent est une problématique large qui cache plusieurs problèmes aux objectifs différents sous une unique dénomination.

La patrouille multi-agent, telle que nous la considérons ici, consiste à visiter le plus régulièrement possible des lieux sensibles d'un environnement. Nous nous plaçons dans le cadre de l'exploration d'environnements inconnus qui doivent être patrouillés intégralement. Ce problème a été, au cours des dernières années, approché selon différents paradigmes.

Ce chapitre propose d'abord de présenter les multiples variantes du problème de la patrouille, en particulier au niveau des objectifs et de la représentation du problème. Dans un second temps, nous faisons un tour d'horizon des approches connues et mettons en avant les avantages et inconvénients de ces méthodes.

4. La stigmergie est un concept biologique énoncé par [Grassé, 1959] qui réfère à l'utilisation de l'environnement, par un groupe d'agents, comme moyen de communication et de coordination à travers l'utilisation de marqueurs chimiques, les phéromones.

5. Par évaporation les zones les plus anciennement visitée sont celles qui contiennent la quantité de phéromone la plus basse.

Chapitre 3 Ce chapitre définit le modèle EVAP qui propose une solution réactive et décentralisée au problème de la patrouille multi-agents. Nous présentons donc EVAP, un algorithme de type *fourmi* reposant sur des agents marquant une grille de l'environnement avec une phéromone digitale. Les agents suivent un comportement basé sur une descente de gradient visitant ainsi localement les zones les plus anciennement explorées.

Nous suivons une démarche parallèle à celle de Wagner [Wagner *et al.*, 1999] qui a déjà posé des résultats théoriques sur VAW, un algorithme pour la patrouille proche d'EVAP. Nous étendons donc certaines preuves mathématiques de VAW (preuve de couverture, persistance des cycles) à EVAP.

Nous avons également observé que l'algorithme EVAP exhibe un comportement émergent de formation de cycles, rendant la patrouille optimale (cycles hamiltoniens) ou quasi optimale. Les chapitres 5 et 6 seront entièrement consacrés à la formalisation et à l'étude de ce phénomène.

Le formalisme d'EVAP, notamment du fait du champ de phéromone en constante modification, rend cette étude difficile. Nous proposons, pour faciliter son étude, l'algorithme EVAW au *comportement équivalent* à celui d'EVAP mais utilisant un marquage statique, c'est-à-dire le marquage de la date de visite. Ce formalisme permet de faciliter l'étude théorique du modèle et permet de transposer directement les résultats obtenus à EVAP.

Chapitre 4 Nous étudions dans ce chapitre la capacité d'EVAP à réaliser la tâche de patrouille en le confrontant à deux autres algorithmes fourmis pour patrouille. Nous nous intéressons notamment aux propriétés typiques des modèles multi-agents (robustesse, adaptabilité, passage à l'échelle) pour mettre en avant l'intérêt de telles approches par rapport aux techniques classiques.

Chapitre 5 EVAP n'étant pas un algorithme déterministe (les agents doivent parfois réaliser des choix aléatoires), une partie des "cycles" que nous avons pu observer expérimentalement correspond en fait à des structures répétitives plus complexes, les *sous-graphes récurrents* (qui possédent des propriétés proches des cycles en ce qui concerne la patrouille). Les sous-graphes récurrents sont cependant difficiles à identifier et à détecter d'un point de vue individu centré. Mettre ces structures en évidence nécessite d'adopter un point de vu extérieur au système. Pour cela, nous décrivons l'évolution des algorithmes fourmis sous la forme d'une chaîne de Markov.

Cette formalisation nous permet également de nous intéresser au problème de la détection des sous-graphes récurrents. Plusieurs algorithmes de recherche de comportements cycliques existent mais ne peuvent être utilisés que pour des systèmes déterministes. Nous étendons donc l'algorithme du lièvre et de la tortue [Floyd, 1967] afin de proposer un outil générique de détection de comportements cycliques pour les systèmes multi-agents réactifs discrets dans des cadres non déterministes [Glad *et al.*, 2009].

Chapitre 6 Ce chapitre se concentre sur l'étude, à la fois théorique et expérimentale, de la convergence d'EVAP vers des comportements cycliques. Nous savons qu'EVAP converge vers des sous-graphes récurrents et des cycles mais le formalisme présenté au chapitre 5 ne permet pas de savoir s'ils présentent un intérêt pour la patrouille. Nous prouvons — dans le cadre multi-agent et sous certaines hypothèses — que seuls les cycles de longueurs égales persistent [Glad *et al.*, 2008]. Cette propriété garantie une répartition équilibrées des agents au travers de l'environnement et les performances de la solution obtenue.

Nous avons également observé que le temps de convergence vers les cycles et sous-graphes récurrents devient très important dès lors que la taille du problème augmente (en nombre d'agents et/ou dans les dimensions de l'environnement). L'étude de l'évolution du champ de phéromones nous a permis de mettre en évidence un motif à la base de l'émergence de cycles et de proposer EVAP+, une heuristique permettant une convergence plus rapide vers des cycles et des sous-graphes récurrents. Une étude expérimentale d'EVAP+ nous a permis d'observer une forte amélioration du nombre de convergences vers des cycles optimaux ainsi que de la vitesse de convergence [Glad *et al.*, 2009].

Chapitre 7 Un point souvent négligé dans l'étude des systèmes multi-agents concerne l'influence de la façon dont on les exécute. Ces systèmes sont implicitement fondés sur les interactions entre agents et la façon de les gérer (ordonnancement, résolution des conflits) peut avoir un impact significatif sur la qualité des comportements obtenus. Nous avons donc mené une étude de l'influence de ces paramètres à la fois en simulation et sur une implémentation robotique (sur des robots khepera 3) [Glad *et al.*, 2010]. Un fait notable concernant cette dernière est la perte de la capacité du système à converger vers des cycles stables — et ce malgré la preuve formelle établie précédemment — à cause d'hypothèses d'exécution différentes de celles considérées initialement.

Nous mettons ainsi en évidence l'importance des modèles d'exécution dans les systèmes multi-agents réactifs et montrons la nécessité de bien séparer définition formelle du modèle et implémentation afin de pouvoir garantir les propriétés du système indépendamment des contraintes implémentatoires.

Première partie

État de l'art

Chapitre 1

Les systèmes multi-agents réactifs

La complexité grandissante des systèmes artificiels laisse entrevoir les limites des approches classiques de l'intelligence artificielle dans la conception et la gestion de ces systèmes.

En effet, l'augmentation de leur taille et du nombre d'entités provoque, par les interactions entre les composants du système, une explosion combinatoire rendant impossible l'utilisation en pratique des théorèmes fondamentaux d'optimalité (on pensera notamment au principe d'optimalité de Bellman qui permet seulement de résoudre de manière exacte, avec la puissance de calcul des machines actuelles, des problèmes n'impliquant que quelques agents et à un horizon très court).

A partir de ce constat, il devient nécessaire de se tourner vers des approche alternatives abandonnant l'optimalité et le contrôle total du système pour la robustesse et l'auto-organisation qui permettront à ces systèmes de s'adapter à des événements inattendus et de résister aux erreurs qui pourraient se produire (défaillances, erreurs de perception).

Dès lors, nous présentons dans ce chapitre le paradigme de modélisation et de simulation multi-agent qui permet de concevoir, modéliser et étudier de tels systèmes complexes.

1.1 Définition générale

Les systèmes multi-agents sont un paradigme de modélisation de systèmes complexes et de résolution de problèmes construit autour de la notion centrale d'interaction entre des entités autonomes, les *agents*. Cette section consistera à présenter rapidement les concepts fondateurs du domaine.

1.1.1 L'agent

Le concept d'agent se retrouve dans de nombreux domaines et est généralement défini comme étant une entité autonome capable de réaliser des actions. Plusieurs définitions de l'agent s'affrontent. En effet, ce concept reste encore aujourd'hui assez flou puisqu'il ne définit pas, à proprement parler, de formalisme de conception ou d'implémentation mais plutôt un ensemble de guides décrivant les grandes propriétés que doit exhiber un agent.

Ainsi nous nous trouvons face à de multiples définitions ([Ferber, 1995; Wooldridge *et al.*, 1999; Briot and Demazeau, 2001]) plus ou moins généralistes. Si ces définitions particulières peuvent se contredire sur certains points, il semble se dégager dans la communauté multi-agent un consensus sur ce qui définit un agent.

Sera donc un agent, une entité :

autonome L'agent est capable d'agir de sa propre volonté, c'est-à-dire qu'il ne peut être contrôlé par une entité externe à lui même et qu'il prend des décisions sur la base de son *état interne*. Cet état interne correspond à l'ensemble des perceptions, des connaissances et des croyances de l'agent, contrôle la dynamique de l'agent et ne peut être accédé par d'autres éléments du système.

située [6] Un agent est nécessairement immergé dans un *environnement* sur lequel il pourra *agir*. L'agent est également capable de *percevoir* son environnement afin de pouvoir réagir aux changements de celui-ci (qu'ils aient été induits par l'agent lui même, d'autres agents ou tout autre événement extérieur).

sociale L'agent lorsqu'il rencontre d'autres agents est capable de communiquer et d'interagir avec eux. Ces interactions sont toujours effectuées à travers l'environnement, qu'elles soient directes [7] ou indirectes [8] [Simonin, 2010].

pro-active L'agent produit des décisions — sur la base de son état interne et de ses perceptions ; cf. autonomie — motivées par la poursuite d'un *but*.

Cette définition reste assez abstraite et ne donne aucune indication sur la façon de concevoir des agents pour répondre à un problème spécifique. Ainsi, au contraire des approches classiques dites *top-down* que nous pouvons considérer comme des frameworks dans lesquels on fait "rentrer" le problème (c'est-à-dire formaliser le problème selon des règles données ; comme les modèles markoviens pour les approches à base de processus décisionnels de Markov), les systèmes multi-agents suivent une approche inverse, dite *bottom-up*, dans laquelle on adapte un formalisme très souple aux contraintes du problème (un modèle multi-agent est donc souvent un modèle *ad-hoc*).

1.1.2 L'agent et l'environnement

L'environnement est un *espace* (physique ou non [9]) dans lequel évoluent les agents, qu'ils peuvent percevoir et modifier et qui leur sert également de *médium de communication*. Le concept d'agent n'a d'ailleurs de sens que s'il est pris en conjonction d'un environnement. Les

6. On parle aussi d'agents incarnés ou d'*embodiment* dans certaines définitions. Il s'agit de concepts forts qui, en plus de sa présence dans un environnement, implique que l'agent possède une enveloppe physique lui permettant d'interagir avec cet environnement et les autres agents.

7. Les interactions sont directes lorsqu'un agent s'adresse directement à un autre. Ce type d'interaction présuppose un langage. Notons que ces interactions sont bien réalisées à travers l'environnement qui sert alors de canal de communication.

8. Les interactions sont indirectes lorsque les agents déposent l'information dans l'environnement. Cette information pourra être lue et exploitée par tout autre agent capable de la percevoir.

9. L'environnement peut être vu comme physique et disposer d'une métrique permettant d'établir une notion de proximité entre les agents qui définira les possibilités d'interactions. A l'inverse, il peut également être considéré comme étant sans dimensions, auquel cas il sera limité à la transmission des communications entre agents.

décisions d'un agent sont en effet dictées par l'*état interne* de l'agent, lui même modifié par les perceptions, elles mêmes modifiées par des changements dans l'environnement pouvant être dus aux précédentes actions de l'agent. Nous sommes donc en présence d'une boucle perception-action liant à la fois l'agent et l'environnement (voir figure 1.1).

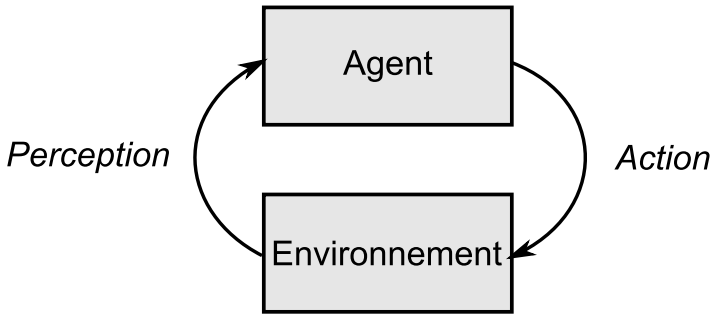


FIGURE 1.1 – Boucle perception-action.

L'environnement possède lui aussi plusieurs propriétés le définissant et conditionnant directement le comportement de l'agent. Ainsi, un environnement peut être :

accessible/inaccessible Cette propriété décrit la possibilité pour un agent de connaître/percevoir l'intégralité de l'environnement. D'une manière générale, l'agent étant une entité immergée dans un environnement, celui-ci n'est pas capable de connaître à un instant donné l'état de l'environnement dans son ensemble et ne possède qu'une *vision locale* de l'état du système, bien souvent insuffisante pour agir de façon *optimale*.

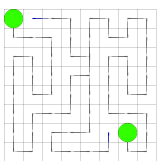
déterministe/non déterministe Dans un environnement déterministe, chaque action effectuée par un agent fait évoluer l'environnement vers un état unique. A l'inverse, dans un environnement non déterministe, chaque action est soumise à une probabilité de réalisation. Cette propriété peut être utilisée pour modéliser les incertitudes que l'on cherche à représenter.

statique/dynamique Cette propriété indique si un environnement peut être modifié autrement que par l'action des agents. Un environnement doté d'une force de pesanteur (les objets tombent s'il ne sont pas posés sur quelque chose), un environnement capable d'évaporer une phéromone ou un environnement modifiant de lui même sa topologie (par exemple un réseau sur lequel certains nœuds peuvent tomber en panne) sont des exemples d'environnement dynamique.

continu/discret Un environnement est continu lorsqu'il est possible d'associer à chaque point de l'environnement une position réelle. Un environnement discret, lorsqu'il est fini, n'admet qu'un nombre fini de positions accessibles aux agents. Typiquement, un environnement discret sera représenté sous la forme d'un graphe.

1.1.3 Des agents réactifs et des agents cognitifs

Nous avons vu précédemment que les agents possèdent un état interne qui définit la dynamique de leur comportement (les actions qu'ils prendront en fonction de la situation rencontrée). Un agent est généralement considéré comme un processus cyclique en trois étapes, *perception* - *décision* - *action* [Ferber, 1995] que nous pouvons représenter par la figure 1.2.



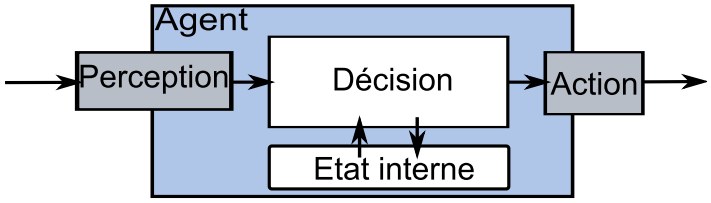


FIGURE 1.2 – Schéma représentant l'architecture d'un agent

Tout agent dispose donc de capteurs lui permettant de percevoir l'environnement, d'effecteurs lui permettant d'agir sur l'environnement et d'un module de décision qui permet, à partir des perceptions, de générer les actions qu'il va effectuer.

Les phases de perception et d'action sont globalement indépendantes de l'architecture physique des agents. La phase de perception peut être ramenée à un opérateur transformant les données perceptives en informations exploitables par l'agent et la phase d'action à un opérateur transformant les décisions de l'agent en action *via* ses effecteurs.

La phase décisionnelle est la part la plus importante de l'architecture interne de l'agent. C'est en effet elle qui détermine le *comportement* de l'agent. La communauté des systèmes multi-agents distingue deux grands types d'architecture interne. Cette distinction est réalisée en fonction du type de traitement réalisé sur les perceptions et de la rationalité du raisonnement effectué par les agents. Nous distinguons ainsi l'architecture cognitive, fondée sur une approche symbolique, qui permet un raisonnement logique de l'architecture réactive, fondée sur le traitement de données numériques, dans laquelle les agents réagissent aux stimuli extérieurs.

L'architecture cognitive — L'architecture cognitive repose sur le traitement d'informations symboliques et suppose donc l'existence d'un *langage* sur lequel pourront raisonner les agents. On fait souvent référence, lorsqu'on évoque cette architecture, à des agents *rationnels* ou des agents *délibératifs*. L'architecture la plus connue lorsque nous parlons de ce type d'agent est l'architecture BDI (pour Beliefs-Desires-Intentions) [Rao and Georgeff, 1995] qui permet aux agents de raisonner sur une base de croyances (Beliefs) pour répondre au but qu'ils se sont fixés (Desires). Ils produisent ensuite des intentions, c'est à dire des actions qu'ils souhaitent effectuer et qui pourront ou non se réaliser.

L'architecture réactive — Dans *The Society of Mind* [Minsky, 1986], Minsky présente un modèle de l'intelligence humaine bâti à partir des interactions entre des entités simples — incapables de produire un raisonnement élaboré — qu'il appelle *agents*. Ce concept d'agents aux capacités cognitives réduites a ensuite été repris et développé, sous l'impulsion de travaux en biologie et en informatique, à travers la modélisation des sociétés d'insectes [Deneubourg *et al.*, 1990; Ferber, 1995; Drogoul, 1993]. Si ces agents réactifs sont individuellement extrêmement limités de par leur comportement "précablé" — typiquement, ils ne disposent d'aucune mémoire et ne sont pas capables de raisonner/communiquer sur des systèmes symboliques —, leurs interactions permettent au système d'adopter un comportement globalement *intelligent* pouvant résoudre des problèmes complexes.

1.2 De l'agent au multi-agent

Jusqu'à présent, nous n'avons parlé que de **l'agent** et de l'environnement. L'intérêt du domaine des systèmes multi-agents repose cependant clairement dans l'utilisation simultanée de plusieurs agents pour la modélisation de systèmes ou la résolution de problèmes.

L'ajout d'agents permet ainsi au système de montrer de nouvelles propriétés . L'étude des systèmes multi-agents ne se résume cependant pas à l'observation du comportement d'agents isolés.

Les agents se trouvent au sein d'un environnement dans lequel ils évoluent ensemble et qu'ils modifient. Ces agents sont en *interaction* permanente. Il en résulte la création de boucles de rétroaction complexes. Ainsi, le comportement d'un agent pourra affecter le comportement des autres et générer, à un niveau systémique, de nouvelles propriétés impossibles à déduire depuis les spécifications du comportement des agents. Cette propriété fait des systèmes multi-agents une catégorie particulière des systèmes complexes [M.R.-Jean, 1997; Muller, 1996].

1.2.1 La complexité dans les systèmes multi-agents

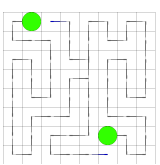
Commençons par une définition simple des systèmes complexes. Selon Wikipédia, *Un système complexe est constitué d'un grand nombre d'entités en interaction qui empêchent l'observateur de prévoir sa rétroaction, son comportement ou son évolution par le calcul.*

Ainsi, les systèmes complexes sont caractérisés par des comportements non linéaires qui mettent en échec les approches analytiques classiques, incapables de modéliser les interactions des composants du système. Les propriétés remarquables des systèmes complexes sont :

l'imprévisibilité du comportement global du système par des moyens analytiques : Comme nous l'avons déjà souligné précédemment, la présence de boucles de rétroaction[10] et les interactions entre les nombreuses entités composant le système sont responsables de sa non linéarité. Il devient alors pratiquement impossible d'obtenir une modélisation analytique satisfaisante du système (il est éventuellement possible de modéliser le système dans un espace de fonctionnement défini par linéarisation mais lorsque le système s'éloigne de cet espace de fonctionnement, le modèle peut perdre tout pouvoir explicatif et prédictif ; il s'agit là d'une des méthodes de l'automatique pour le contrôle de systèmes dynamiques non-linéaires.). Dans ces conditions, plutôt que de chercher à effectuer une étude analytique exacte du système, on préférera le *simuler* (étudier le système de façon incrémentale en suivant ses lois d'évolution à partir d'un état initial particulier).

l'émergence de structures/de propriétés : Les interactions entre les entités composant le système ont généralement pour effet de produire des propriétés, des structures cohérentes ou des motifs sans que ces éléments soient explicitement décrits dans les spécifications des entités composant le système [M.R.-Jean, 1997]. On parle alors d'*émergence* ou d'*auto-organisation* [Resnick, 1994]. Les termitières sont ainsi fabriquées par les termites sans pourtant qu'aucune d'elles ne possède de plan de construction ou n'ait même

10. Le terme boucle de rétroaction (en anglais, *feedback*) représente le fait que le résultat d'un comportement puisse le modifier dans le futur. On pensera par exemple à l'utilisation de l'erreur dans une boucle de régulation.



conscience de son existence. Un autre exemple connu est celui des *gliders* et des *canons à gliders* dans le jeu de la vie de Conway (cité dans [Gardner, 1977]).

la sensibilité aux conditions initiales : L'évolution du système peut être profondément modifiée par des variations des conditions initiales. L'image habituellement utilisée pour illustrer ce phénomène est celle de l'effet papillon : cette métaphore météorologique dit que le battement des ailes d'un papillon peut déclencher une tempête à l'autre bout du monde ("*Predictability : Does the Flap of a Butterfly's Wings in Brazil Set off a Tornado in Texas ?*", Lorenz). L'effet papillon fait référence aux phénomènes d'amplification observables dans les boucles de rétroaction. Une variation infime d'un paramètre pourra faire basculer le système vers un attracteur ou un autre ou encore faire entrer le système dans un état *chaotique*. On pourra également se référer à la théorie du chaos et en particulier à l'*exposant de Lyapunov* qui permet d'estimer la durée après laquelle l'amplification de l'erreur sur les conditions initiales rend impossible toute prévision du comportement du système (c'est notamment pour cela que les prévisions météorologiques sont imprécises au delà de quelques jours).

la robustesse (ou l'absence de robustesse) aux perturbations : Cette propriété exprime la capacité du système à conserver son organisation (sa structure) lorsqu'il est perturbé. La redondance présente dans le système (en particulier due au nombre d'entités le constituant) lui permet de résister, dans une certaine mesure, aux perturbations — par exemple, des modifications des conditions environnementales ou la perte d'un certain nombre des entités constituant le système [Drogoul, 1993].

convergence des systèmes complexes — A un horizon fini, les systèmes complexes ont tendance à converger vers des attracteurs (nous pouvons considérer que ces attracteurs minimisent l'énergie du système). Il est intéressant, dans le cadre de leur étude, de savoir vers quel(s) type(s) d'attracteurs peuvent converger les systèmes complexes. L'étude des attracteurs est par exemple un aspect important dans le domaine des automates cellulaires [Fatès, 2004].

Ces attracteurs peuvent être plus ou moins simples à détecter et permettent de déterminer si le système est entré dans un régime stable. On peut distinguer 4 classes d'attracteurs :

les points fixes correspondent à un état unique dans lequel le système va s'arrêter.

les attracteurs simples correspondent à un ensemble fini d'états autour desquels le système va osciller ou évoluer de manière cyclique.

les attracteurs complexes ou étranges correspondent à un ensemble fini d'états que le système va visiter de façon imprévisible (ou stochastique).

le chaos Il est également possible, par des mécanismes d'amplification et de récurrence, que le système ne converge jamais vers un régime stable (un sous ensemble fini d'états du système). Il est alors dit chaotique.

1.2.2 Etude par simulation

Nous avons vu que, de par la nature complexe des systèmes multi-agents, il est impossible de prédire leur comportement de manière analytique. L'étude de ces systèmes se fait donc

itérativement, par la simulation de leur évolution. A chaque itération de la simulation, il est alors nécessaire d'exécuter la boucle perception-décision-action de chaque agent puis de mettre à jour l'état du monde (c'est à dire de répercuter dans l'environnement (et sur les agents) le résultat des actions des agents).

Modèles d'exécution — Nous avons déjà dit précédemment que les systèmes multi-agents reposent implicitement sur les interactions entre agents et qu'ils sont potentiellement sensibles aux perturbations qui peuvent engendrer des phénomènes de *bifurcation* entre des régimes périodiques et le chaos. Cet état de fait pose problème lorsqu'il s'agit de simuler le système. En effet, les biais de simulation, qui vont de la représentation du temps ou de l'espace à la façon de résoudre les conflits entre agents pour l'utilisation d'une même ressource, peuvent modifier en profondeur les propriétés et le comportement du système simulé. On réfère à l'ensemble de ces paramètres de simulation sous le terme de modèle d'exécution. Un des grands problèmes du domaine provient de l'absence de spécification de ces modèles d'exécution au sein des modèles multi-agents [Michel *et al.*, 2001; Axtell, 2001; Weyns and Holvoet, 2003]. Il s'agit d'un point particulièrement gênant pour la reproductibilité des expérimentations en particulier lorsqu'il s'agit de modèles explicatifs [11].

Le modèle Influence/Réaction — [Ferber and Muller, 1996] proposent le modèle Influence/Réaction qui est défini comme un formalisme général de la théorie de l'action et qui se base sur la distinction entre les influences produites par les agents et la réaction de l'environnement (Influence/Réaction interdit la modification directe de l'environnement par les agents). Cette distinction implique une résolution *explicite* des conflits entre agents et introduit la prise en charge de la simultanéité des actions. Ce formalisme apporte une réponse partielle au problème des modèles d'exécution puisqu'il devient possible de formuler *sans ambiguité* le fonctionnement d'une simulation. Toutefois, cette étape de formalisation n'est pas indispensable lors de la conception de modèles multi-agents puisqu'il est tout à fait possible de négliger le modèle d'exécution et de laisser les conflits se régler de façon implicite par les contraintes imposées par la plateforme d'implémentation. Le modèle Influence/Réaction est en outre difficile à implémenter [12], ce qui explique peut être sont utilisation limitée. Notons également qu'il n'existe pas, dans Influence/Réaction, de méthodologie permettant de déduire le modèle d'exécution (qui définit l'ensemble des hypothèses d'exécution) à partir de résultats expérimentaux.

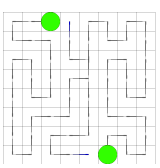
Seulement des agents —

1.3 L'intelligence en essaim

L'intelligence collective (ou encore intelligence en essaim, *swarm intelligence* en anglais), qui a émergé de l'intelligence artificielle distribuée au début des années 90, tire son inspiration de

11. Les modèles explicatifs ont pour vocation de reproduire le comportement d'un système réel, qu'il s'agisse d'expliquer les mécaniques de son fonctionnement (comment fonctionne la construction d'un nid de termites ?) ou de prédire l'évolution d'un système réel (quel temps fera-t'il demain ?).

12. même si une adaptation du principe aux simulations individu centrées a été proposée [Michel, 2007] et que des méthodes d'agentification de toutes les entités simulées [Kubera *et al.*, 2010] facilitent son implémentation



phénomènes biologiques — en particulier de l'étude des insectes sociaux tels que les fourmis, les termites ou les abeilles.

Ce sont des systèmes composés d'agents simples (l'intelligence en essaim est une sous catégorie des systèmes multi-agents réactifs) mais dont les interactions peuvent cependant produire une solution à des problèmes complexes, notamment à des problèmes de tri, d'optimisation de parcours ou de routage et d'ordonnancement [Dorigo and Gambardella, 1997].

Les agents disposent de capacités très limitées. Ils ne peuvent percevoir et agir que sur leur environnement immédiat, ne sont capables de communiquer que de manière indirecte en déposant de l'information dans l'environnement, ne possèdent généralement aucune mémoire et ne réalisent leurs décisions qu'en réagissant à leur perceptions.

Quelques exemples emblématiques du domaine sont :

- les *boids* de Reynolds [Reynolds, 1987] qui reproduisent des comportements de déplacement en formation tels que nous pouvons les observer dans les nuées d'étourneaux ou les bancs de poissons ;
- les fourmis fourrageuses [Deneubourg *et al.*, 1990] qui construisent et optimisent des chemins entre leur nid et des sources de nourriture ;
- l'ACO (Ant Colony Optimization) de Dorigo [Dorigo and Gambardella, 1997] qui s'inspire lui aussi des fourmis et qui apporte une réponse au problème du voyageur de commerce (même si son appartenance à l'intelligence collective fait débat puisqu'il s'agit d'un algorithme centralisé).

Les approches relevant de l'intelligence en essaim s'inspirent principalement de deux disciplines, la physique pour les champs de potentiels et la biologie pour les approches à base de phéromones.

1.3.1 Les champs de potentiels

Inspirés de la physique, ces approches reposent sur l'utilisation de champs scalaires ou, plus généralement, de champs vectoriels. Ainsi, chaque point de l'espace est associé à un vecteur. En observant leur voisinage, les agents peuvent déduire des informations de gradient utiles à la résolution du problème [Khatib, 1985; Brooks, 1986; Arkin, 1987]. En pratique, ce type d'approche est principalement utilisé pour des tâches de navigation. D'après [Ferber, 1995], les objets présents dans l'environnement (agents compris) émettent des signaux (dont l'intensité est fonction de la distance à l'émetteur) définissant des champs attractifs (pour les buts) ou répulsifs (pour les obstacles). La figure 1.3 présente un tel champ avec un but et deux obstacles.

Le principal problème concernant les champs de potentiel provient de l'existence de maxima et minima locaux qui peuvent agir comme des *pièges à agents*, les empêchant de mener leur tâche à bien.

Nous pouvons noter que les champs de potentiels ont également été utilisés pour des tâches différentes de la navigation. Par exemple, [Simonin, 2001] propose un modèle de foraging dans lequel un agent ayant trouvé une source de nourriture émet un signal attractif pour attirer les autres agents. [Moujahed, 2007] propose quant à elle un modèle pour le positionnement

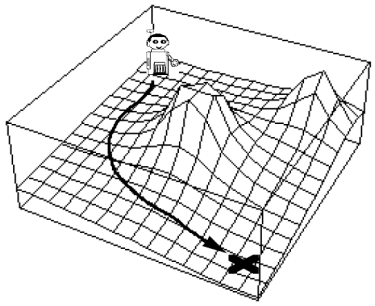


FIGURE 1.3 – Représentation 3D d'un champ de potentiel tiré de [Ferber, 1995].

d'usines et de dépôts afin d'optimiser le coût de transport des marchandises des usines vers les dépôts et des dépôts vers les clients.

1.3.2 Les algorithmes fourmi et les phéromones digitales

La stigmergie

La stigmergie est un mode de communication indirect entre des agents naturels ou artificiels. Ce concept a été introduit en 1959 par le biologiste Pierre-Paul Grassé [Grassé, 1959] qui étudiait la construction des nids chez les termites.

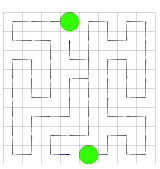
D'après lui, la coordination entre les ouvrières et la répartition des tâches ne dépendent pas des insectes (pas de plan de construction, pas d'organisation hiérarchique) mais de la construction elle-même. En effet, les insectes sociaux (termites, fourmis, araignées sociales, etc...) utilisent l'environnement comme moyen de communication indirect (via le dépôt de phéromones).

Le concept de stigmergie a directement inspiré le concept de phéromones digitales.

Les phéromones digitales

Les phéromones digitales sont le pendant artificiel des phéromones utilisées par les insectes que les agents inscrivent sur une matrice modélisant l'environnement. On a pu voir apparaître, dès la fin des années 80, des algorithmes de foraging (fourragement, soit la recherche et le transport de ressources) basés sur le comportement des fourmis [Bonabeau *et al.*, 2001]. Plus tard, Dorigo [Dorigo and Gambardella, 1997] propose une méthode ACO (Ant Colony Optimisation), elle aussi basée sur le dépôt de phéromones, pour l'optimisation de parcours dans un graphe (problématique du voyageur de commerce, Travelling Salesman Problem). Plus récemment Parunak [Parunak *et al.*, 2002] propose des modèles de coordination entre drones (véhicules sans pilotes) pour la surveillance et la poursuite, eux aussi fondés sur des mécanismes reposant sur les phéromones digitales.

Les avantages des phéromones digitales par rapport aux champs de potentiels proviennent principalement des mécanismes d'évaporation et de diffusion qui permettent respectivement une persistance (temporelle) limitée et une diffusion de l'information dans l'environnement.



L'utilisation de phéromones présuppose toutefois l'existence d'un environnement actif capable de calculer ces deux processus. Notons également que la vitesse de propagation des phéromones est limitée. Il ne s'agit donc pas d'un moyen de communication privilégié pour des communications rapides à longue distance.

Les algorithmes fourmis

Cette classe d'algorithme est inspirée par le comportement des fourmis, en particulier par les mécanismes à base de phéromones mis en évidence par l'expérience sur le fourragement de Deneubourg [Deneubourg *et al.*, 1990]. Il s'agit de modèles combinant des agents simples aux perceptions locales qui n'agissent que par réaction aux stimuli (c'est-à-dire aux phéromones déposées dans l'environnement) et un environnement capable d'évaporer et diffuser les phéromones digitales déposées par les agents. Les modèles appartenant à cette classe sont tous construits sur un même modèle de perception/marquage/déplacement mais utilisent des sémantiques de marquage différentes et entretiendraient un lien de parenté avec la programmation dynamique [Koenig *et al.*, 2001].

1.3.3 Modélisation informatique des mécanismes d'évaporation et de diffusion

La communication par phéromones fait appel à des boucles de rétroaction, à la fois positives (dépôt supplémentaire de phéromones sur une trace déjà existante) et négatives (évaporation des phéromones qui conduit à leur disparition qui permettent de réguler le phénomène). Une boucle de rétro-action positive permet de renforcer et d'accélérer la résolution d'une tâche. Une boucle de rétro-action négative permet de l'atténuer lorsque la tâche est terminée ou d'éviter une amplification incontrôlée du processus. Les phéromones digitales, version informatique des phéromones réelles, reposent elles aussi sur ces processus de régulation.

Nous définissons ici les phéromones digitales comme un couple *(type, quantité)*.

- Le type permet aux agents de distinguer différents types de phéromones. On pourra ainsi utiliser des types différents pour des tâches indépendantes entre elles ou encore combiner des phéromones attractives et répulsives
- La quantité de phéromone en un point de l'environnement est représentée pas un nombre flottant, permettant ainsi la création de gradients que les agents pourront descendre ou remonter.

Il est à noter que les processus à base de phéromones digitales sont difficilement implémentables et utilisables dans des représentations continues (principalement à cause du coût des calculs). Nous faisons donc l'hypothèse d'un environnement discret (graphe, grille/lattice) dont les phéromones marqueront les nœuds ou les arcs.

Nous présentons ci-après les processus d'évaporation et de diffusion à la base du modèle des phéromones digitales permettant respectivement la disparition et la propagation de l'information.

Le processus d'évaporation

L'évaporation est un phénomène qui mène progressivement à la disparition de la phéromone. Ce processus a pour but de faire disparaître les informations obsolètes et contribue à la création de gradients, c'est-à-dire à des traces orientées, que les agents pourront suivre. Ce processus peut être modélisé par une suite géométrique (l'évolution dans le temps de la quantité de phéromone sera donc discrète). Nous posons $q_n \in \mathbb{R}$, la quantité de phéromone en un point au temps $n \in \mathbb{N}$ et le coefficient d'évaporation $coefEvap \in]0;1[$. Nous avons donc :

$$q_{n+1} = q_n \times (1 - coefEvap), q_0 = constante \tag{1.1}$$

Nous sommes en présence d'une suite monotone strictement décroissante qui converge vers 0+ (voir figure 1.4). Ce processus modélise donc une phéromone qui ne s'évaporera jamais totalement. Ainsi, si la variation du taux d'évaporation agit bien sur la vitesse d'évaporation — ce qui peut éventuellement permettre de donner plus d'importance à un type de phéromone qu'à une autre —, il ne fera jamais disparaître complètement la phéromone. Ainsi, dans l'hypothèse où les agents disposent de capteurs parfaits, le taux d'évaporation n'influe pas sur leurs perceptions. Dans le cas de capteurs imparfaits — des capteurs bruités et/ou incapables de détecter la phéromone en dessous d'un certain seuil —, la phéromone pourra, pour les agents, sembler avoir disparu et modifier leur comportement.

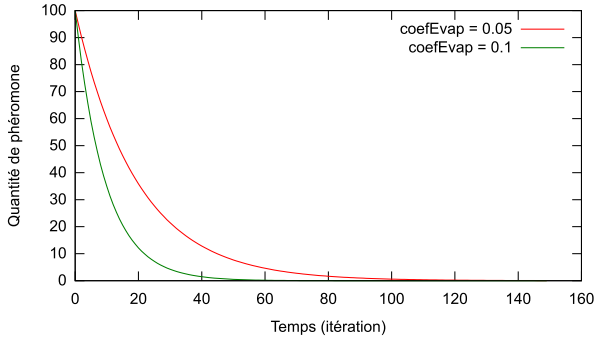


FIGURE 1.4 – Evaporation de la phéromone selon deux taux d'évaporation différents

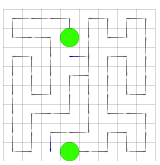
Calcul du temps écoulé depuis le dépôt de la phéromone — Le modèle EVAP, présenté plus loin dans la thèse, se fonde sur la mesure du temps écoulé depuis le dépôt d'une marque de phéromone. Nous montrons ici comment calculer cette durée à partir de la valeur courante d'une marque, connaissant le taux d'évaporation et la quantité initialement déposée.

La suite étant strictement monotone et connaissant le premier terme q_0 de la suite, nous pouvons calculer la valeur q_n pour un n donné (il s'agit là de résultats connus pour les suites géometriques).

La valeur du $n^{ième}$ terme de la suite sera :

$$q_n = q_0 \times (1 - coefEvap)^n \tag{1.2}$$

A partir de l'équation 1.2 et connaissant la quantité courante q_n, nous pouvons déterminer



le temps n écoulé depuis le dépôt de la phéromone.

$$n = \frac{log(\frac{q_n}{q_0})}{log(1 - coefEvap)} \tag{1.3}$$

Paramètrage du comportement des phéromones — Le phénomène de durée de vie infinie de l'information évoqué précédemment peut s'avérer gênant dans certaines applications (comme par exemple à cause de la persistance d'informations périmées et/ou qui ne sont plus pertinentes). Il est cependant possible d'introduire, pour éliminer ce problème, un seuil ε sous lequel les agents ne perçoivent pas la phéromone.

A partir de l'équation 1.2 et en substituant une valeur cible à la valeur courante, nous pouvons déterminer la quantité de phéromones q_0 à déposer pour qu'elle passe sous le seuil de perception ε des agents après un délai n.

$$q_0 = \frac{\varepsilon}{(1 - coefEvap)^n} \tag{1.4}$$

Le processus de diffusion

La diffusion est le phénomène qui propage de proche en proche une fraction de la quantité de phéromone présente sur un nœud. Il existe plusieurs implémentations possibles de la diffusion modifiant légèrement la dynamique du champ de phéromones, principalement le long des obstacles et des bords de l'environnement. Nous présentons ici la technique de diffusion implémentée dans de nombreuses plateformes de simulation multi-agent.

Le principe de la diffusion est de transférer aux nœuds adjacents, en la répartissant équitablement, une part de la phéromone présente sur un nœud. Il s'agit donc d'un processus isotrope qui formera, en faisait l'hypothèse d'un environnement régulier et infini, des *équipotentielles* centrées sur la cellule contenant le dépôt initial. Dans le cas des environnements de type grille, nous limitons la diffusion au voisinage de *von Neumann* (c'est-à-dire en 4-connexité).
Soit $q_i(t)$, la quantité de phéromones présente sur le nœud i à l'instant t, n le nombre de nœuds voisins et $coefDiff \in]0;1[$, le coefficient de diffusion. Nous posons la loi d'évolution de q_i :

$$q_i(t+1) = (1 - coefDiff) \times q_i(t) + \sum_{j=voisinage(i)} \frac{coefDiff}{n} \times q_j(t) \tag{1.5}$$

L'équation 1.5 est décomposable en deux termes. Le premier $((1 - coefDiff) \times q_i(t))$ correspond à la quantité de phéromone restante et le second à la somme des quantités reçues des nœuds voisins. A un horizon temporel très grand et l'absence d'évaporation, le processus de diffusion répartira homogènement la phéromone à travers l'environnement. Puisque nous n'utilisons pas de diffusion dans cette thèse, nous ne détaillerons pas davantage les problèmes rencontrés dans l'exploitation de ce processus.

1.4 Conclusion du chapitre

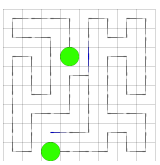
Dans ce chapitre, nous avons présenté les systèmes multi-agents. Ce paradigme, qui entretient un lien de parenté avec les systèmes complexes, permet d'approcher des problèmes et la

modélisation de systèmes difficilement accessibles aux méthodes classiques de par leur degré de complexité.

La résolution des problèmes s'effectue à travers les interactions entre agents. On assiste ainsi à une auto-organisation du système, c'est-à-dire à l'émergence d'une structuration de ce système qui n'est pourtant pas inscrite explicitement dans le comportement des agents.

La redondance et la simplicité des agents (en particulier dans les approches réactives) permettent au système de montrer des propriétés intéressantes de robustesse aux perturbations qui lui permettent de résister et de continuer à fonctionner, même lorsqu'une partie des agents n'est pas en mesure de réaliser leur tâche. Les systèmes multi-agents cachent cependant une part d'imprévisibilité puisqu'ils sont susceptibles de produire subitement des comportements indésirables ou de changer de régime.

Nous nous trouvons donc face à une approche proposant de nombreux points forts mais pour laquelle il n'existe, dans le cas général, aucune assurance sur la qualité de son comportement global.



Chapitre 2

La Patrouille multi-agent

Nous présentons, dans ce chapitre , le problème de la patrouille multi-agent qui est devenu au cours de ces dernières années un banc d'essai du multi-agent. Dans sa formulation initiale, la patrouille multi-agent consiste à effectuer des couvertures répétitives de l'environnement, c'est-à-dire de visiter les nœuds d'un graphe le plus souvent possible. Dans le cadre d'un graphe connu, résoudre cette tâche de manière optimale est un problème NP-complet.

Nous présentons d'abord les deux variations du problème de la patrouille, à savoir, la couverture répétitive de l'environnement, et la recherche et la poursuite d'intrus. Nous nous intéressons ensuite aux représentations de l'environnement et leurs implications sur la patrouille. Nous présentons également les critères de performances permettant d'évaluer la qualité d'une patrouille. Nous terminons le chapitre en nous intéressant aux approches existantes pour le problème de la patrouille.

2.1 Le problème de la patrouille multi-agent — définition

Nous pouvons définir le problème de la patrouille multi-agent comme le fait de surveiller efficacement un environnement à l'aide d'un groupe d'agents afin d'assurer sa protection. En particulier, nous cherchons à empêcher toute intrusion (par exemple pour la surveillance d'un bâtiment) ou à *contrôler* l'apparition d'évènements (par exemple pour la prévention des feux de forêt ou encore la surveillance de réseaux ou de sites web [Andrade *et al.*, 2001; Cho and Garcia-Molina, 2000]).

Une patrouille efficace revient donc à déterminer et à appliquer une politique de patrouille (c'est-à-dire la fonction permettant aux agents de choisir la prochaine action à effectuer) minimisant (ou maximisant) une fonction objectif de performances définie en fonction du scénario de patrouille envisagé.

En effet, la protection de l'environnement peut être envisagée selon plusieurs points de vue. La littérature définit deux variantes majeures du problème de la patrouille multi-agent de natures profondément différentes et répondant chacune à des impératifs et des besoins particuliers :

- La première variante considère qu'une patrouille efficace revient à visiter aussi souvent que possible l'ensemble des lieux accessibles dans l'environnement. Maximiser la

fréquence de visite, permet un *suivi* (détection d'un événement ou d'un changement d'état) efficace de l'environnement mais n'offre pas de solution réellement pertinente à la capture d'intrus (en particulier si ces derniers sont capables d'observer les agents patrouilleurs et de modéliser leur politique de patrouille).

– La seconde variante considère qu'une patrouille efficace revient à rechercher et capturer d'éventuels intrus aussi vite que possible, la couverture exhaustive de l'environnement n'étant pas considérée comme un critère d'efficacité pertinent. Pour cette variante, les intrus sont généralement capables de modéliser les agents patrouilleurs afin de pouvoir leur échapper au mieux.

Nous présentons ci-après ces deux variantes principales de la patrouille multi-agent.

2.1.1 La patrouille vue comme la visite répétitive de l'environnement

Pour cette variante de la patrouille multi-agent, on cherche à visiter l'ensemble des lieux accessibles de l'environnement le *plus souvent* et le *plus régulièrement* possible à l'aide d'un groupe d'agents. Cette représentation du problème est principalement issue des domaines de la décision (et relève en particulier de la planification), des systèmes multi-agents réactifs et de la robotique mobile.

Nous considérons ici l'environnement comme étant un espace discret représenté par un graphe. On cherchera à *minimiser une fonction objectif reposant sur un critère d'occupation* des nœuds.

Le problème pourra alors être considéré comme un problème de décision classique revenant à chercher le circuit le plus court visitant tous les nœuds ou couvrant l'intégralité de l'environnement (respectivement dans des cadres discrets et continus).

Cette variante sera principalement destinée à la détection d'évènements statiques et durables au sein de l'environnement.

Ce type d'approche sera donc particulièrement pertinent pour des tâches telles que la détection de départs de feux de forêt par un groupe de drones[13] ou la surveillance de la modification de sites web pour leur indexation.

La couverture de l'environnement, une tâche connexe à la patrouille —

La couverture consiste simplement à explorer l'ensemble de l'environnement, c'est-à-dire à visiter au moins une fois chaque nœud de l'environnement. L'efficacité d'un algorithme en couverture est simplement mesurée par la durée d'une couverture complète de l'environnement. Cette durée est souvent désignée sous le terme de *blanket time*. Une couverture optimale consiste, dans un cadre mono-agent, à suivre un *chemin hamiltonien*. Dans un contexte multi-agent, cela revient à diviser l'environnement en un ensemble de chemins hamiltoniens (un par agent) ne s'intersectant pas (c'est-à-dire que chaque agent visitera un nœud inexploré à chaque itération).

13. Les drones sont des appareils volants sans pilotes, autonomes ou opérés à distance.

Notons cependant que l'obtention d'une couverture optimale ne garantit pas une patrouille optimale. Si l'on est tenté de considérer intuitivement la patrouille comme une couverture répétée de l'environnement, un simple exemple prouve le contraire. La figure 2.1.a présente une couverture optimale sur un environnement grille. En suivant cette stratégie, une fois que l'agent a effectué la couverture de l'environnement, celui-ci se retrouve dans une situation où sa prochaine décision sera nécessairement sous optimale puisque la cellule de plus grande oisiveté correspondra au point de départ de l'agent (ici en haut à gauche). Une patrouille optimale correspondra cependant à la répétition d'une couverture optimale si ses points de départ et d'arrivée sont adjacents (cf. figure 2.1.b).

a. une couverture optimale b. une patrouille optimale

FIGURE 2.1 – Une couverture optimale ne signifie pas nécessairement une patrouille optimale

2.1.2 Perception d'intrus à distance

La seconde variante principale de la patrouille multi-agent est issue des domaines des systèmes multi-agents cognitifs, de la théorie des jeux et de la décision dans l'incertain, et se concentre sur la *capture d'un ou plusieurs intrus* se déplaçant dans l'environnement [Basilico *et al.*, 2009; Marino *et al.*, 2009].

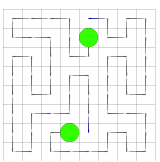
La patrouille est alors modélisée comme un jeu avec adversaire où l'on cherche une politique maximisant l'utilité des agents (voir figure 2.2), c'est-à-dire une politique maximisant l'espérance des gains futurs.

Les adversaires sont modélisés comme des agent rationnels capables d'observer les agents patrouilleurs et cherchant eux aussi à maximiser leur utilité. Dans certains cas, les agents patrouilleurs sont capables de modéliser le comportement des intrus afin d'augmenter leur propre utilité.

Pour cette variante du problème de la patrouille multi-agent, la couverture de l'environnement n'est pas un critère de performance pertinent (il ne s'agit ici pas d'un but explicite pour les agents). On ne pourra donc, dans cette optique, ni garantir une visite exhaustive de l'environnement (couverture répétitive) ni se donner de contraintes sur la fréquence de visite.

2.1.3 Problème considéré

Ces deux grands types d'approches au problème de la patrouille sont donc très différents, tant dans les objectifs affichés que dans la façon d'y répondre. Un modèle construit pour un type de problème ne répondra que partiellement aux besoins de l'autre classe.



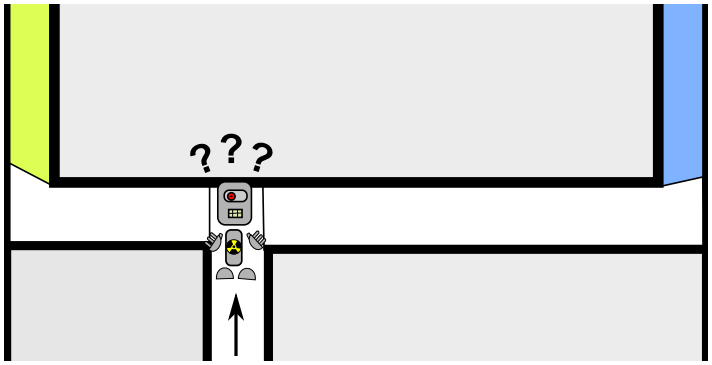


FIGURE 2.2 – Maximisation de l'utilité pour la capture d'intrus. Sachant qu'un intrus a été récemment perçu à cet endroit, l'agent aura intérêt à se déplacer vers le couloir de gauche. En effet, des deux zones (en vert et en bleu) contenant de l'information (sur la présence ou l'absence d'intrus), l'agent rejoindra la zone vert plus rapidement et minimisera ainsi le temps perdu dans l'éventualité ou l'intrus ne s'y trouve pas.

Nous nous positionnons dans la suite de cette thèse sur le problème de la patrouille défini comme la visite répétitive des lieux accessibles de l'environnement afin de minimiser le délai entre deux visites consécutives.

2.2 Représentation des environnements discrets

Cette thèse se concentrant sur les algorithmes fourmi, nous ne considérons ici que les environnement discrets. Ceux-ci sont représentés sous forme de graphes, les arcs étant les chemins valide entre les nœuds représentant les lieux à visiter.

Si les environnements discrets sont toujours représentés sous la forme de graphes, nous pouvons cependant les considérer et les interpréter de différentes façons en fonction de la manière dont ils sont décrits. Nous présentons ici deux représentations — une représentation topologique et une représentation métrique — et expliquons leur influence sur la tâche de patrouille.

2.2.1 Représentation topologique de l'environnement

Cette représentation de l'environnement ne considère que les lieux stratégiques à surveiller en priorité et les chemins qui les relient (souvent avec une indication sur leur longueur).

L'environnement est alors représenté sous la forme de graphes valués. Il s'agit de graphes généraux (implicitement considérés comme connexes[14]) orientés ou non, et dont les arcs sont valués relativement à la distance séparant les nœuds qu'ils relient.

Les nœuds du graphe sont alors considérés comme les lieux accessibles et les arcs comme les chemins possibles reliant ces différents lieux.

14. Un graphe est connexe lorsque, pour tout couple de nœuds du graphe, il existe un chemin les reliant.

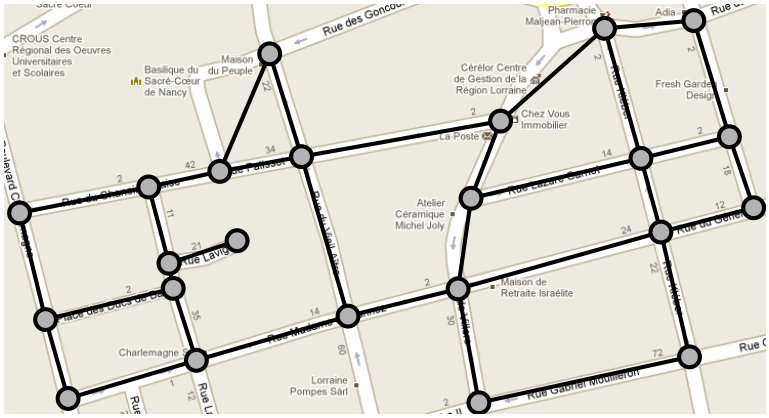


FIGURE 2.3 – Représentation d'un environnement (un quartier de Nancy) par un graphe pondéré. Les arcs sont valués relativement à la distance séparant les nœuds (les valeurs ne sont pas représentées sur le schéma). *Fond de carte tiré de google maps.*

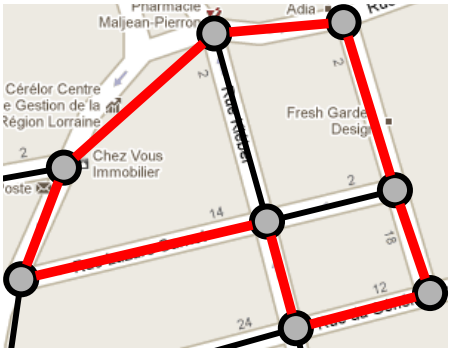
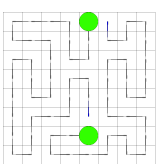


FIGURE 2.4 – Un cycle (arcs rouge en gras) visitant tous les nœuds du graphe sans réaliser de couverture de l'environnement à proprement dire (les zones couvertes par les arcs noirs restent non visités).

Nous pouvons rapprocher ce type de représentation des cartes routières (figure 2.3) où les nœuds représentent les intersections entre les routes et les arcs les rues. Les particularités de cette représentation de l'environnement peuvent avoir des implication sur la réalisation de la patrouille :

- Cette représentation est particulièrement adaptée à la description d'environnements de grande taille. Il s'agit là d'une propriété particulièrement intéressante pour les approches reposant sur des techniques de planification (voir section 2.4.1), le temps de calcul nécessaire à l'obtention d'une solution par ces approches augmentant souvent de façon exponentielle avec la taille du graphe.

- La visite exhaustive des nœuds du graphe ne garantie pas la *couverture* de l'environnement. Il est en effet généralement possible de trouver un chemin (voir un cycle, soit un chemin dont le début et la fin coïncident) visitant l'ensemble des nœuds du graphe sans passer par tous les arcs (cf. figure 2.4). Dans l'exemple donné précédemment (voir figure 2.3), cela reviendrait à visiter l'ensemble des intersections sans se soucier des rues. Il peut potentiellement s'agir d'un problème si la tâche requière de visiter l'ensemble



des rues (par exemple, la recherche de fuites de gaz par une voiture renifleuse).
Il est donc important de construire le graphe en fonction de la tâche à réaliser (la représentation de l'environnement proposé sur la figure 2.3 est particulièrement inadapté pour une tâche requérant la visite de chaque maison de chaque rue).

2.2.2 Représentation métrique de l'environnement

Cette représentation correspond à une approximation discrète de l'environnement. Nous considérons ici que l'espace patrouillé par les agents est recouvert par un pavage régulier de zones élémentaires.

L'environnement sera donc représenté par une *grille* (aussi connu sous le nom de *lattice* en anglais ou *réseau* en français), un graphe particulier, qui correspond à un maillage de nœuds couvrant un espace continu de façon régulière (voir figure 2.5).

Dans ce contexte, nous désignons les nœuds par le terme *cellule*. Deux cellules sont connectées entre elles selon leurs relations spatiales (c'est-à-dire qu'on ne pourra accéder à une cellule a_i à partir d'une cellule a_j que si elles sont adjacentes). Notons que l'environnement ainsi décrit est donc *isotrope*, c'est-à-dire que la distance entre tout couple de cellules adjacentes est constante. Nous pouvons donc considérer que cette distance est unitaire. Le degré des cellules définit la forme du pavage. Ainsi pour un degré 3, le maillage sera triangulaire ; pour un degré 4, nous obtenons des carrés ; enfin pour un degré 6, la grille sera hexagonale (il s'agit là des seuls pavages réguliers existants dans un espace de dimension 2). Bien que la patrouille puisse être réalisée dans des espaces de dimensions supérieures, nous ne considérons pas de tels environnements dans cette thèse.

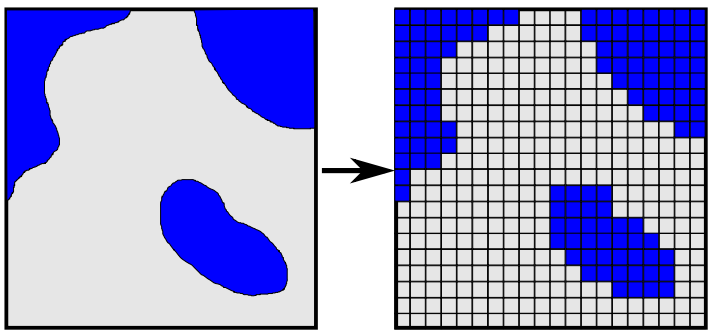


FIGURE 2.5 – Représentation d'un environnement continu par une grille régulière composée de nœuds de degré 4.

Comme pour la représentation de l'environnement par un graphe à arcs pondérés, nous pouvons souligner plusieurs particularités de ce type de représentation :

- Le principe de la représentation par grille régulière consiste à associer chaque point de l'environnement à une zone élémentaire (typiquement, une zone pouvant s'inscrire dans le champ de perception d'un agent). Ainsi :
  1. une visite exhaustive de l'ensemble des nœuds du graphe obtenu garantit une couverture totale de l'environnement à patrouiller et tout élément statique sera détecté ;

2. la granularité de l'environnement doit être adaptée aux capacités (champ de perception) des agents.

– Le grand nombre de nœuds composant l'environnement peut être un obstacle à la planification d'une patrouille efficace. Ce type de représentation est donc mieux adapté aux contextes où les agents décident de leurs actions en cours d'exécution, comme c'est le cas, par exemple, pour les algorithmes fourmi.

En conclusion, ce type de représentation est particulièrement intéressant dans le cadre d'une patrouille en environnement réaliste — en opposition à des environnements virtuels tels qu'un réseau — à l'aide d'agents situés dont les capacités perceptives sont limitées. Il est en outre plus facile, en utilisant cette représentation, de construire la carte d'un environnement initialement inconnu puisqu'il suffit de marquer les cellules comme accessibles (ou inaccessibles) lors de la visite des lieux qu'elles représentent.

2.3 Critères de performance pour la patrouille en environnement discret

Afin d'étudier l'interêt d'une approche particulière pour la résolution du problème de la patrouille, nous avons besoin de pouvoir mesurer ses performances. Nous présentons ici les deux critères utilisés dans la littérature.

2.3.1 Critères basés sur l'oisiveté

Le premier critère à considérer, de par sa large utilisation dans de nombreux travaux [Almeida *et al.*, 2004], est l'oisiveté. Aussi connue sous son appellation anglaise d'*idleness*, l'oisiveté correspond au temps écoulé depuis la dernière visite d'un nœud par un agent. Il s'agit, à l'origine, d'un critère définit au niveau du nœud mais l'agrégation de cette donnée au niveau du graphe permet de représenter la qualité du système à un niveau global.

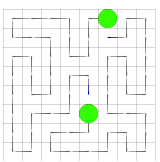
Plusieurs critères basés sur l'oisiveté des nœuds permettent d'évaluer la qualité d'une patrouille comme définis dans [Machado *et al.*, 2002] :

INI – Instantaneous Node Idleness l'INI correspond à l'oisiveté d'un nœud du graphe à un instant donné, c'est-à-dire au temps écoulé depuis sa dernière visite ;

IGI – Instantaneous Graph Idleness l'IGI correspond à la moyenne des INI de l'ensemble des nœuds du graphe à un instant donné ;

WGI – Worst Graph Idleness la WGI correspond à l'INI la plus importante de l'ensemble des nœuds du graphe à un instant donné et représente donc l'oisiveté de la cellule la plus anciennement visitée ;

AGI – Average Graph Idleness l'AGI correspond à la moyenne des IGI sur une fenêtre temporelle donnée ; typiquement, sur la durée de la patrouille.



2.3.2 Critères basés sur la fréquence de visite

Un second critère de performances, moins répandu que l'oisiveté, repose sur la maximisation de la fréquence de visite des nœuds du graphe [Elmaliach *et al.*, 2007]. Nous pouvons distinguer plusieurs critères relatifs à la fréquence de visite des nœuds :

Fréquence uniforme Ce critère consiste à observer la variance entre les fréquences de visite des nœuds de l'environnement. Une patrouille optimale est atteinte lorsque tous les nœuds sont visités à une fréquence f telle que $f = \frac{k}{n}$, avec k le nombre d'agents patrouilleurs et n le nombre de nœuds de l'environnement.

Fréquence moyenne Ce critère repose simplement sur la moyenne des fréquences de visite des nœuds du graphe. Il est donc équivalent à l'IGI pour l'oisiveté. Il reste toutefois intéressant lorsque le critère d'uniformité ne peut être atteint — c'est-à-dire lorsque l'environnement n'admet pas de cycle hamiltonien.

2.3.3 Calcul de l'oisiveté optimale

Afin de juger de l'efficacité d'une politique de patrouille, il est intéressant de pouvoir la comparer à une borne d'optimalité. Il est cependant impossible de définir cette borne par des moyens calculatoires, la topologie même des graphes patrouillés ayant une influence sur la forme de la solution optimale. Il serait donc nécessaire de réaliser une analyse de chaque graphe pour recherche un plan de patrouille optimal. Nous calculons un minorant

a. $t = t_0$ b. $t = t_0 + 1$

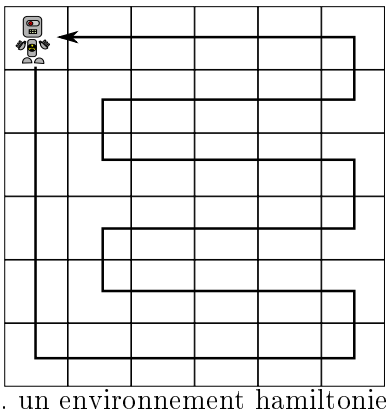
FIGURE 2.6 – En visitant toujours, à un instant t, le nœud le plus anciennement visité, l'agent se trouvera toujours au temps $t+1$ sur un nœud voisin du nœud le plus anciennement visité. Le système suit alors un comportement cyclique optimal.

Une patrouille optimale

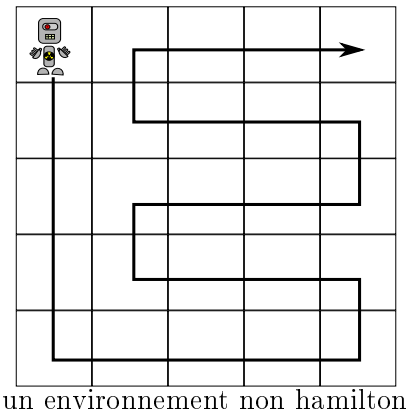
Une politique de patrouille optimale consiste simplement à toujours visiter le nœud (dans le cadre mono-agent) ou les nœuds (dans le cadre multi-agent) dont le délai depuis la dernière visite est le plus important. Pour cela, il faut que les nœuds les plus anciennement visités soient voisins des nœuds sur lesquels se trouvent les agents. Lorsque cette condition est remplie, le système visite les nœuds toujours dans le même ordre et entre alors dans un *cycle hamiltonien* (voir figure 2.6). La particularité des cycles hamiltoniens est de visiter *chaque nœud du graphe*

exactement une fois au cours du cycle. Il s'agit donc d'une solution optimale à la patrouille. Dans le cadre multi-agent, une solution communément utilisée est de déployer les agents à intervalles réguliers le long d'un tel cycle [Almeida *et al.*, 2004]. Si l'oisiveté moyenne d'une patrouille optimale sur un environnement admettant un cycle hamiltonien est facile à calculer, prouver qu'un environnement admet un cycle hamiltonien est un problème en soi (il s'agit d'un des 21 problèmes NP complets de Karp [Karp, 1972]).

Tous les environnements n'admettent toutefois pas de cycle hamiltonien (voir figure 2.7). Dans ce cas, la recherche d'un cycle optimal est alors encore plus ardue.



a. un environnement hamiltonien (6x6 cellules)



b. un environnement non hamiltonien (5x5 cellules)

FIGURE 2.7 – Deux environnements grille carrés, admettant ou non un cycle hamiltonien, l'un de coté de taille paire et l'autre de coté de taille impaire.

Calcul d'une borne d'oisiveté optimale

Nous nous contentons donc, à défaut de mieux, de poser un minorant sur la borne d'oisiveté moyenne optimale (à partir de laquelle nous déduisons la borne de pire oisiveté).

Ce minorant à la borne optimale est calculé en faisant l'hypothèse de l'existence d'un cycle hamiltonien dans l'environnement qui correspond donc à une patrouille optimale (il s'agit là de considérer l'hypothèse la plus optimiste et cette borne ne sera pas nécessairement atteignable).

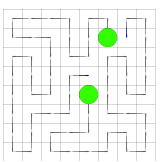
Les valeurs théoriques d'optimalité d'oisiveté sont calculées comme suit : Soit v le nombre de nœuds accessibles de l'environnement. Nous considérons que chaque agent visite un nœud à chaque itération.

Un agent seul visitera donc l'ensemble de l'environnement en v itérations. Lorsque l'agent finit son parcours, le nœud le plus anciennement visité aura donc une oisiveté de $v-1$ (puisque l'oisiveté du nœud courant est égale a 0). Pour n agents, l'oisiveté maximale vaudra donc $\frac{v}{n}-1$.

Les valeurs d'oisiveté peuvent être décrites par une suite arithmétique de raison 1 et de premier terme $u_0 = 0$ et dont la moyenne au n^{ieme} terme vaut donc $\frac{u_n}{2}$.

– L'oisiveté moyenne instantanée optimale (IGIo) vaut donc :

$$IGIo = \frac{\frac{v}{n} - 1}{2} \tag{2.1}$$



– L'oisiveté maximale instantanée optimale (WGIo) vaut donc :

$$WGIo = \frac{v}{n} - 1. \tag{2.2}$$

2.4 Approches existantes

Nous présentons ici quelques modèles issus de différents domaines de l'intelligence artificielle pour la patrouille multi-agent.

2.4.1 Techniques relevant de la recherche opérationnelle

Optimisation par colonie de fourmis — Ant Colony Optimization

Souvent considérée comme relevant des algorithmes fourmi, l'optimisation par colonie de fourmis, proposée par Dorigo et Gambardella [Dorigo and Gambardella, 1997], est en fait une *métaheuristique d'optimisation* inspirée du comportement de fourragement des fourmis. Il s'agit, à l'origine, d'une méthode de planification *centralisée* bioinspirée utilisée pour la recherche de chemins optimaux dans des graphes, comme par exemple pour le problème du voyageur de commerce qui consiste à rechercher le cycle le plus court visitant tous les nœuds d'un graphe pondéré, c'est-à-dire rechercher le chemin hamiltonien le plus court dans ce graphe. Il a donc été naturel d'adapter cette approche au problème de la patrouille pour lui faire rechercher des cycles hamiltoniens utilisés comme plans de patrouille par les agents.

L'approche ACO repose intégralement sur les mécanismes de renforcement de traces de phéromones, chez certaines espèces de fourmis, identifiés par Deneubourg [Deneubourg *et al.*, 1990] dans le cadre de la tâche de fourragement (cf. section 1.3). Ainsi, en parcourant le graphe, les fourmis déposent derrière elles une quantité fixée de phéromones. Le comportement des fourmis est régi par un processus stochastique [15] les guidant préférentiellement vers les arcs possédant le plus de phéromones. Après que les fourmis aient visité l'ensemble des nœuds, les phéromones sont évaporées, la quantité évaporée dépendant de la longueur de l'arc (plus un arc est long, plus le coefficient d'évaporation est élevé). Suit une sélection de la meilleure solution (chaque fourmi est indépendante et n'est pas influencée par les marquages des autres fourmis) qui servira d'état initial à l'itération suivante de l'algorithme.

Ce comportement définit une boucle de rétroaction positive qui renforce les chemins les plus courts au détriment des autres (voir figure 2.8).

Dans le cas général cette approche ne possède toutefois aucune garantie d'optimalité. Notons également qu'il ne s'agit pas d'un "algorithme fourmi" dans le sens où nous l'entendons dans cette thèse (cf. 2.4.4) puisque son mode de fonctionnement rappelle les méta-heuristiques d'optimisation comme les algorithmes génétiques (une construction itérative de la solution par sélection de la meilleure solution intermédiaire et re-exécution de l'algorithme en utilisant cette dernière comme état initial).

15. Un arc possédant plus de phéromones aura plus de chance d'être sélectionné par une fourmi. Ce comportement favorise l'exploration face à un processus décisionnel déterministe.

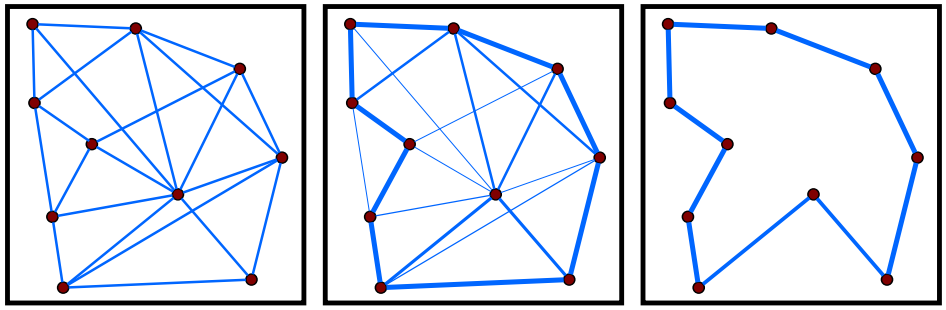


FIGURE 2.8 – Calcul du cycle le plus court par optimisation par colonie de fourmi. A l'initialisation (à gauche), tous les arcs possèdent la même quantité de phéromones. Au cours de l'exécution (au centre), les agents renforcent certains arcs ; les arcs non renforcés disparaissent progressivement (à droite) jusqu'à ne laisser plus qu'un seul chemin. L'expérience est ensuite répétée jusqu'à obtention d'une solution satisfaisante.

Spanning trees

La méthode proposée par [Gabriely and Rimon, 2001] est une heuristique reposant sur l'utilisation de *spanning trees* et permettant de construire des *cycles hamiltoniens* — soit une stratégie de patrouille optimale — en *temps polynomial* (nous rappellons que la recherche d'un cycle hamiltonien dans un graphe est un problème NP complet).

En théorie des graphes, les *spanning trees* permettent la représentation de graphes par des arbres de taille finie (voir figure 2.9). Cette représentation en arbre permet de représenter l'ensemble des nœuds du graphe sans risquer de rencontrer d'éventuels boucles ou cycles.

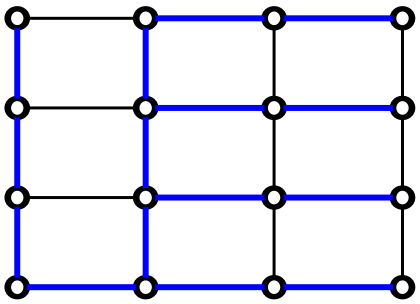
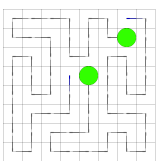


FIGURE 2.9 – Un spanning tree (arcs bleus) d'un graphe grille

La méthode proposée repose donc sur l'utilisation de tels arbres. Considérant un environnement grille avec des cellules de côté de longueur n, les auteurs lui superposent une seconde grille, de granularité plus importante, avec des cellules de côté de longueur $2n$ (voir figure 2.10). Ainsi, à chaque cellule du graphe *grossier* sont associées 4 cellules du graphe original. Un algorithme de construction [16] de *spanning tree* est ensuite exécuté sur le graphe *grossier*. L'agent n'a alors plus qu'à suivre le spanning tree (chemin fléché, figure 2.10) pour effectuer une patrouille optimale de l'environnement.

16. Il s'agit d'un simple algorithme de parcours de graphe (en profondeur ou en largeur) notant les nœuds déja rencontrés.



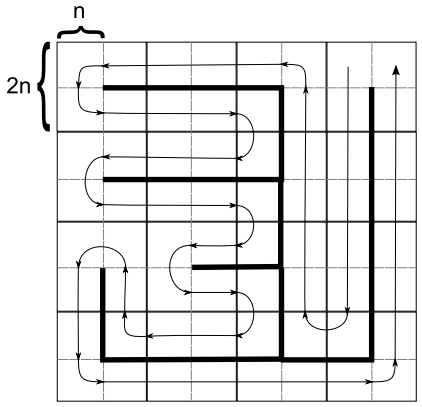


FIGURE 2.10 – Un cycle hamiltonien réalisé à partir d'un spanning tree sur un environnement grille de 8x8 nœuds.

Si cette méthode permet de planifier une patrouille optimale très rapidement (en temps linéaire $O(N)$, N étant le nombre de cellules de l'environnement), elle reste cependant assez limitée et pose des contraintes fortes sur l'environnement :

- Le graphe patrouillé est contraint à être une grille à voisinage de Von Neumann (chaque nœud à au plus 4 voisins) dont les cotés sont de longueurs paires.
- Les obstacles présents dans l'environnement peuvent rendre cette méthode inopérante si leur densité est importante.

2.4.2 Apprentissage par renforcement

Dans [Almeida *et al.*, 2004], Almeida *et al.* comparent plusieurs approches pour la patrouille multi-agent dont deux méthodes reposant sur l'apprentissage par renforcement. L'apprentissage par renforcement revient à associer des situations à des actions. Pour cela, ils utilisent le formalisme des processus décisionnels de Markov (MDP) qui associent à chaque couple état-action une récompense calculée par rapport au critère à optimiser. Pour simplifier le principe, nous pouvons considérer que les agents chercheront à maximiser leur espérance de gains (et donc la performance de leur comportement à long terme) en apprenant une politique à partir des récompenses résultant de leur actions.

Si les performances de ces approches ne sont pas à remettre en question (puisqu'elles font parti des meilleures parmi les approches proposées dans cet article), elles posent des contraintes fortes sur leur utilisation. Le graphe représentant l'environnement doit être connu à l'avance et la politique de patrouille des agents doit être replanifiée pour chaque environnement. Ce type d'approche est également sujet à une explosion combinatoire, les auteurs estiment la taille de l'espace d'état à *200.000 états par agent* pour un environnement de seulement 50 nœuds environ (il n'est donc pas envisageable d'exploiter ce type d'approches avec une représentation métrique de l'environnement). De plus, la qualité de la politique obtenue est souvent dépendante des paramètres d'apprentissage, sachant qu'il n'existe pas de méthode pour déterminer une bonne valeur (à part réaliser une exploration des paramètres).

2.4.3 Agents cognitifs

Dans [Faratin *et al.*, 2002], Faratin *et al.* proposent une architecture cognitive pour la patrouille multi-agent reposant sur un mécanisme de négociation. L'environnement est partagé (aléatoirement) en sous-ensembles de nœuds qui seront chacun assignés à un agent. Chaque agent analyse l'ensemble de nœuds qu'il doit patrouiller. Lorsqu'un agent détecte un nœud qu'il ne pourra pas visiter en temps raisonnable (c'est-à-dire un nœud trop éloigné du reste du sous-ensemble), il le met aux "enchères". Les autres agents chercheront alors à échanger ce nœud contre l'un des leurs ; la transaction sera alors effectuée avec l'agent offrant en contrepartie le nœud le plus intéressant (c'est-à-dire l'échange qui maximise l'utilité des agents vendeur et acheteur).

Durant sa thèse [Pellier, 2005], Pellier développe un modèle multi-agent cognitif de co-construction de plan dans lequel les agents négocient l'exécution de tâches en fonction des ressources disponibles. Bien que n'étant pas à la base prévu pour la tâche de patrouille, des applications proches laissent penser que ce modèle pourrait être adapté à ce problème.

Ce type d'approche suppose aussi que le graphe servant de support à l'environnement soit préalablement connu. Il est éventuellement possible d'adapter la politique de patrouille des agents en cours d'exécution (suite à des modifications de l'environnement ou à l'ajout ou la perte d'agents) mais cela suppose que les agents soient tous capables de communiquer entre eux et que le réseau de communication soit capable de supporter l'afflux de transactions entre les agents, très coûteux en bande passante.

2.4.4 Algorithmes fourmi

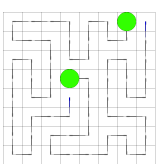
Dans [Koenig *et al.*, 2001], Koenig analyse et compare un groupe d'algorithmes fourmi pour la patrouille. Dans cet article, il met en avant les ressemblances entre ces algorithmes. Ces derniers reposent en effet sur un squelette commun, décrit par l'algorithme 1, et ne diffèrent que par la sémantique du marquage. Cependant, si les algorithmes fourmi pour la patrouille se ressemblent beaucoup, ces derniers produisent des comportement très différents.

Algorithme 1 : Une itération du comportement d'un agent fourmi type

```
1 Choix d'un nœud destination parmi le voisinage;
2 Marquage du nœud courant;
3 Déplacement de l'agent vers le nœud sélectionné;
```

Il existe peu de comparaisons entre les algorithmes fourmi et les autres approches, en particulier du fait des différences de représentation de l'environnement. Les approches classiques utilisent principalement une représentation par graphe pondéré (à cause du faible nombre de nœuds nécessaires à la représentation de l'environnement) alors que les algorithmes fourmi considèrent l'environnement comme une grille. Les algorithmes fourmi ne sont pas explicitement conçus pour opérer sur des graphes valués (voir section 2.2.1) et les autres approches ne sont pas très à l'aise sur des graphes comportant un nombre élevé de nœuds.

Si cette classe d'algorithmes semble moins performante que d'autres types d'approches — notamment à cause de la vision locale des agents et de l'absence de communications directes



qui empêche la planification de stratégies de coopération élaborées —, les algorithmes fourmi proposent des capacités uniques inaccessibles aux approches *offline*. En particulier, ces algorithmes présentent des capacités intéressantes de robustesse à la perte d'agents et apportent des réponses aux problèmes de gestion d'énergie là où d'autres approches nécessiteraient une replanification du plan de patrouille. Au delà de cela, la sous-optimalité de leur politique de patrouille leur confère même un avantage. L'imprévisibilité de leur comportement — même à un horizon court — empêche d'éventuels intrus d'exploiter les régularités que l'on peut trouver avec d'autres approches. L'aspect *online* de ces modèles et le critère de décision des agents basé uniquement sur une perception locale permet également leur fonctionnement sur des environnements inconnus [17].

Nous présentons ici certains de ces modèles, en particulier les approches proposées par Wagner qui appuie souvent ses résultats expérimentaux par des preuves formelles.

Node Counting

Algorithme 2 : Une itération du comportement d'un agent Node Counting

1 $x \leftarrow minValue(voisinage(v_{courant}))$;
2 $m(v_{courant}) \leftarrow m(v_{courant}) + 1$;
3 $moveTo(x)$;

L'approche Node Counting [Thrun, 1992] vise à assurer un nombre de visites homogène pour l'ensemble des nœuds de l'environnement. Pour cela, les agents incrémentent à chaque visite d'un nœud la valeur de son marquage (voir algorithme 2). De ce fait, le marquage des nœuds indique le nombre de visites qu'il a reçu depuis le démarrage de la tâche (d'où le nom du modèle).

L'interêt de cet algorithme pour la patrouille reste toutefois limité. En effet, la vision locale des agents les mène souvent à faire des choix sous optimaux ; par exemple en les emmenant vers un "puits" de valeurs basses, les forçant à visiter les cellules de ce puits jusqu'à ce que leurs valeurs soient plus hautes que le "bord", ce qui est nécessaire pour s'en échapper. Ce comportement forcera les agents à préférer de petits groupes de cellules. Le principal handicap de ce modèle provient donc de la faible corrélation entre le *sens* du marquage effectué par les agents et la tâche qu'ils ont à effectuer.

Learning Real Time A*

Proposé dans [Korf, 1990], LRTA* est à l'origine conçu pour la recherche du plus court chemin entre deux points dans des graphes connus ou inconnus.

Utilisé dans le cadre de la patrouille multi-agent, LRTA* réalise à chaque instant une estimation de la distance au nœud de plus bas marquage le plus proche.

17. Un environnement est inconnu lorsque les agents ne possèdent pas de connaissance a priori ni sur sa topologie ni sur ses dimensions. Il reste toutefois nécessaire de pouvoir borner l'environnement pour éviter que les agents n'aillent se perdre dans la nature

Algorithme 3 : Une itération du comportement d'un agent LRTA*

1 $x \leftarrow minValue(voisinage(v_{courant}))$;
2 $m(v_{courant}) \leftarrow m(x) + 1$;
3 $moveTo(x)$;

Un nœud n'ayant pas encore été visité aura une valeur de 0. Ainsi, d'après le comportement des agents défini par l'algorithme 3, un nœud de valeur *val* se trouvera donc à une distance *val* du nœud non visité le plus proche. Les agents exploitent ensuite ce marquage pour se diriger vers les nœuds inexplorés.

Ce comportement est particulièrement intéressant pour l'exploration puisque lorsque qu'un agent se trouve confronté à un choix entre deux nœuds de même valeur, il laisse derrière lui une piste permettant :

- à ce même agent de retourner sur ses pas si son voisinage est "peu intéressant" (c'est-à-dire si la valeur de la marque qu'il a déposé à l'itération précédente est inférieure à la valeur des autres marques de son voisinage), ou
- à un autre agent croisant cette "piste" de la remonter jusqu'au nœud inexploré.

Une fois, la première couverture effectuée (c'est à dire quand tous les nœuds ont un marquage supérieur a 0), les agents LRTA* vont suivre le même comportement pour visiter les nœuds de marquage égal à 1 (puis de même pour les marquages à 2, 3, ...).

Ce comportement est très intéressant pour l'exploration de l'environnement. Il n'est toutefois pas très performant pour la patrouille puisqu'à l'image de node counting, LRTA* ne repose pas sur la notion d'oisiveté.

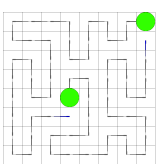
Les approches de Wagner

Algorithme 4 : Une itération du comportement d'un agent EAW

1 choix de l'arc e de marquage minimum partant du nœud courant;
2 $e \leftarrow t + 1$;
3 déplacement le long de l'arc choisi;
4 $t \leftarrow t + 1$;

Dans les modèles que propose Wagner, le comportement des agents est directement lié a une mesure de l'oisiveté. Les agents choisissent de se déplacer localement vers les cellules de plus forte oisiveté. Wagner présente plusieurs modèles, reposant sur les mêmes mécanismes de marquage de la date. Edge Ant Walk (voir algorithme 4) repose sur le marquage des arcs du graphe (mais visite les nœuds). Vertex Ant Walk est une variante qui repose sur le marquage des nœuds. Notons toutefois que l'algorithme VAW à été, sous ce même nom, décliné en deux versions.

- Une première version préliminaire (à laquelle nous référerons par VAW_0, voir algorithme 5) repose uniquement sur l'oisiveté [Wagner *et al.*, 1999].
- Une seconde version, qui est en fait une fusion du VAW original et de Node Counting a été proposée et étudiée plus en détails dans [Yanovski *et al.*, 2001]. Cette seconde



Algorithme 5 : Une itération du comportement d'un agent VAW_0

$x \leftarrow minValue(voisinage(v_{courant}))$;
$val(v_{courant}) \leftarrow t$;
$moveTo(x)$;
$t \leftarrow t + 1$;

Algorithme 6 : Une itération du comportement d'un agent VAW

$x \leftarrow minValue(\mu(vois(v_{courant})), \tau(vois(v_{courant})))$;
$\mu(v_{courant}) \leftarrow \mu(v_{courant}) + 1$;
$\tau(v_{courant}) \leftarrow t$;
$moveTo(x)$;
$t \leftarrow t + 1$;

version (voir algorithme 6), repose sur l'utilisation de deux types de marquages à la fois. Le premier correspond au nombre de visites reçues par un nœud (μ) et le second au délai depuis sa dernière visite (τ). L'agent choisira pour destination le nœud voisin de marquage μ minimal. En cas d'égalité entre deux nœuds, il choisira celui dont le délai depuis la dernière visite τ est le plus important.

Un point remarquable des travaux de Wagner concerne les preuves formelles posées sur ses modèles. Ainsi, il pose une borne de couverture, assurant que ses modèles couvrent l'intégralité de l'environnement en temps fini. Il remarque également que les algorithmes EAW et VAW convergent vers des stratégies en cycles éventuellement optimales.

2.5 Conclusion du chapitre

Nous avons distingué deux variantes majeures de la patrouille multi-agent. La première consiste à localiser un intrus se déplaçant dans l'environnement et la seconde à la visite répétitive et exhaustive de l'ensemble des lieux accessibles de l'environnement. Il s'agit de deux problèmes très différents et nous nous intéressons à la seconde variation.

Nous avons présenté plusieurs types d'approches à ce problème. Les approches par planification ou apprentissage posent cependant des contraintes fortes sur la complexité de l'environnement ou sur le nombre d'agents utilisable. Ainsi, le temps nécessaire à la planification d'une patrouille augmente rapidement avec le nombre de nœuds à patrouiller alors que les méthodes d'apprentissage sont limitées à un faible nombre d'agents. De plus, les perturbations pouvant survenir dans le système peuvent éventuellement provoquer une forte dégradation (voir même l'interruption) de la patrouille. Une implémentation robotique de telles approches ne pourrait donc être réalisée que dans un environnement contrôlé.

Les algorithmes fourmi, quant à eux, s'ils présentent des performances sous optimales du fait des capacités limitées des agents, proposent des agents autonomes capables de s'adapter aux perturbations et de compenser d'éventuelles défaillances.

Deuxième partie

EVAP - un algorithme fourmi pour la patrouille

Cette partie de la thèse présente EVAP, un algorithme fourmi conçu pour patrouiller un environnement discret. Au-delà de l'aspect "résolution du problème de la patrouille, nous nous intéressons au comportement de cet algorithme et en particulier aux propriétés auto-organisées apparaissant lors de son exécution. Plus précisément, nous étudions la capacité du système à s'adapter à diverses conditions de simulation (environnements de complexité croissante, résistance du système à l'ajout/suppression d'agents ou aux changements d'ordonnancement). Une grande part de ce travail concerne également l'étude du phénomène d'auto-organisation en structures cycliques que nous avons observé lors des expérimentations et qui garantissent au système une bonne oisiveté. Nous démontrons formellement la convergence systématique du système vers ces attracteurs cycliques et examinons leur propriétés.

Ce travail puise en partie sa source dans les travaux de Wagner [Wagner *et al.*, 1999] qui propose des algorithmes qui garantissent la visite répétitive de tous les nœuds d'un environnement discret avec des performances compétitives en terme d'oisiveté. Ces algorithmes ne requièrent que peu d'informations sur leur environnement — les agent ne possèdent pas de carte ni de mémoire des lieux visités — puisque chaque agent ne décide de sa prochaine action qu'à partir de ses perceptions locales, c'est-à-dire des marques présentes sur les nœuds adjacents à sa potition.

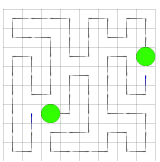
EVAP repose uniquement sur l'évaporation d'une phéromone digitale déposée par les agents lors de la visite des nœuds du graphe. Le comportement des agents est simplement défini par une descente du gradient de phéromones qui les mène vers les zones de forte oisiveté. Après une première exploration de l'environnement et une phase d'organisation du champ de phéromones par les agents, nous avons pu observer que le système tend à s'auto-organiser vers des cycles stables. Ces cycles représentent des solutions optimales ou proches de l'optimal en terme d'oisiveté pour le problème de la patrouille [Almeida *et al.*, 2004].

Nous proposons dans le chapitre 3 l'algorithme EVAP utilisant une phéromone digitale et son équivalent EVAW utilisant un marquage basé sur la date de visite des nœuds.

Le chapitre 4 présente les performances de l'algorithme EVAP en termes d'oisiveté avant sa convergence vers des cycles stables sur des environnements divers. Nous y étudions et expliquons en particulier les difficultés rencontrées par EVAP lors de la première exploration de l'environnement. Nous comparons ensuite EVAP à CLInG, un autre algorithme fourmi reposant sur la propagation de l'information à travers l'environnement et qui semble proposer une solution aux problèmes rencontrés par EVAP. Nous profitons également de ce chapitre pour nous intéresser à la robustesse de l'approche que nous proposons. En plus d'étudier l'effet du bruit nous proposons d'examiner les réactions du système face à l'ajout et la suppression dynamique d'agents.

Le chapitre 5 introduit le formalisme utilisé pour l'étude des attracteurs cycliques. Il recense également les différentes catégories de comportements cycliques vers lesquels peut converger EVAP ainsi qu'un moyen de les détecter. Nous présentons également un détecteur automatique de comportements cycliques que nécessite l'étude expérimentale de ces attracteurs.

Le chapitre 6 se concentre sur l'étude théorique et expérimentale du comportement de convergence d'EVAP vers des cycles. Nous prouvons, à l'aide du formalisme définit en chapitre 5, la convergence systématique du modèle EVAP vers un comportement cyclique stable qui, nous le rappelons, produit une patrouille de très bonne qualité. Cette preuve ne fournit cependant pas de résultats quantitatifs. Nous illustrons donc expérimentalement les résultats



théoriques obtenus précédemment sur des environnements de complexité croissante.

Finalement, le chapitre 7 s'éloigne un peu de la problématique de la patrouille pour s'intéresser à des problèmes implémentatoires. Les systèmes multi-agents relevant des systèmes complexes des changements dans les hypothèses d'exécution (on réfère à l'ensemble de ces hypothèses sous le terme de modèle d'exécution) peuvent entraîner la perte ou l'apparition de nouvelles propriétés. Nous étudions donc les effets de ces biais d'implémentation (et en particulier dans la cas de l'implémentation robotique) sur le comportement d'EVAP.

Chapitre 3

Le modèle EVAP

Sommaire

3.1 EVAP - un algorithme fourmi fondé sur le dépôt d'informations 38
3.1.1 Principe du modèle EVAP 38
3.1.2 Définition du modèle 39
3.1.3 Représentation de l'environnement 42
3.1.4 Un premier aperçu du comportement d'EVAP 43
3.2 EVAW - un EVAP sans phéromones 47
3.2.1 Les modèles VAW et EVAW 48
3.2.2 Équivalence entre les marquages 49
3.2.3 Marquage statique contre marquage dynamique 50
3.3 Quelques propriétés du modèle EVAP 51
3.3.1 Borne sur le temps de couverture 51
3.3.2 Propriété de patrouille 53

Nous présentons dans ce chapitre le modèle EVAP, un algorithme fourmi dédié à la patrouille multi-agent en environnement discret et inspiré du comportement des insectes sociaux.

La couverture ou la patrouille est réalisée de manière totalement décentralisée par des agents non communicants ne nécessitant que très peu d'information sur l'environnement. Leur comportement est uniquement régi par leurs perceptions locales de l'environnement. Ils n'ont donc besoin ni de carte de l'environnement, ni de mémoire interne.

En 2001, Koenig [Koenig *et al.*, 2001] mène une étude comparative sur plusieurs de ces algorithmes fourmi pour la patrouille, en particulier Learning RealTime A* (LRTA*), Node Counting et VAW. Il en ressort que ces différents algorithmes mélangent planification et exécution (Les agents replanifient leur prochaine action à chaque itération avec une profondeur de recherche très limitée, généralement de 1 pour conserver le principe de perception locale) et ne diffèrent qu'en fonction de la sémantique du marquage (cf. 2.4.4).

EVAP appartient à cette catégorie d'algorithmes et repose sur la seule évaporation de la phéromone déposée par les agents lors de la visite d'un nœud du graphe de l'environnement (le processus de diffusion n'est pas utilisé). La quantité évaporée représente alors directement le temps écoulé depuis la dernière visite du nœud. Le comportement des agents défini comme la descente du gradient de phéromone les guide localement vers le nœud d'oisiveté la plus élevée (cf. section 2.3.1).

Nous décrivons d'abord l'algorithme EVAP ainsi que les hypothèses nécessaires à son exécution sur un environnement actif, support des processus liés aux phéromones digitales. Nous introduisons ensuite EVAW, en section 3.2, un algorithme *équivalent* à EVAP basé sur le marquage de la date courante, ne nécessitant donc pas d'environnement actif.

3.1 EVAP - un algorithme fourmi fondé sur le dépôt d'informations

3.1.1 Principe du modèle EVAP

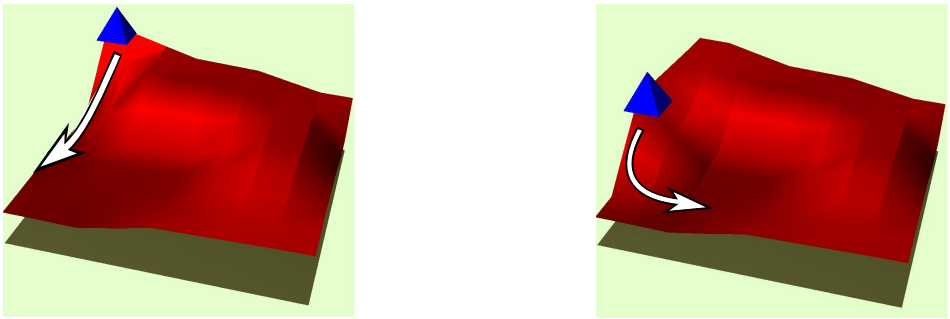
EVAP repose sur une organisation collective des agents à travers des communications indirectes. Celles-ci se font par l'intermédiaire de phéromones digitales déposées dans l'environnement par des agents autonomes qui ne décident de leur prochaine action qu'en fonction de leurs perceptions locales.

Ce type d'algorithme s'adapte particulièrement bien aux spécificités du problème de la patrouille en environnement inconnu. En effet, l'un des principaux intérêts de cette approche est qu'il s'agit d'un processus réalisé *online* qui ne nécessite que très peu d'information sur l'environnement (les agents ne possèdent pas de carte ou de mémoire de leurs actions) et ne requière aucune planification préalable (les agents prennent leurs décisions en fonction de leurs seules perceptions locales). Il n'est de ce fait pas soumis aux contraintes de complexité et de taille des approches par planification, lui permettant ainsi de s'exécuter sur des problèmes de grande taille, tant en nombre d'agents que dans la taille de l'environnement.
Ces algorithmes possèdent en outre une grande robustesse aux perturbations telles que des modifications de la topologie de l'environnement ou l'ajout et la suppression d'agents en cours d'exécution. De plus le comportement global du système est difficilement prévisible, même à court terme. C'est d'un avantage qui peut être utile pour des tâches de surveillance.

La grande majorité des travaux portant sur la patrouille multi-agent considèrent l'environnement à patrouiller comme un graphe $G(V, E)$ dont les nœuds V représentent les lieux à visiter et les arcs E, les liens reliant les nœuds entre eux. Une patrouille efficace revient à minimiser l'oisiveté des nœuds du graphe, c'est-à-dire à minimiser le temps écoulé entre deux visites consécutives d'un même lieu (cf. section 2.3.1).

Pour réaliser cette tâche, EVAP repose sur des agents simples marquant les nœuds de l'environnement à chaque visite avec une quantité fixe de phéromone dont la seule propriété est de s'évaporer après son dépôt (le processus de diffusion n'est pas utilisé dans le modèle).

Comme montré en section 1.3.3, cette évaporation permet de mesurer le temps écoulé depuis le dépôt de la phéromone et permet aux agents de comparer l'oisiveté relative entre deux nœuds. Le critère de performance à minimiser étant l'oisiveté, le comportement d'un agent



a. b.

FIGURE 3.1 – Descente du gradient de phéromone et marquage de l'environnement par un agent

sera de choisir et de se déplacer vers le nœud d'oisiveté la plus importante (soit celui dont la valeur de phéromone est minimale) et de marquer son passage par un dépôt de phéromones (figure 3.1). A un niveau global, le système tendra donc à effectuer une descente de gradient pour visiter les zones de plus grande oisiveté.

3.1.2 Définition du modèle

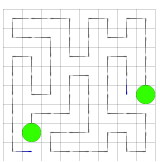
Nous présentons ici les algorithmes décrivant le comportement des agents et de l'environnement dans le modèle EVAP. Ces algorithmes ne permettent cependant pas d'expliquer comment le modèle EVAP est exécuté puisqu'ils ne spécifient pas si les agents sont activés de façon synchrone ou asynchrone ou à quelle fréquence l'environnement évapore les phéromones. Nous présenterons, dans un deuxième temps, les hypothèses d'exécution par défaut pour les expérimentations menées par la suite. Nous reviendrons également sur ce problème des modèles d'exécution au chapitre 7.

Notations : Chaque agent se trouve sur un nœud v appartenant à l'ensemble V des nœuds du graphe G représentant l'environnement. Chaque nœud v possède une liste de ses voisins notée $Vois(v)$ vers lesquels l'agent peut se déplacer. Nous définissons $m_{\varphi}(v)$ comme la valeur de la marque de la phéromone φ sur le nœud v. Ces marques appartiennent à l'intervalle $[0; Q_{max}]$.

Comportement des agents

Les agents EVAP sont des agents fourmi typiques (voir section 1.3.2). Leur comportement est défini par un cycle perception-déplacement-marquage et leur perception limitée au voisinage immédiat de leur position courante (l'ensemble des nœuds adjacents). L'algorithme 7 présente une itération du comportement d'un agent.

Pour minimiser l'oisiveté du graphe, les agents adoptent un comportement glouton, c'est-à-dire qu'ils choisissent toujours de se diriger vers la plus grande oisiveté soit le nœud y contenant la quantité de phéromone la plus basse (algorithme 7, ligne 1). Dans le cas d'une égalité entre plusieurs destinations, l'agent effectue un choix au hasard. Après leur déplacement vers le



Algorithme 7 : Une itération du comportement d'un agent EVAP

1 $y \leftarrow \arg\min_{v' \in Vois(v)}(m_{visite}(v'))$; `/* choix du nœud destination */`
2 $moveTo(y)$; `/* déplacement */`
3 $m_{visite}(v) \leftarrow Q_{max}$; `/* marquage du nœud destination */`

nœud choisi (algorithme 7, ligne 2), les agents déposent une phéromone marquant leur visite d'une quantité totale égale à Q_{max} [18] (algorithme 7, ligne 3). Cette quantité de phéromone Q_{max} peut être considérée comme représentant une oisiveté nulle.

Environnement

Cet algorithme de patrouille ne peut fonctionner sans l'activité d'évaporation réalisée par l'environnement. Ainsi, l'environnement effectue à la même fréquence une mise à jour de la valeur des nœuds du graphe selon la formule 1.1 présentée en section 1.3.3 (cf. algorithme 8 présenté ci-dessous).

Algorithme 8 : Une itération du processus d'évaporation par l'environnement

1 **pour chaque** *noeud* $v \in V$ **faire**
2 $\quad m(v) \leftarrow m(v) \times (1 - coefEvap)$;

Hypothèses d'exécution

Ces deux algorithmes, s'ils expliquent les comportements des agents et de l'environnement, ne précisent pas comment est exécuté le modèle, c'est-à-dire la manière d'activer les agents et d'exécuter les processus actifs de l'environnement.

Nous avons choisi dans la suite de cette thèse d'utiliser un ordonnancement asynchrone séquentiel cyclique. Cela signifie que les agents sont activés l'un après l'autre et dans un ordre lexicographique. Une fois que l'ensemble des agents a effectué trois actions (un mouvement), l'environnement s'active pour évaporer les phéromones. Cette contrainte, si elle pose toutefois des problèmes qui seront évoqués au chapitre 7, a l'avantage de résoudre le problème de l'accès concurrent à un nœud du graphe tel que présenté en figure 3.2.

Faisons l'hypothèse que deux agents $a1$ et $a2$ aient pour nœud voisin de marquage minimal le même nœud v. Une exécution simultanée des deux agents les mènerait sur le même nœud. Il s'agit d'une solution clairement sous optimale (un des agents aurait pu visiter un autre nœud) et n'est pas forcément possible dans le cas d'agents physiques qui ne peuvent pas occuper le même espace. Il faudrait dans ce cas résoudre le conflit par un choix arbitraire quelconque (premier arrivé, priorité selon l'identifiant de l'agent, gagnant à pierre-papier-ciseaux, etc.). Cet ordonnancement résout ce problème. Ainsi, lorsque l'agent $a1$ — qui aura la priorité de

18. Nous avons vu en section 1.3.3 que la fonction d'évaporation est définie par une suite convergeant en 0+ pour toute valeur initiale u_0 de phéromone. Le choix de cette valeur (tout comme le coefficient d'évaporation de cette phéromone) est donc arbitraire en simulation et peut être dirigé par la précision d'éventuels capteurs de phéromone dans le cadre d'une application robotique.

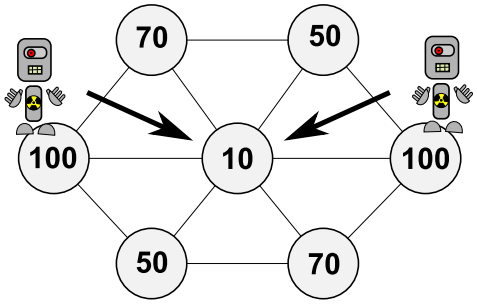


FIGURE 3.2 – Conflit entre deux agents pour l'accès simultané à un nœud

par son identifiant — se sera déplacé et aura marqué le nœud v, l'agent $a2$ devra choisir une nouvelle destination. Nous reviendrons au chapitre 7 sur l'adaptation de cette démarche au cadre réel de robots autonomes.

L'algorithme 9 présente le modèle EVAP tel qu'il sera considéré dans le reste de cette partie du manuscrit.

Algorithme 9 : Implémentation en simulation du modèle EVAP avec une exécution séquentielle cyclique des agents

```
1  pour chaque agent a faire
2      v ←nœud (donné ou choisi aléatoirement) ∈ V ;          /* initialisation */
3      moveTo(v) ;
4      m_visite(v) ← 0 ;
5  tant que vrai faire
6      pour chaque agent a faire
7          y ← arg min_{v'∈Vois(v)}(m_visite(v')) ;     /* choix du nœud destination */
8          moveTo(y) ;                                          /* déplacement */
9          m_visite(v) ← Q_max ;                  /* marquage du nœud destination */
10     pour chaque nœud v ∈ V faire
11         m(v) ← m(v) × (1 − coefEvap) ;                       /* évaporation */
```

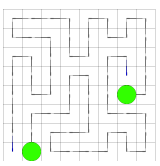
La figure 3.3 représente une itération de la boucle *tant que* de l'algorithme 9 :

a) Les agents se trouvent chacun sur le nœud qu'ils ont choisi à l'itération précédente.

b) L'agent a1 choisit sa destination, se déplace et dépose de la phéromone. Remarquons que nous sommes ici dans la configuration expliquée précédemment où deux agents ont pour nœud voisin de valeur minimale la même destination.

c) C'est au tour de l'agent $a2$ de faire de même. L'agent $a1$ ayant modifié la valeur d'un des nœuds voisins, $a2$ choisit une autre destination.

d) Tous les agents se sont déplacés ; L'environnement évapore les phéromones. A titre d'information, le coefficient d'évaporation coefEvap a été fixé a 0,9 dans nos simulations. Toutefois, la suite géométrique représentant l'évaporation présente une limite en 0+. Le taux d'évaporation n'a donc aucune influence sur la patrouille.



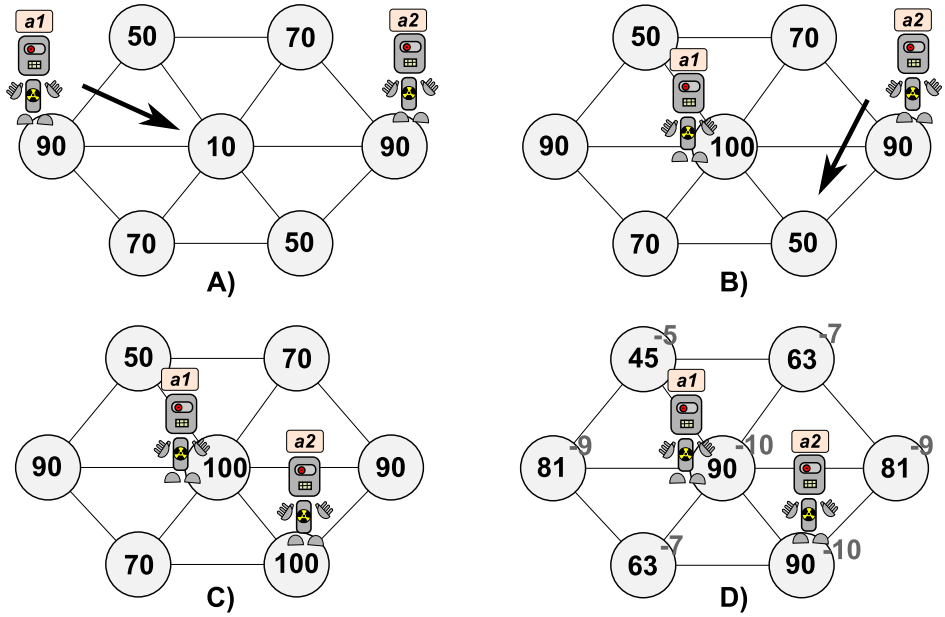


FIGURE 3.3 – Une itération du modèle EVAP tel que décrit par l'algorithme 9

3.1.3 Représentation de l'environnement

Nous définissons l'environnement discret à patrouiller comme un graphe $G(V, E)$ dont les nœuds V représentent l'ensemble des lieux accessibles et les arcs E les chemins reliant deux nœuds. Comme présenté en section 2.2, nous pouvons considérer cette représentation de l'environnement sous plusieurs angles :

- Sous la forme d'un graphe dont les nœuds représentent des points d'intérêt à visiter et les arcs (éventuellement valués) les connexions entre deux nœuds (cf. figure 3.4.a). Cette représentation de l'environnement correspond à la surveillance de points sensibles au sein d'une zone (par exemple des bâtiments sensibles dans une base militaire tels que des dépôts de munitions ou de carburant, des hangars à véhicules, etc...). Elle ne garantit pas la visite de tous les arcs du graphe par les agents. Pour plus de précisions, voir section 2.2.1).
  Notons que cette représentation suppose que les agents soient capables de percevoir la valeur d'un nœud distant pour effectuer une descente de gradient. Bien que cela soit possible, cette approche poserait de fortes contraintes quant à une implémentation avec des agents physiques. Nous référerons désormais à cette représentation sous le nom *d'environnement graphe*.

- Sous la forme d'un pavage régulier (pavage par triangles, carrés, hexagones) approximant l'environnement où chaque cellule de l'environnement correspond au champ de perception des agents (cf. figure 3.4.b). Les agents se déplacent alors de cellule en cellule pour effectuer une couverture totale de l'environnement. Notons qu'il n'est cependant pas possible, pour ce type de représentation, de définir explicitement des zones critiques à surveiller avec une priorité accrue. L'ensemble des cellules de l'environnement — même les endroits jugés "peu intéressants" — sera visité avec une régularité du même ordre. Nous référerons par la suite à cette représentation sous le nom d'*environnement grille*.

a. un environnement de type graphe b. un environnement de type grille

FIGURE 3.4 – Deux représentations possibles de l'environnement

Si EVAP est capable d'évoluer indifféremment sur les deux types d'environnements, nous nous concentrerons essentiellement dans le suite sur des environnements de type grille dont les cellules sont carrées et connectées à leurs quatre voisines (environnement de von Neumann, figure 3.5).

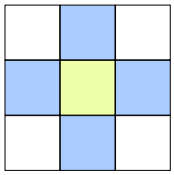
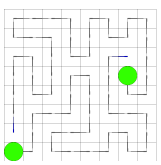


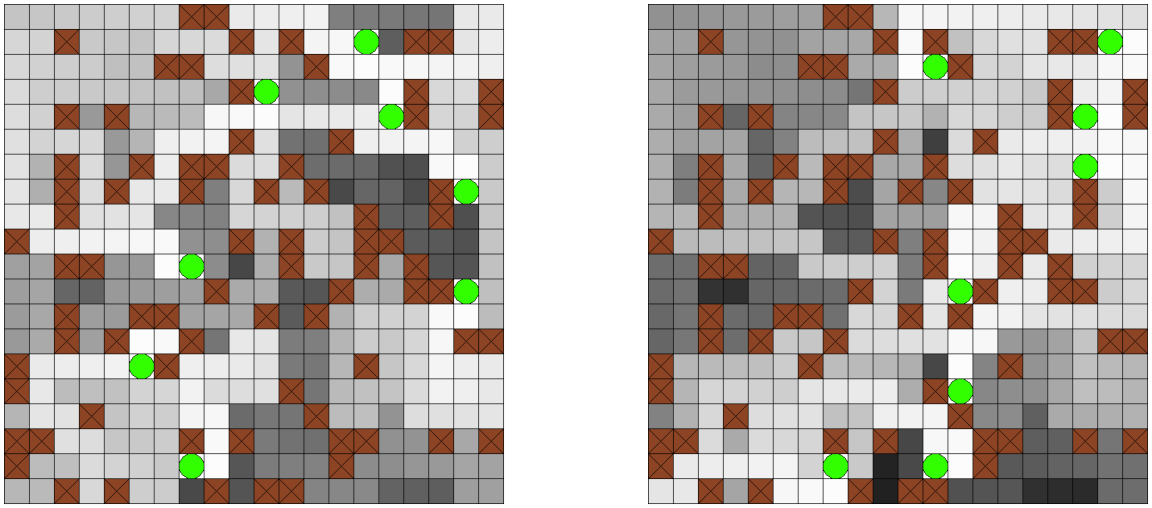
FIGURE 3.5 – Voisinage de von Neumann (en bleu) de la cellule jaune sur une grille régulière

3.1.4 Un premier aperçu du comportement d'EVAP

Lors de l'exécution d'EVAP, nous pouvons observer que l'algorithme traverse plusieurs phases distinctes. Nous présentons le comportement de l'algorithme tel que présenté dans l'algorithme 9 à l'aide de captures d'écran réalisées en simulation. Il ne s'agit que d'un rapide résumé du comportement global d'EVAP et le lecteur est invité à suivre une analyse plus complète du modèle proposée en chapitre 4.

Description d'une capture d'écran — La figure 3.6.a représente une capture d'écran réalisée lors d'une simulation de patrouille par EVAP sur l'environnement grille *densité* (cf. figure 3.4.b). Nous pouvons y observer 8 agents, représentés par des disques verts, patrouillant l'environnement en le visitant cellule par cellule. Lors de la visite d'une cellule, les agents y déposent une phéromone telle que la quantité totale dans cette cellule soit égale à Q_{max}. La couleur des cellules indique directement la quantité de phéromone présente ; allant du blanc pour une quantité $Qmax$ au noir pour une quantité de phéromone proche de 0. Les phéromones s'évaporant au cours du temps, nous pouvons observer une modification constante du champ de phéromone qui tend à s'assombrir (voir figure 3.6.b). Le comportement des agents étant défini comme une descente locale du gradient de phéromone — soit comme le déplacement vers les zones de plus grande oisiveté —, nous pouvons l'interpréter graphiquement comme une préférence des agents à se déplacer vers les zones les plus sombres. Les cellules colorées en marron et barrées d'une croix représentent des obstacles, c'est à dire des cellules inaccessibles aux agents.





a. b.

FIGURE 3.6 – Captures d'écran d'une simulation de patrouille de l'environnement *densité* par 8 agents EVAP. Plus les cellules sont claires, plus elles ont été visitées récemment. La capture b. représente le système quelques itérations après que la capture a. ait été prise.

Analyse d'une simulation de patrouille par EVAP

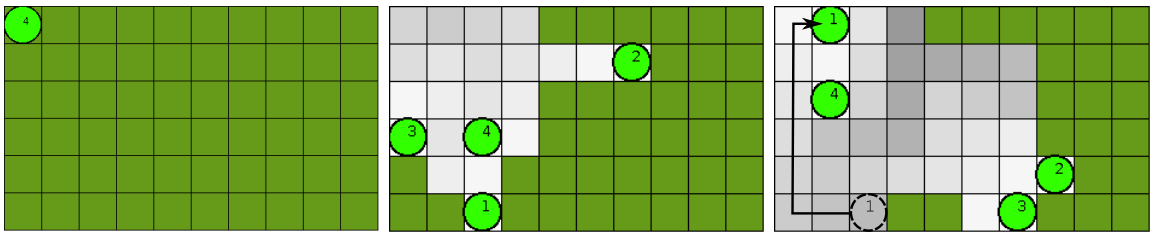
Nous présentons ici une simulation d'EVAP sur un environnement rectangulaire de 6×10 cellules patrouillé par quatre agents. L'environnement est initialement vierge de phéromones et les agents partent simultanément du coin supérieur gauche de l'environnement.

Phase 1 : exploration de l'environnement — La figure 3.7 montre les premiers pas de la simulation. Les agents débutent la patrouille sur un environnement vierge de phéromones depuis le coin supérieur gauche de l'environnement (figure 3.7.a, les cellules non explorées sont représentées en vert). En l'absence de marquage dans l'environnement, les agents choisissent leur destination aléatoirement [19]. Notons toutefois que les marques laissées derrière eux les empêchent, en quelque sorte, de revenir sur leurs pas puisqu'ils préféreront visiter les cellules voisines dont le marquage est nul. Ce comportement aléatoire est cependant générateur de comportements sous-optimaux. Sur les figures 3.7.b et 3.7.c, nous pouvons voir que l'agent 1 effectue un choix (à l'itération 8) qui l'amène un peu plus tard à revenir sur ses pas et à revisiter des cellules déjà explorées.

Phase 2 : phase de patrouille — Après une première couverture de l'environnement, les agents sont *guidés* par les phéromones qu'ils ont déposées précédemment (en formant ainsi une boucle de rétroaction ; les agents agissent sur l'environnement qui agit sur les agents en retour). La qualité de la patrouille augmente alors significativement [20] mais nous notons l'apparition de zones sombres dans le champ de phéromones, c'est à dire de cellules n'ayant

19. Nous rappelons au lecteur que les agents ne disposent ni de carte de l'environnement ni de mémoire et que leur perception est limitée aux quatre cellules adjacentes à leur position.

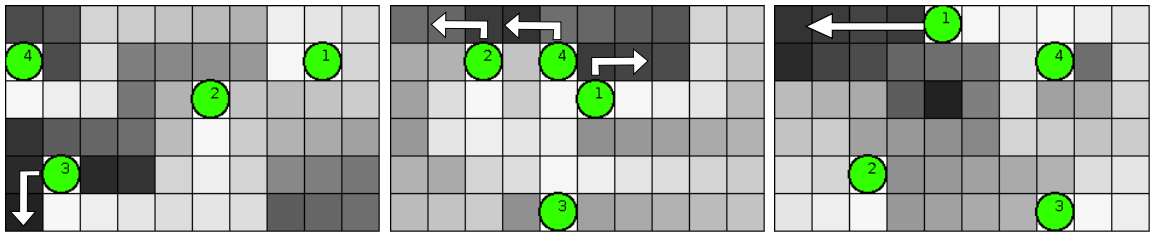
20. Cette affirmation sera étudiée en chapitre 4.



a. t=0 b. t=8 c. t=15

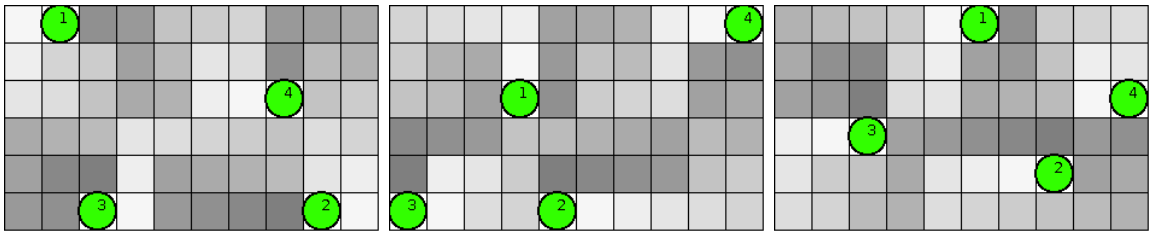
FIGURE 3.7 – Exploration d'un environnement grille de 6*10 cellules par un groupe de 4 agents

pas été visitées depuis "longtemps" [21] (voir figure 3.8). Nous pouvons cependant voir que les agents sont répartis dans l'environnement de façon relativement homogène et qu'au moins un agent se situe toujours relativement près d'une de ces cellules sombres. Lors de cette phase, la patrouille n'est pas optimale mais le nombre de cellules de forte oisiveté reste faible. De plus, comme elles sont très attractives pour les agents, dès que l'un d'entre eux passera dans le voisinage d'une de ces cellules, il la visitera.



a. t=35 b. t=75 c. t=110

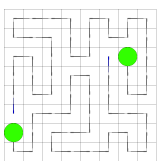
FIGURE 3.8 – Patrouille de l'environnement après une première exploration. Les zones sombres correspondent aux cellules dont le délai depuis la dernière visite est le plus important. Les flèches indiquent les directions que prendront les agents au cours des prochaines itérations.



a. b. c.

FIGURE 3.9 – Le système a convergé vers un cycle unique sur lequel sont répartis 4 agents (cf. représentation du cycle en figure 3.10).

21. A titre d'information, *pour cette simulation*, la visualisation a été calibrée pour qu'une cellule passe de *blanc* à *noir* en 30 itérations. La cellule du coin inférieur gauche de la figure 3.8.a a été visitée pour la dernière fois il y a 20 à 25 itérations environ et le délai de visite pour une patrouille optimale se situe à 15 itérations



Phase 3 : phase cyclique — EVAP présente un comportement émergent observable a long terme. En effet, nous assistons lors de la phase de patrouille à une organisation du champ de phéromone par les agents qui va mener le système à entrer dans une stratégie en cycle rendant la patrouille plus efficace, en terme d'oisiveté, que lors de la phase précédente (cf. figures 3.9 et 3.10). En l'absence de perturbations, ce cycle persistera indéfiniment. Le processus d'auto-organisation est complexe et correspond plus à un changement de phase plutôt qu'à une construction incrémentale de la solution. Nous ne nous attardons pas ici sur l'auto-organisation et étudierons plus en détails ce phénomène au chapitre 6.

Nous pouvons observer visuellement la différence de qualité entre la phase de patrouille au cours de laquelle le champ de phéromone est en permanente réorganisation (figure 3.8) et la phase cyclique (figure 3.9). La figure 3.9 ne présente, en effet, plus que des cellules relativement claires, signe d'une bonne patrouille.

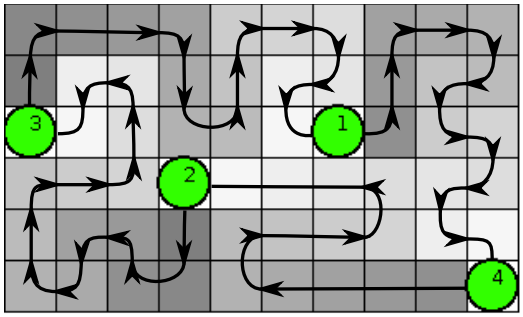


FIGURE 3.10 – Une représentation plus explicite du cycle présenté sur la figure 3.9.

La figure 3.10 explicite le cycle suivi par les agents sur la figure 3.9. Il existe de multiples types de cycles (hamiltoniens, non hamiltoniens, uniques ou partitionnant l'environnement, comme décrit au chapitre 5) mais ce cas précis présente un cycle unique partagé par les agents qui sont répartis régulièrement le long du chemin. Il ne s'agit cependant pas d'un cycle optimal puisque les interdistances entre agents ne sont pas égales (l'écart entre l'agent 1 et l'agent 4 ainsi qu'entre l'agent 4 et l'agent 2 est de 14 cellules alors que l'espace entre les agents 2 et 3 et les agents 3 et 1 est de 16 cellules). De ce fait, la variance de la fréquence de visite des cellules n'est pas nulle (mais reste faible. Il s'agit là d'un critère d'optimalité de la patrouille, voir section 2.3.2).

Une animation de cycle — Afin de mieux comprendre à quoi ressemble un cycle, nous proposons une animation tirée d'une simulation d'EVAP sous la forme d'un flipbook. Le coin inférieur droit de la thèse présente des imagettes à faire défiler (on peut aussi voir l'animation sur la version pdf en choisissant un affichage pages doubles non continu et en pressant la touche *page down*). Cette animation représente un cycle de 64 itérations patrouillé par deux agents sur un environnement de 8×8 cellules. L'animation commence en page 5.

Les trois phases de patrouille d'EVAP vues à travers des mesures d'oisiveté

La figure 3.11 présente les courbes d'oisiveté — moyenne instantanée sur le graphe (IGI) et pire instantanée sur le graphe (WGI), cf. section 2.3.1 — de la simulation présentée dans la sec-

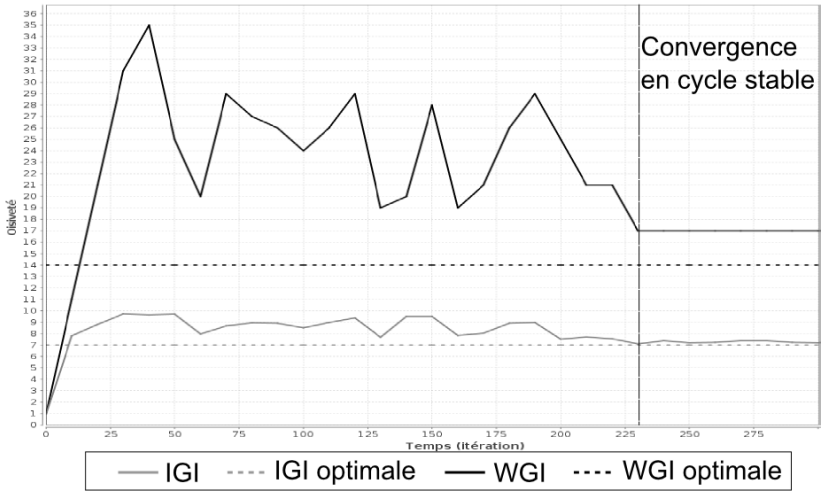


FIGURE 3.11 – Oisiveté pour 4 agents sur un environnement grille de 10x6 cellules. Les courbes se stabilisent à l'optimal après 230 itérations, reflétant la convergence vers un cycle stable.

tion précedente (un environnement de 10x6 cellules patrouillé par 4 agents, voir figures 3.7, 3.8 et 3.9).

Dans un cadre plus général, comme présenté en figure 3.12, le comportement de l'algorithme EVAP peut être découpé en trois phases distinctes :

- une phase d'exploration dans laquelle les agents effectuent une première couverture de l'environnement ;
- une phase de convergence durant laquelle les agents réorganisent le champ de phéromone, les performance s'améliorent et sont stables en moyenne ;
- une phase cyclique où les agents répètent indéfiniment le même chemin, les performances deviennent stables et s'approchent de l'oisiveté optimale et peuvent l'atteindre dans certains cas.

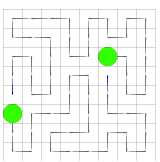
Nous étudions dans le chapitre 4 le comportement d'EVAP ainsi que ses performances dans les deux premières phases. Les chapitres 5 et 6 se focalisent sur la définition et l'étude de l'auto-organisation vers des cycles ainsi que sur les performances des solutions obtenues par ce processus.

3.2 EVAW - un EVAP sans phéromones

L'étude formelle du modèle EVAP est rendue difficile par la nature dynamique du champ de phéromone. En plus des modifications induites par la visite des nœuds par les agents, l'évaporation modifie le marquage à chaque itération.

Dans cette section nous présentons une version préliminaire de VAW [22], un algorithme proposé par Wagner [Wagner *et al.*, 1999]. Ce dernier produit une politique de patrouille

22. Cette version de VAW n'apparaît qu'en annexe de [Wagner *et al.*, 1999]. L'algorithme évoluera par la suite vers un système utilisant deux phéromones. Il s'agira basiquement de l'algorithme *node counting* augmenté d'un mécanisme de mesure de l'oisiveté pour départager deux nœuds de valeurs égales.



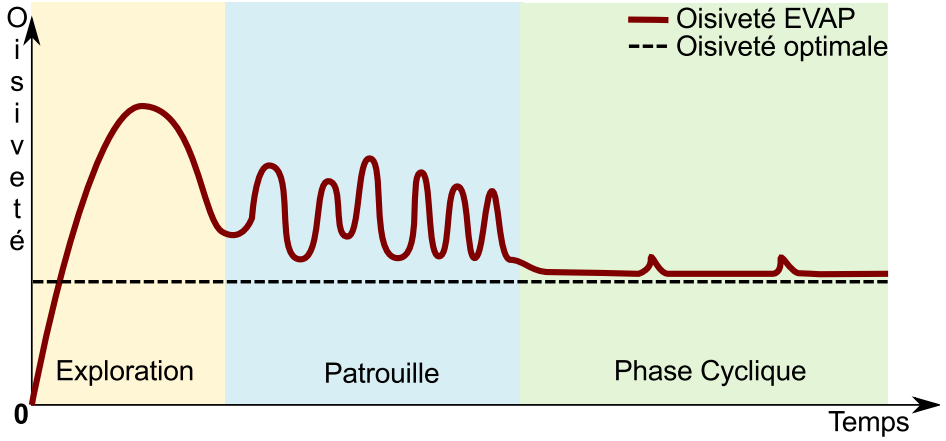


FIGURE 3.12 – Profil typique de l'oisiveté moyenne au niveau du graphe

proche de celle d'EVAP mais en utilisant un marquage statique à valeur entière — le marquage d'un nœud n'est modifié que par la visite d'un agent — facilitant les études formelles. La principale différence entre VAW et EVAP provient de l'ordre des actions des agents. Les agents VAW marquent la cellule courante puis se déplacent alors que les agents EVAP se déplacent puis marquent leur destination. Cette différence par rapport au prototype des algorithmes fourmi mis en évidence par [Koenig *et al.*, 2001] (VAW appartient à ce type d'algorithme, EVAP s'en éloigne un peu. Voir algorithme 1, section 2.4.4.) mène à des comportements locaux différents.

Nous adaptons donc EVAP pour intégrer ce marquage statique et obtenir EVAW, un algorithme strictement équivalent à EVAP sur le plan du comportement des agents et permettant de réaliser plus facilement les preuves formelles des propriétés observées expérimentalement. Nous présentons dans la sous-section suivante l'algorithme VAW et discutons de ses différences avec EVAP. Nous prouvons ensuite l'équivalence entre les deux types de marquage (marquage à base de phéromone pour EVAP et marquage statique pour VAW) et proposons EVAW. Finalement, nous discutons des conséquences relatives au choix du type de marquage.

3.2.1 Les modèles VAW et EVAW

VAW (Vertex Ant Walk) est un algorithme fourmi construit sur le même principe qu'EVAP, à savoir la mesure du temps écoulé depuis la dernière visite d'un nœud. Au lieu de déposer une quantité fixe de phéromone, les agents marquent la date courante. L'oisiveté d'un nœud est donc donnée par la différence entre la valeur du marquage et la date courante. Le comportement des agents VAW est donc lui aussi défini comme une descente de gradient (algorithme 10, lignes 3 et 4).

Nous définissons donc le modèle EVAW (algorithme 11) comme le modèle EVAP mais en y substituant le dépôt de phéromone par le marquage de la date courante. Nous définissons de cette façon un modèle qui ne nécessite plus le support d'un environnement actif capable d'effectuer le calcul de l'évaporation.

Nous pouvons remarquer que les deux algorithmes proposent des comportements très proches et ne se distinguent que par *l'ordre* des actions des agents — les agents EVAW se

Algorithme 10 : Comportement d'un agent VAW

```
1 t = t + 1;
2 m_visite(v) ← t ;                                  /* marquage du nœud courant */
3 y ← arg min_{v'∈Vois(v)} m_visite(v') ;            /* choix du nœud destination */
4 moveTo(y) ;                                        /* déplacement */
```

Algorithme 11 : Comportement d'un agent EVAW

```
1 t = t + 1;
2 y ← arg min_{v'∈Vois(v)} m_visite(v') ;            /* choix du nœud destination */
3 moveTo(y) ;                                        /* déplacement */
4 m_visite(v) ← t ;                                  /* marquage du nœud destination */
```

déplacent puis marquent quand les agents VAW marquent puis se déplacent. Ce changement mineur dans le comportement des agents induit pourtant une différence dans le comportement global du système (en considérant l'hypothèse d'une exécution séquentielle des agents) :

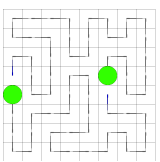
- Après avoir exécuté leur algorithme, les agents EVAW se trouvent toujours sur un des nœuds de l'environnement de plus haute valeur (c'est-à-dire sur le nœud qu'ils viennent de marquer, voir figure 3.13). Ainsi, deux agents ont peu de chances de se retrouver sur une même cellule. Dans l'hypothèse où plusieurs agents se trouvent sur le même nœud, ils se sépareront à la prochaine itération à la condition que le nœud courant possède au moins autant de voisins que d'agents présents.
- Après avoir exécuté leur algorithme, les agents VAW (et des autres algorithmes de Wagner) ne se trouvent jamais sur un des nœuds de l'environnement de plus haute valeur (ils ont marqué le nœud v qu'ils viennent de quitter et se trouvent sur le nœud voisin de v de plus faible marquage). Ainsi, si un nœud correspond au minimum du voisinage de deux agents ou plus, ceux-ci se retrouveront nécessairement au même endroit au début de l'itération suivante (voir figure 3.13). De plus, ces derniers ne se sépareront potentiellement que dans le cas ou plusieurs destinations sont éligibles, à savoir des nœuds dont les marques sont équivalentes. Dans ce cas, leur séparation reste soumise au mécanisme de décision ; dans le cas d'un choix aléatoire, deux agents ayant le choix entre deux destinations possibles n'ont qu'une probabilité de 0.5 de se séparer.

En pratique les deux modèles produisent des patrouilles de qualité comparable. Nous reviendrons plus loin sur les différences entre les modèles et les conséquences de l'usage de l'un ou l'autre des marquages.

3.2.2 Équivalence entre les marquages

Nous prouvons ici l'équivalence entre les comportements d'EVAP et d'EVAW afin de pouvoir bénéficier du formalisme simplifié du marquage statique et transposer directement à EVAP les résultats théoriques obtenus sur EVAW.

Dans le cas d'EVAP, nous avons vu en section 1.3.3 que l'évaporation définit une fonction monotone et strictement décroissante selon le temps. Nous avons également vu que la différence



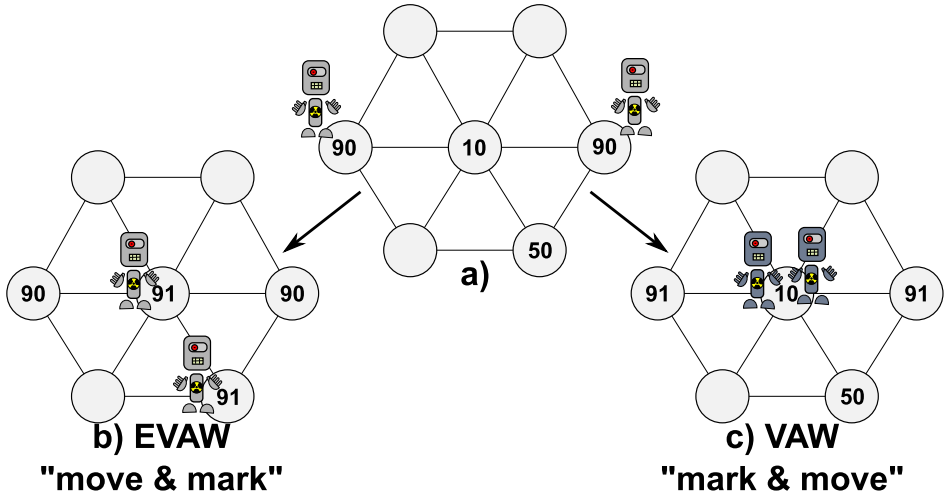


FIGURE 3.13 – Différences de comportement entre EVAW et VAW : b) les agents EVAW se déplacent puis marquent, c) les agents VAW marquent puis se déplacent.

entre la valeur d'un nœud et Q_{max} représente le nombre d'itérations écoulées depuis le dépôt de la marque (soit l'oisiveté du nœud).

Dans le cas d'EVAW, le compteur de temps est une fonction monotone strictement croissante et les agents choisissent pour destination la valeur la plus basse parmi les nœuds voisins. La différence entre la valeur d'un nœud et la date courante correspond ainsi également à son oisiveté.

Ainsi, il est possible d'exprimer pour les deux marquages $\delta t(x)$, le temps écoulé depuis la dernière visite du nœud x :

$$\begin{aligned} \delta t(x) &= \log(m_{visite}(x)/Q_{max})/\log(\rho) \quad \text{pour EVAP,} \\ \delta t(x) &= t - m_{date}(x) \quad \text{pour EVAW.} \end{aligned}$$

Nous pouvons établir sans ambiguïté une bijection entre la fonction d'évaporation d'EVAP et de marquage du temps d'EVAW (à chaque paire *date courante, valeur de marquage* nous pouvons associer une unique valeur de phéromone fonction de Q_{max} et d'un taux d'évaporation donné).

Les comportements d'EVAP et d'EVAW sont donc équivalents (l'évolution des deux algorithmes, à partir d'un même état intial, aboutira au même état final). Toute propriété prouvée sur EVAW sera donc directement transposable à EVAP (et inversement).

3.2.3 Marquage statique contre marquage dynamique

Nous venons de montrer que les deux types de marquage produisent des comportements équivalents et donc que les algorithmes EVAP et EVAW exhibent les **mêmes** comportements. Cependant, ces deux types de marquages posent des contraintes d'implémentation différentes.

- Dans le cas d'EVAW (marquage du temps), les agents ont besoin de posséder des horloges internes synchronisées entre elles. En plus d'avoir la même fréquence, celles-ci doivent avoir la même origine (t=0). Si l'on veut pouvoir ajouter des agents en cours d'exécution (démarrage décalé, retour d'un agent après un changement de batterie ou une panne...), il est nécessaire de s'assurer de la synchronisation des horloges. Un agent dont l'horloge est décalée effectuera correctement sa tâche mais perturbera le fonctionnement des autres agents :
  - pour $t_{agent} < t_{courant}$: les marques laissées par cet agent semblent anciennes et *poussent* les autres agents à visiter des cellules dont l'oisiveté effective est faible.
  - pour $t_{agent} > t_{courant}$: les marques laissées par cet agent *empêchent* les autres agents de visiter ces cellules tant que $t_{courant}$ n'a pas dépassé la valeur de ces marques. Les cellules visitées par cet agent sembleront pour les autres agents avoir été visitées récemment alors que leur oisiveté effective est importante.

  Cette synchronisation peut être difficile à réaliser dans une implémentation robotique décentralisée.
- Dans le cas d'EVAP, l'algorithme ne requiert pas une telle synchronisation et permet donc d'ajouter des agents en cours d'exécution sans se soucier d'une quelconque synchronisation des horloges. Cependant celui-ci nécessite un environnement actif capable de gérer le processus d'évaporation des phéromones. Il s'agit d'une opération simple devant être réalisée sur l'ensemble des nœuds de l'environnement de façon identique, notamment au niveau de la fréquence d'évaporation. Dans la nature ces questions ne se posent pas puisque l'évaporation des — vraies — phéromones repose sur des processus chimiques. Dans les systèmes artificiels, une telle approche requiert une structure capable de soutenir de tels calculs. Nous présentons à ce titre, en chapitre 7, l'architecture I-Tiles [Pépin *et al.*, 2009] que nous utilisons pour une implémentation robotique d'EVAP.

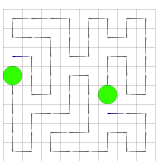
3.3 Quelques propriétés du modèle EVAP

Nous terminons ce chapitre en présentant deux propriétés du modèle EVAP assurant que celui-ci est capable de réaliser la tâche de patrouille, c'est-à-dire

1. de couvrir l'environnement en temps fini pour
2. revisiter régulièrement l'ensemble des nœuds de l'environnement.

3.3.1 Borne sur le temps de couverture

Nous reprenons ici le calcul d'une borne de couverture (ce qui prouve la couverture de l'environnement en temps fini) issu des travaux de Wagner sur la version préliminaire de VAW dans [Wagner *et al.*, 1999], et directement adaptable au modèle EVAP. S'il s'agit seulement d'une borne théorique très éloignée des performances que nous pouvons observer expérimentalement. Ce résultat est crucial pour le travail mené au chapitre 6 sur la convergence vers des cycles.



Théorème 3.3.1 *Suivant le comportement décrit par l'algorithme 9, un groupe de k agents EVAP couvre l'ensemble des nœuds d'un graphe G en un temps T_{vis} tel que :*

$$T_{vis} \leq \frac{n\Delta^d}{k}$$

avec n le nombre de nœuds, d, le degré maximum des nœuds du graphe, et Δ le diamètre du graphe[23] *[Wagner* et al.*, 1999].*

Notons qu'il ne s'agit là que d'un majorant (très large) pour le pire cas et qu'en pratique (voir chapitre 4), le temps nécessaire à la couverture de l'environnement est bien meilleur.

A titre d'exemple, pour un environnement grille de 10x10 cellules patrouillé par 10 agents, la borne de couverture que nous avons calculée donne un temps de $\frac{100\times18^4}{10}$, soit 1.049.760 itérations. Lors des expérimentations que nous avons menées, le temps nécessaire à la couverture, pour ces paramètres, n'a *jamais* excédé les 50 itérations.

Le paragraphe suivant illustre le comportement adopté par EVAP dans ce *pire cas* de couverture.

Illustration du pire cas de couverture

Le problème des comportements extrêmement sous-optimaux dans les algorithmes fourmi — tels que celui du pire temps de couverture que nous avons présenté précédemment — est intrinsèque aux limitations des capacités perceptives et mémorielles des agents. La faible profondeur de recherche (usuellement unitaire en ce qui concerne les approches discrètes) ne leur permet de réaliser efficacement la planification de leurs actions qu'à un horizon très court (typiquement, une seule itération).

Ainsi, la décision optimale, pour un agent, à un instant t risque de le mener à $t+1$ vers une situation peu avantageuse qu'il aurait peut être été possible d'éviter en faisant un autre choix moins avantageux immédiatement mais plus intéressant à moyen terme.

La figure 3.14 présente ainsi le pire cas envisageable pour un agent EVAP [Wagner *et al.*, 1999]. Nous faisons l'hypothèse que l'agent se trouve initialement sur le nœud le plus à gauche et que l'environnement est vierge de toute phéromone. Ainsi, quand l'agent arrive sur un nouveau nœud, il effectue un choix aléatoire pour décider de sa prochaine destination.

Lorsque l'agent se trouve dans une situation équivalente à celle présentée sur la figure 3.14.A — soit sur le nœud le plus à droite du motif triangulaire —, celui-ci a le choix entre deux destinations équivalentes en termes de marquage. Ainsi, l'agent peut effectuer un "mauvais" choix (figure 3.14.B) et sera forcé de retourner à son point de départ avant de pouvoir retourner au même endroit et continuer son exploration. Ce comportement se reproduira à chaque fois fois qu'un tel choix se présentera (figure 3.14.C). De plus, le marquage crée un gradient qui mènera l'agent à reparcourir les nœuds déja visités et ceci à chaque nouvelle visite du nœud.

23. Le diamètre d'un graphe correspond au plus long chemin séparant deux nœuds en excluant boucles, revisites et retours en arrière. Autrement dit, il s'agit du *plus court chemin entre les deux nœuds les plus éloignés* entre deux nœuds.

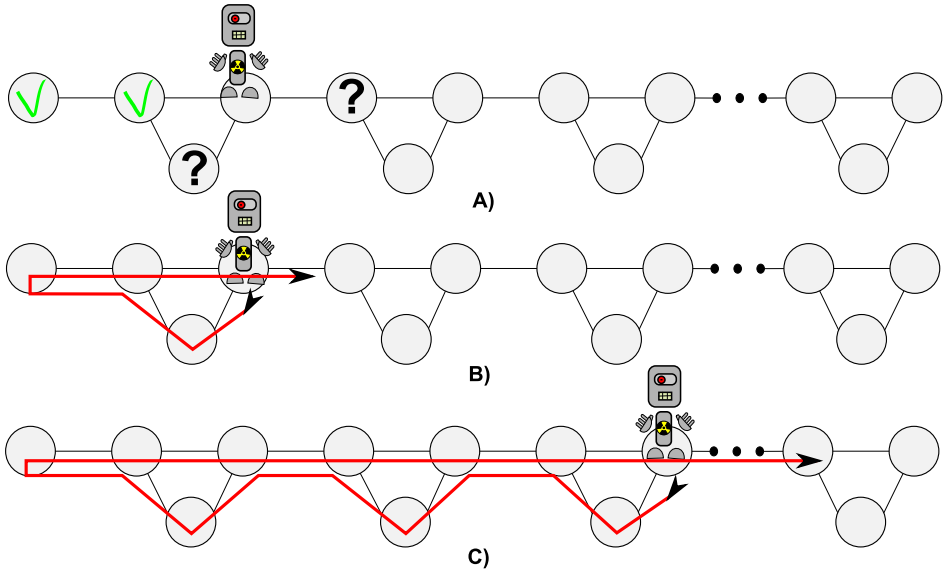


FIGURE 3.14 – Illustration des conséquences de la perception locale pour un agent EVAP

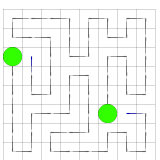
Le seul moyen d'éviter ce type de comportement serait de disposer d'une carte de l'environnement et de planifier les déplacements des agents avec une profondeur de recherche suffisante, ce qui nous ferait perdre tous les avantages de l'approche fourmi. Il faut cependant relativiser cette limitation d'EVAP. Tout d'abord, ce comportement est obtenu sur un environnement très contraint, difficilement rencontrable dans la réalité. En second lieu, le choix de la *mauvaise* destination est stochastique. La probabilité d'atteindre le *pire cas* est donc relativement faible. Pour finir, l'utilisation simultanée de plusieurs agents coopérant de manière indirecte rend ce problème moins pénalisant puisque si deux agents visitent un nœud nécessitant un tel choix et que l'un des deux réalise le 'mauvais' choix, l'autre continuera l'exploration de l'environnement. A cause du marquage, deux agents passant par la même cellule dans un court délai choisiront nécessairement des destinations différentes. Notons également qu'une fois la première couverture réalisée, les agents disposent — *via* le champ de phéromone — d'informations sur leur environnement leur permettant de ne pas retomber dans ce type piège.

3.3.2 Propriété de patrouille

Nous posons ici la preuve de patrouille, soit le fait que chaque nœud de l'environnement sera visité une infinité de fois et qu'il ne se formera donc pas d'îlots de cellules non visitées. Il s'agit d'un propriété fondamentale de l'algorithme puisqu'elle garantie que la patrouille est effectivement réalisée et qu'elle ne s'arrête pas, quelle que soit la topologie de l'environnement et le nombre d'agents présents.

Théorème 3.3.2 (Preuve de patrouille) *Chaque nœud d'un environnement complètement connecté est visité infiniment souvent par les agents EVAP.*

Preuve: Soit un groupe d'agents patrouillant l'environnement $G(V, E)$ représenté sous la



forme d'un graphe entièrement connecté [24]. Nous faisons l'hypothèse qu'à partir d'une certaine date, un sous-ensemble N de nœuds $\in V$ ne sera plus visité par les agents, formant ainsi un îlot, et que le reste des nœuds $V \setminus N$ continuera à être patrouillé normalement. D'après la définition du modèle EVAP, la valeur du marquage d'un nœud v décroît dans le temps à cause de l'évaporation. Dès lors, après la couverture de la zone patrouillée, les valeurs des nœuds de N seront nécessairement plus faibles que celles de n'importe quel autre nœud ; c'est-à-dire qu'après un délai suffisant, on aura pour tout nœud $v_i \in N$ et pour tout nœud $v_j \in V \setminus N$: $m(v_i) < m(v_j)$.

Quand un agent visite un nœud de la frontière de la zone patrouillée, il percevra donc un nœud $\in N$ contenant la valeur minimale parmi ses voisins et, d'après le comportement des agents décrit dans l'algorithme 7, ira le visiter.

Il y a contradiction avec les hypothèses initiales. Nous pouvons donc conclure qu'EVAP patrouille l'ensemble des nœuds de l'environnement et que la formation durable d'ilôts de cellules non (re)visitées est impossible.

24. Pour tout couple de nœuds (u,v), il existe un chemin connectant u et v.

Chapitre 4

Evaluation des performances d'EVAP pour la patrouille multi-agent

Sommaire

4.1 CLInG, un algorithme fondé sur la propagation d'information 57
4.1.1 Présentation 57
4.1.2 Comportement et hypothèses sur l'environnement 58
4.2 Hypothèses de simulation et détails techniques 59
4.2.1 Hypothèses de simulation 59
4.2.2 Mesure de performances 60
4.3 Etude de la couverture 61
4.3.1 Observations expérimentales 61
4.3.2 Etude du comportement exploratoire des agents 63
4.3.3 Les algorithmes fourmi et la couverture 66
4.4 Etude de la patrouille (phase d'organisation) 67
4.4.1 Comparaison des patrouilles d'EVAP, de LRTA* et de CLInG . 68
4.4.2 Impact du nombre d'agents sur la patrouille 71
4.4.3 Conclusion sur la patrouille par les algorithmes fourmis 72
4.5 Etude de la robustesse d'EVAP 73
4.5.1 Robustesse à la perte d'agents 73
4.5.2 Passage à l'échelle 75
4.6 Conclusion sur la patrouille d'EVAP 78

Ce chapitre se concentre sur l'évaluation de la patrouille d'EVAP. Pour cela, nous comparons les performances d'EVAP à deux autres approches fourmi pour la patrouille.

La première approche retenue est LRTA*. Cet algorithme, présenté au chapitre 2, est très connu puisqu'il s'agit de l'adaptation (à la couverture et à la patrouille multi-agent) d'un algorithme de recherche de chemin vers un but. Il a été reconnu pour son efficacité pour la

couverture face à d'autres algorithmes fourmi [Koenig *et al.*, 2001]. La seconde approche, moins connue, se nomme CLInG [Sempé, 2004]. Le comportement des agents CLInG est défini comme une remontée du gradient d'oisiveté (les agents ont un comportement analogue à EVAP). L'originalité de CLInG repose sur un environnement intelligent capable de propager l'oisiveté à travers l'environnement. Les cellules de forte oisiveté *appellent* ainsi les agents, ce qui permet de compenser la simplicité de leur comportement.

Nous avons vu dans le chapitre précédent que la patrouille réalisée par EVAP peut-être découpée en trois phases distinctes, comme présenté sur la figure 4.1 :

- Une première phase de couverture lors de laquelle les agents explorent un environnement initialement vierge de phéromones. Dans cette phase l'oisiveté moyenne (IGI) augmente avant de redescendre, formant ainsi une *bosse* dont le profil dépend directement de la topologie.
- Une deuxième phase lors de laquelle les agents patrouillent l'environnement en réorganisant constamment le champ de phéromones et où l'oisiveté du système oscille avec une amplitude limitée autour d'une valeur moyenne.
- Une dernière phase, survenant à un horizon temporel lointain et qui persiste jusqu'à la fin de la simulation, lors de laquelle le système converge durablement vers un comportement cyclique stable en termes d'oisiveté.

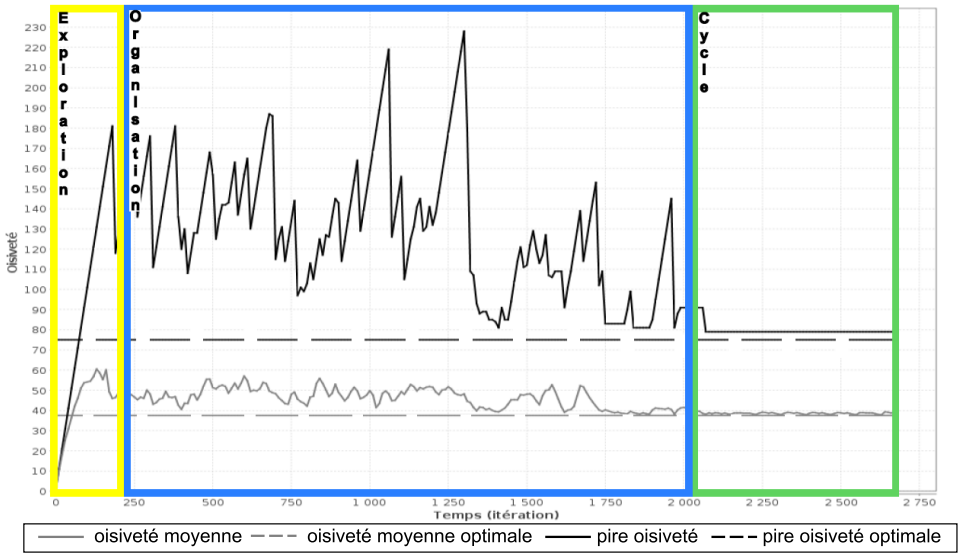


FIGURE 4.1 – Courbes d'oisiveté pour un environnement 12x12 cellules patrouillé par deux agents. On peut distinguer trois phases différentes : la phase d'exploration, la phase d'organisation et la phase cyclique.

Nous laissons pour le moment de côté l'étude de la troisième phase (la phase cyclique), à laquelle sont consacrés les chapitres 5 et 6, pour nous intéresser aux deux autres phases.

Nous allons évaluer EVAP en étudiant le comportement des trois algorithmes sur les phases de couverture et d'organisation. L'étude de la patrouille sera donc organisée en deux parties.

La première partie (section 4.3) concerne la couverture (c'est-à-dire la phase d'exploration). Nous nous intéressons en particulier à expliquer les fortes variations de performance entre les

approches considérées (en particulier, les performances médiocres d'EVAP sur ce terrain) ainsi que l'influence de la topologie de l'environnement sur le temps de couverture.

La seconde partie (section 4.4) concerne la patrouille proprement dite dans laquelle nous pouvons observer une évolution notable de la qualité du comportement des approches considérées par rapport à la couverture.

Nous consacrons la section 4.5 à l'étude de la robustesse d'EVAP. La capacité d'un système à résister aux perturbations est un point très important dans le cadre d'une implémentation robotique.

Nous commençons tout d'abord par présenter l'algorithme fourmi CLInG et ses particularités (section 4.1) puis nous définissons les hypothèses d'exécution utilisées lors des expérimentations (section 4.2).

4.1 CLInG, un algorithme fondé sur la propagation d'information

CLInG [Sempé, 2004] est un algorithme très proche d'EVAP puisque les agents suivent un comportement analogue de remontée du gradient d'oisiveté. Cette approche se distingue toutefois d'EVAP par la propagation de l'information d'oisiveté par les cellules elles mêmes. Ainsi, lorsqu'une cellule atteint une forte oisiveté, elle se met à *appeler* les agents en diffusant un signal. Ce procédé permet donc en principe de réduire l'impact de la vision locale des agents.

4.1.1 Présentation

CLInG utilise des agents réactifs et fait l'hypothèse d'un environnement capable de calculer deux informations :

l'oisiveté des nœuds du graphe représentant l'environnement,

l'oisiveté propagée depuis les nœuds de fortes oisivetés vers les autres nœuds du graphe.

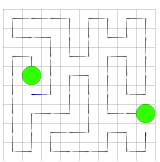
L'originalité de cet algorithme est donc d'introduire, en plus de l'oisiveté des nœuds, une oisiveté propagée créant un second gradient décroissant partant des cellules de forte oisiveté. Les agents remontent ce gradient qui les guide vers les cellules d'interêt et, lorsqu'un agent visite un nœud, il remet son oisiveté à 0.

Chaque nœud v portera donc, en plus de son oisiveté propre O_v, une oisiveté propagée OP_v. L'oisiveté propagée pour un nœud dépend de l'oisiveté propagée de ses voisins et de son oisiveté propre :

$$OP_i = max(O_i, max(f(i,j))), j \in Vois(i), \tag{4.1}$$

avec j les nœuds voisins de i, et f la fonction de propagation telle que décrite ci-dessous :

$$\begin{cases} si \mathrm{OP}_j - \alpha - \beta.I(j) \geq OP_{min}, \\ \mathrm{f(i,j)} = \mathrm{OP}_j - \alpha - \beta.I(j), \quad \text{sinon, si } OP_j < OP_{min}, \\ \mathrm{f(i,j)} = \mathrm{OP}_{min}, \quad \text{sinon, } f(i,j) = OP_j - 1. \end{cases}$$



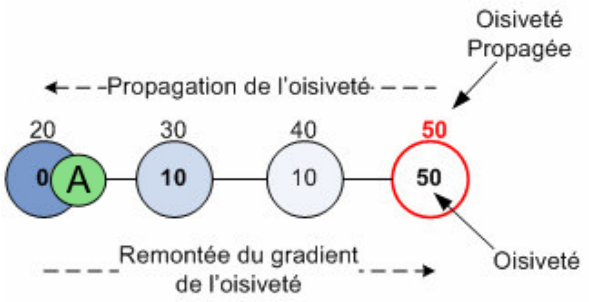


FIGURE 4.2 – Schéma représentant le fonctionnement de CLInG

Cette fonction de propagation est paramétrée par les coefficients α, β et OP_{min} de la fonction d'interception $I(j)$ présentés ci dessous :

- α est un cœfficient de propagation à mettre en relation avec le taux de diffusion des phéromones digitales. Ainsi, avec le paramètre $\alpha = 1$, l'oisiveté propagée décroîtra d'une unité par cellule ; ceci permet d'obtenir un gradient linéaire ne se propageant pas à travers tout l'environnement (l'oisiveté appartenant à $\mathbb{R}^+$, si celle-ci venait à être négative, elle serait considérée comme nulle). Augmenter α permet donc de diminuer le rayon de propagation l'oisiveté.
- $I(j)$ est la fonction d'interception valant 1 si un agent est présent sur le nœud et 0 sinon. Cette fonction permet de stopper la propagation du signal lorsqu'il rencontre un agent et limite également le regroupement d'agents provenant d'une même zone.
- β est le cœfficient d'interception qui permet de définir la "quantité" d'oisiveté interceptée par un agent.
- OP_{min} est un seuil empêchant l'oisiveté propagée de descendre sous une valeur fixée.

Les valeurs des paramètres varient en fonction des environnements et de la tâche à accomplir. Nous préciserons les valeurs de paramètres utilisées dans les expérimentations.

4.1.2 Comportement et hypothèses sur l'environnement

Le comportement des agents est analogue à celui d'EVAP puisque les agents CLInG remontent simplement le gradient d'oisiveté propagée [25].

CLInG se différencie des autres algorithmes fourmi pour la patrouille par un mécanisme de propagation de l'oisiveté. Ce mécanisme permet de compenser la vision locale des agents puisqu'ils perçoivent une donnée pouvant provenir de nœuds éloignés. Cette diffusion permet de transformer une donnée objective, l'oisiveté du nœud, en une donnée subjective, l'oisiveté propagée perçue par un agent dépendant de la distance au nœud émetteur et des agents présents sur le chemin qui mène vers ce nœud. Ce modèle réalise de cette façon une organisation des agents en fonction de la distribution de l'oisiveté dans l'environnement, les nœuds de plus forte oisiveté appelant les agents proches pour être visités. Plus l'oisiveté d'un nœud est forte

25. On notera d'ailleurs que si l'on empêche la propagation de l'oisiveté pour CLInG (en choisissant des valeurs adéquates pour les paramètres α et OP_{min}), les deux modèles produisent des patrouilles équivalentes. En effet, le comportement des agents EVAP correspond à une remontée du gradient d'oisiveté.

plus son oisiveté se propagera loin. Ainsi, un nœud de faible oisiveté ne pourra pas appeler les agents éloignés.

Le paramétrage des coefficients de propagation influe cependant fortement sur le comportement du modèle. Si l'oisiveté se propage trop loin, la cellule de plus forte oisiveté pourra par exemple attirer vers elle un grand nombre d'agents et les détourner des autres cellules ayant besoin d'être visitées. Il est donc nécessaire d'effectuer une recherche expérimentale des paramètres (il n'existe pas actuellement de méthode analytique) à chaque changement d'environnement et en fonction du nombre d'agents présents au risque d'assister à une forte augmentation de l'oisiveté moyenne du système.

Un autre handicap de CLInG provient des hypothèses fortes faites sur l'environnement. En effet, celui-ci doit être capable de réaliser le calcul de propagation de l'oisiveté, ce qui implique :

1. que la patrouille ne peut être effectuée qu'en environnement connu puisque les nœuds inexplorés doivent diffuser leur oisiveté, et
2. que le coût de calcul pour mettre l'environnement à jour est important face à un marquage statique (Node counting, VAW, LRTA*, EVAW) ou une simple évaporation (EVAP). Pour un environnement de taille n, il faudra n opérations d'incrémentation de l'oisiveté plus n opérations de propagation.

4.2 Hypothèses de simulation et détails techniques

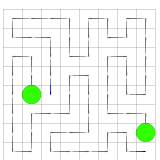
Nous précisons ici comment sont réalisées les simulations présentées dans le reste du chapitre ainsi que le calcul des bornes d'optimalité que nous utilisons pour évaluer la qualité de la patrouille produite par EVAP.

4.2.1 Hypothèses de simulation

Les expérimentations présentées dans ce chapitre sont exécutées avec un ordonnanceur séquentiel cyclique où les agents sont, à chaque itération, activés les uns après les autres dans l'ordre lexicographique (agent 1 puis agent 2 puis ... puis agent n et toujours dans le même ordre). L'environnement diffuse/évapore ensuite les phéromone si l'algorithme nécessite un tel processus. L'itération suivante se déroule de la même manière.

A l'initialisation, l'environnement est vierge de toute phéromone (chaque cellule contient une quantité nulle de phéromones), chaque nœud possède une oisiveté nulle et tous les agents démarrent simultanément (c'est-à-dire au cours de la même itération) *à partir du même nœud*.

Les expérimentations sont réalisées avec un nombre variable d'agents. L'utilisation d'un faible nombre d'agents permet de limiter le nombre d'interactions entre agents, ce qui facilite l'explication du comportement du modèle. L'utilisation d'un grand nombre d'agents, jusqu'à plusieurs centaines, permet d'évaluer l'intérêt de l'approche dans un cadre swarm.



Méthodologie et environnements considérés

Pour obtenir des chiffres statistiquement représentatifs et gommer les cas extrêmes, chaque expérience est répétée 100 fois. Sauf indication contraire, les données présentées sont les moyennes sur ces 100 expériences.

Les expérimentations sont effectuées sur les environnements présentés sur la figure 4.3 avec un nombre d'agents croissant.

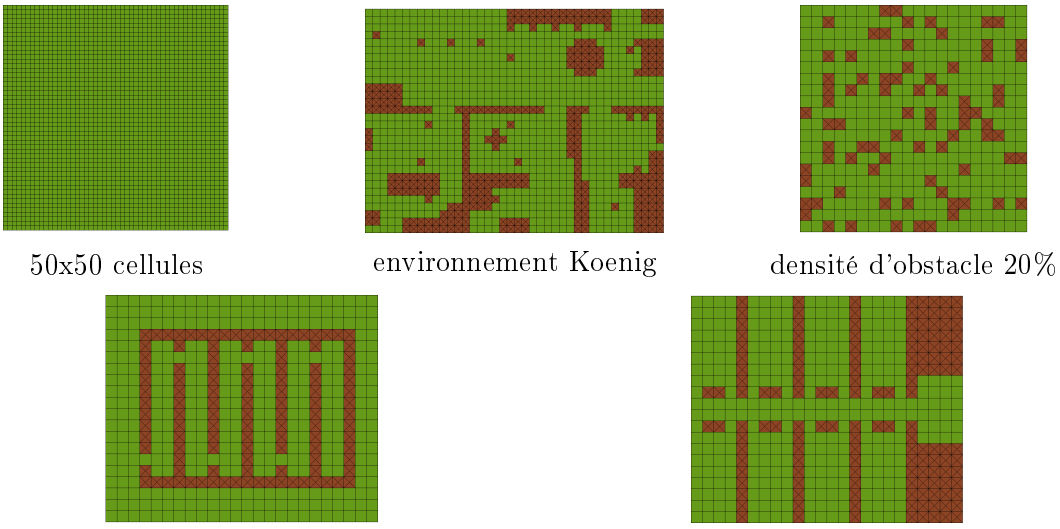


FIGURE 4.3 – liste des environnements utilisés : 50x50 (2500 nœuds), Koenig (883 nœuds), densité 20% (321 nœuds), pièces imbriquées (364 nœuds) et n-pièces (336 nœuds)

Dans un second temps, nous utilisons les mêmes environnements pour comparer l'oisiveté moyenne obtenue par ces différents modèles lors de la phase de patrouille.

4.2.2 Mesure de performances

Afin de mesurer la qualité de la patrouille effectuée par les agents, nous utilisons des mesures de performance reposant sur l'oisiveté, comme présenté en section 2.3.1. Nous rappelons au lecteur que l'oisiveté d'une cellule correspond au délai depuis sa dernière visite par un agent.

En particulier, nous utilisons :

l'IGI – Instantaneous Graph Idleness : l'IGI correspond à la moyenne des INI de l'ensemble des nœuds du graphe à un instant donné ;

la WGI – Worst Graph Idleness : la WGI correspond à l'INI la plus importante de l'ensemble des nœuds du graphe à un instant donné et représente donc l'oisiveté de la cellule la plus anciennement visitée.

Borne d'optimalité — Nous avons également besoin, pour évaluer la patrouille, d'un point de référence. Pour cela nous utilisons le minorant théorique présenté en section 2.3.1.

Nous insistons bien sur le fait qu'il ne s'agit que d'un minorant sur l'oisiveté optimale et qu'il ne peut pas être systématiquement atteint (pour plus de précisions, voir section 2.3.1).

4.3 Etude de la couverture

Nous comparons dans cette section les couvertures réalisées par EVAP, LRTA* et CLInG.

Il s'agit d'une tâche compliquée puisqu'en l'absence d'informations sur l'environnement (pas de carte, rayon de perception limité, pas encore d'information de gradient pour guider les agents vers les zones à explorer), les agents ne sont pas capables de l'explorer de façon optimale. Nous proposons d'observer le comportement de ces trois algorithmes fourmis au travers d'expériences sur divers environnements puis d'expliquer les différences de comportements qu'ils présentent. Nous mettons en avant l'intérêt de la propagation d'information de CLInG pour compenser la vision locale des agents et le rôle de la sémantique du marquage (à travers la comparaison d'EVAP et de LRTA*).

4.3.1 Observations expérimentales

La figure 4.4 présente les temps de couverture moyens pour chacun des modèles et pour chaque environnement considéré (cf. figure 4.3). En comparant les résultats d'EVAP et de CLInG, nous pouvons observer qu'en moyenne, sur l'ensemble des environnements présentés, CLInG couvre l'environnement 2,5 fois plus rapidement qu'EVAP. Dans le cas de l'environnement "pièces imbriquées", EVAP met 3,7 fois plus de temps que CLInG pour réaliser la couverture. L'utilisation de la propagation des fortes oisivetés de CLInG semble donc être particulièrement adaptée à la *tâche de couverture* et permet à CLInG de couvrir l'environnement significativement plus vite qu'EVAP (au prix d'hypothèses fortes sur l'environnement).

Les temps de couverture moyens de LRTA* se situent entre ceux d'EVAP et de CLInG (en étant d'ailleurs plus proches de CLInG que d'EVAP). Nous voyons ici l'effet de la *sémantique du marquage* sur le comportement des algorithmes puisque malgré la vision locale des agents, LRTA* explore l'ensemble des environnements environ deux fois plus vite qu'EVAP.

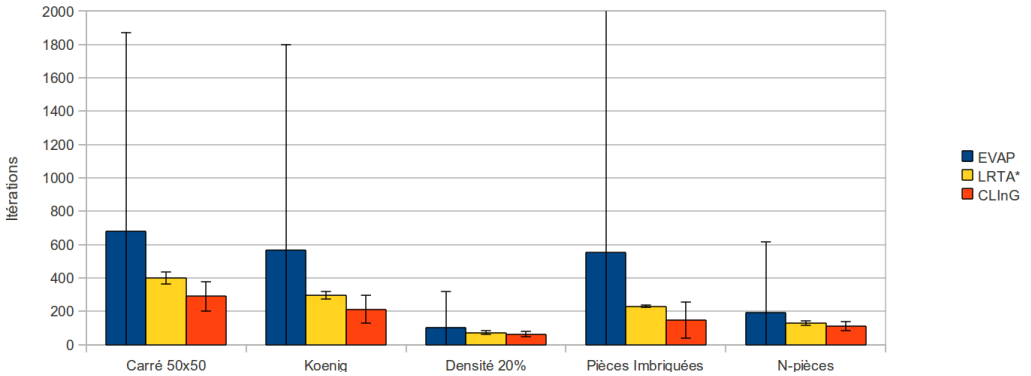
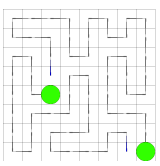


FIGURE 4.4 – Temps de couverture moyens comparés pour EVAP, LRTA* et CLInG avec 20 agents sur les 5 environnements proposés.



Penchons nous cependant sur les figures 4.5 et 4.6 qui représentent le nombre de cellules couvertes en fonction du temps[26] pour les environnements respectivement le plus favorable (augmentation de 65% du temps de couverture par rapport à CLInG) et défavorable (augmentation de 275% du temps nécessaire à la couverture par rapport à CLInG) pour EVAP.

Nous pouvons observer sur la figure 4.5 que CLInG et LRTA* atteignent les 80% de cellules visitées après 40 itérations alors qu'EVAP atteint cette même proportion après 54 itérations, ce qui revient à une augmentation de 35% du temps nécessaire pour atteindre cette proportion de cellules couvertes.

Nous pouvons également observer une vague inflexion des courbes après que 80% des cellules aient été couvertes. Si cette inflexion est assez forte pour EVAP et LRTA*, elle existe également pour CLInG (autour de 50 itérations environ) et dénote la difficulté pour les agents de trouver les dernières cellules inexplorées. Pour EVAP et LRTA* (dans une moindre mesure) ce phénomène est la conséquence directe de la vision locale des agents. LRTA* est moins handicapé puisque le marquage de l'environnement à tendance à conduire les agents vers les cellules inexplorées. Dans le cas de CLInG, cela peut être dû au paramètrage de la diffusion qui limite la distance depuis laquelle les cellules peuvent "appeler" les agents.

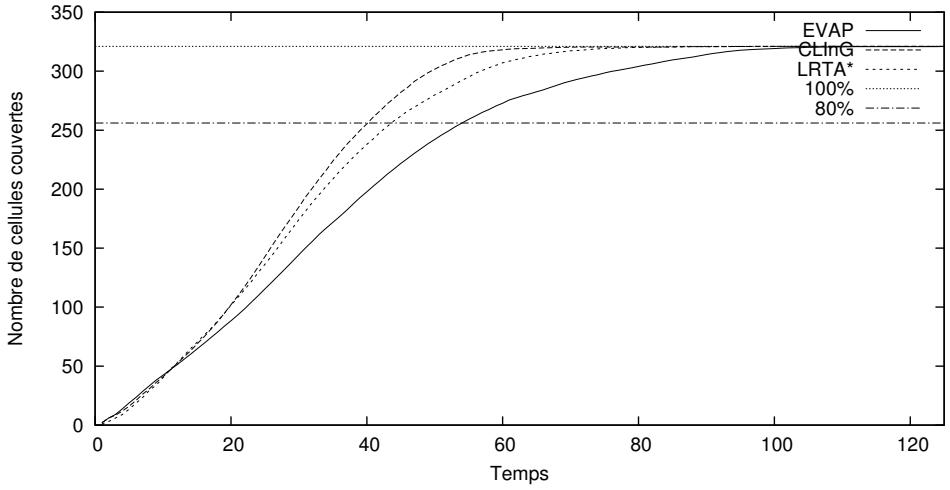


FIGURE 4.5 – Proportion de l'environnement "densité 20%" couvert par EVAP, LRTA* et CLInG en fonction du temps.

La figure 4.6 présente la même expérience réalisée sur l'environnement “pièces imbriquées”. Le comportement des modèles est similaire à celui observé lors de l'expérience précédente. L'avantage de CLInG augmente avec le pourcentage de cellules couvertes grâce à la propagation de l'information d'oisiveté. De la même manière, l'estimation de la distance à la cellule inexplorée la plus proche permet à LRTA* d'entrer assez facilement dans les pièces imbriquées.

Nous remarquons cependant une très forte inflexion des trois courbes après la couverture d'environ 220 cellules, ce qui correspond à la zone à “l'extérieur” des pièces imbriquées (moins quelques cellules). Cette inflexion est due au faible nombre d'agents entrant simultanément dans les pièces imbriquées (en plus de la difficulté à trouver l'entrée). Pour LRTA*, lorsqu'un

26. moyenne sur 100 répétitions de l'expérience pour 20 agents avec un départ simultané de tous les agents dans le coin supérieur gauche de l'environnement

agent entre dans la première pièce, la valeur de la cellule de l'entrée augmente, ce qui la rend moins attractive pour les autres agents. Enfin, pour EVAP, c'est la phéromone déposée par l'agent entrant qui bloque l'entrée aux autres agents. Pour CLInG, cela est principalement dû à la topologie de l'environnement et au *coefficient d'interception* β. Ce coefficient, décrit en section 4.1, empêche l'oisiveté de se propager au delà d'une cellule occupée par un agent. Les agents sont donc attirés par les pièces imbriquées mais lorsqu'un agent entre, il *coupe* la propagation de l'oisiveté. Les pièces imbriquées deviennent moins attractives pour les autres agents et ne seront visitées que par un ou deux agents.

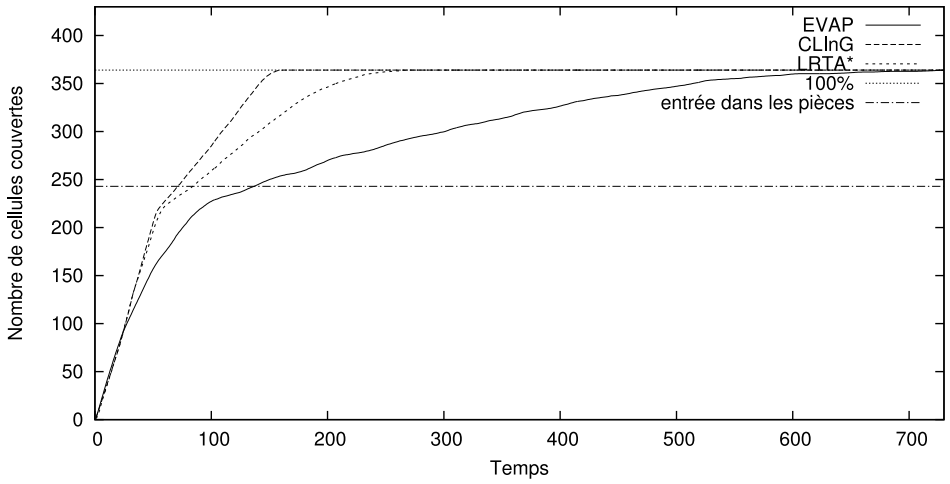


FIGURE 4.6 – Proportion de l'environnement "pièces imbriquées" couvert par EVAP, LRTA* et CLInG en fonction du temps.

Les trois approches étudiées sont capables de réaliser la couverture de l'environnement. CLInG et LRTA* se révèlent cependant beaucoup plus performantes qu'EVAP qui est moins intéressant pour cette tâche.

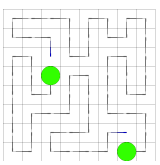
4.3.2 Etude du comportement exploratoire des agents

L'étude du comportement des agents permet d'expliquer à la fois les différences de performance entre ces trois algorithmes et pourquoi il est plus difficile de couvrir certains environnements (les temps de couverture pour chaque algorithme sont très proches pour les environnements *pièces imbriquées* et *Koenig* alors que les *pièces imbriquées* comptent deux fois moins de nœuds).

Sur des environnements “simples”

Nous cherchons ici à comprendre les différences de performance entre les algorithmes. Nous considérons le cas d'environnements “simples” (des grilles sans ou avec peu d'obstacles) pour ne pas ajouter le facteur environnemental à cette étude.

Pour les agents des trois algorithmes, en l'absence de marquage, rien ne différencie une cellule d'une autre et ils choisissent leurs destinations au hasard. Pourtant, certains choix



sont plus intelligents que d'autres. La figure 4.7 présente une situation initiale défavorable que nous allons considérer pour les trois algorithmes.

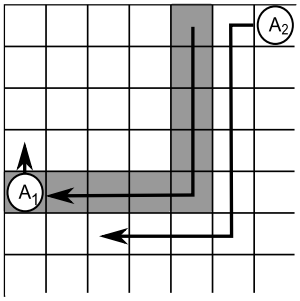


FIGURE 4.7 – Présentation de la situation initiale permettant d'expliquer le comportement exploratoire des trois approches.

L'agent A_1 vient de s'enfermer avec la trace qu'il a laissée. L'agent A_2 va longer la trace de l'agent A_1 pendant que celui-ci visite le carré de cellules en haut à gauche.

Regardons maintenant vers quels résultats vont tendre les trois approches (figure 4.8). Il s'agit d'un cas très simplifié mais qui représente bien la dynamique de chaque algorithme et permet d'expliquer les difficultés exploratoires qu'ils peuvent rencontrer.

Pour EVAP : L'agent va visiter le carré de cellules en haut à gauche. Lorsqu'il essayera d'en sortir, il se trouvera face à la trace récente (claire) de l'agent A_2. Par définition du comportement des agents EVAP, A_1 va se diriger vers la marque la plus ancienne (sombre), rester bloqué dans ce carré de cellules (il va suivre la zone sombre puisque les traces qui l'entourent sont plus "fraîches" et donc moins attractives) et les visiter une fois de plus avant de pouvoir explorer le reste de l'environnement. Il s'agit d'un *comportement très sous-optimal directement lié à la signification du marquage* d'EVAP.

Pour LRTA* : L'agent LRTA* va également visiter le carré de cellules en haut à gauche. Il va cependant rapidement créer un gradient de marques qui le dirigera vers des cellules encore inexplorées. La sémantique du marquage permet aux agents de trouver un chemin vers les nœuds inexplorés de l'environnement.

Pour CLInG : L'agent CLInG suivra initialement le même comportement que l'agent EVAP. Cependant, lorsqu'il aura terminé la visite du carré de cellules, l'oisiveté propagée d'une cellule inexplorée proche (en vert sur la figure) l'attirera directement vers une zone à explorer. La propagation de l'oisiveté permet d'éviter un comportement proche de celui d'EVAP.

Nous voyons donc que les phéromones d'EVAP peuvent agir comme un *mur* empêchant les agents de passer. Ce phénomène est assez courant et handicape fortement EVAP pour la couverture. Il y a toutefois une conséquence positive à ce comportement puisqu'il est possible de voir les agents "bloqués" comme ayant commencé la patrouille des zones déjà explorées. Nous avons en effet pu observer expérimentalement que la répartition spatiale des agents est homogène à l'issue de la couverture. EVAP évite ainsi de former des "paquets" d'agents et de laisser certaines zones de l'environnement sans surveillance.

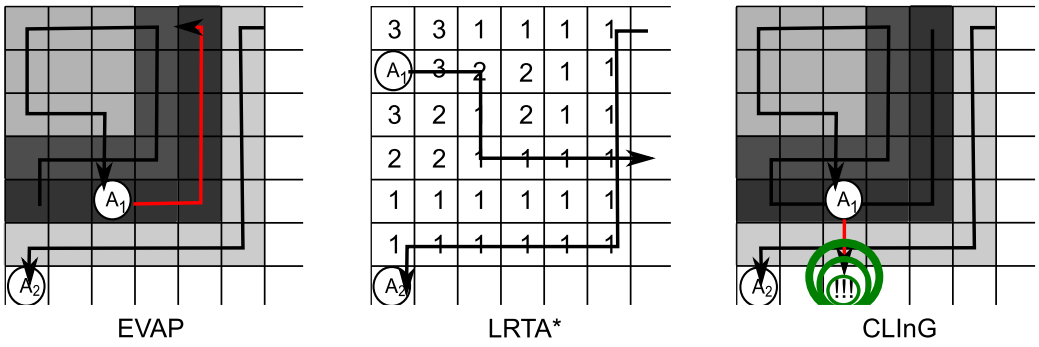


FIGURE 4.8 – Comportements des trois algorithmes à partir de la situation initiale de la figure 4.7.

Sur des environnements complexes

Patrouiller un environnement plus complexe (comportant des obstacles et des *pièces*) amplifie l'impact des comportements sous-optimaux présentés précédemment (toujours à cause de la vision locale des agents, ce qui est vrai pour tous les algorithmes fourmis). Ces environnements sont plus difficiles à explorer principalement à cause de la présence de certains nœuds de faible degré qui agissent comme des goulots d'étranglement. Ces nœuds correspondent aux portes que nous pouvons par exemple observer sur les environnements *pièces imbriquées* et *n-pièces*.

Ces nœuds sont des points de passage obligatoires pour pouvoir entrer dans les pièces et les explorer (comme le nœud c_e sur la figure 4.9), mais du fait de leur faible degré, ils ont une plus faible probabilité d'être explorés que les autres.

En supposant que l'ensemble des agents soit en dehors de la pièce, le nœud c_e ne peut être atteint que depuis le nœud c_1. En supposant également qu'un agent soit arrivé de c_4 (c_4 est donc marquée), la probabilité de visiter c_e est donc de $\frac{1}{3}$ (l'agent doit choisir aléatoirement entre c_e, c_2 et c_3). Lorsque plusieurs pièces sont imbriquées, la probabilité d'atteindre les pièces suivantes à la première visite est encore plus réduite puisque les agents auront à effectuer un choix semblable pour entrer dans chaque nouvelle pièce.

La difficulté à explorer un environnement est donc directement liée à sa topologie.

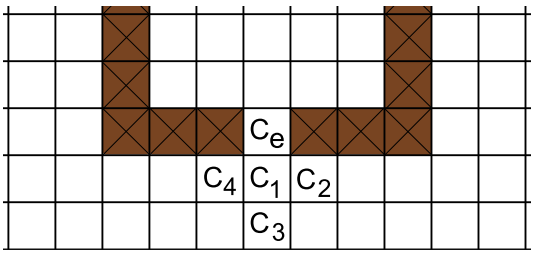
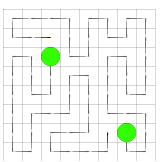


FIGURE 4.9 – Schéma représentant l'entrée d'une *pièce*.



Cas d'EVAP — Lorsqu'on combine un environnement difficile à explorer comme les *pièces imbriquées* avec le comportement exploratoire sous-optimal d'EVAP, nous obtenons une couverture relativement médiocre.

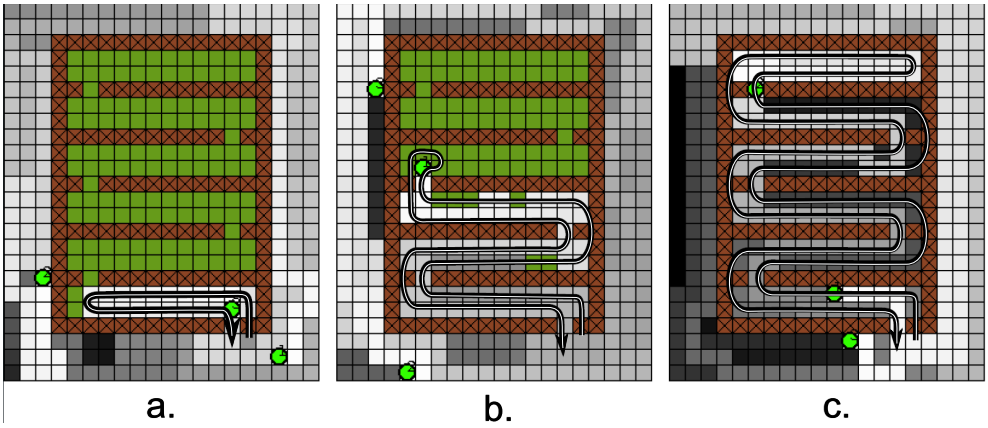


FIGURE 4.10 – Exploration des pièces imbriquées par EVAP.

Premier problème : lorsqu'un agent entre dans les pièces imbriquées, il en ressort avant de les avoir entièrement visitées (cf. figure 4.10).

Sur la première image (figure 4.10.a), un agent se trouvait devant l'entrée de la seconde pièce et a choisi de se déplacer vers le bas plutôt que d'y rentrer (la probabilité de visite des cellules en haut, en bas et à gauche étant équiprobable puisqu'elle n'ont pas encore été visitées). A la seconde tentative (figure 4.10.b), un agent a réussi à pénétrer jusque dans la quatrième pièce avant de ressortir. Sur la dernière image (figure 4.10.c), nous pouvons voir que le gradient de phéromones, formé lors des précédentes visites, guide les agents jusqu'à la dernière pièce imbriquée avant de les amener efficacement vers la sortie.

Second problème : lorsqu'un agent EVAP visite la cellule c_1 et y dépose la quantité maximale Q_{max} de phéromones, cette cellule ne pourra plus être visitée par un agent tant qu'elle n'est pas le minimum local des cellules c_2, c_3 et c_4 (ce qui peut prendre un certain temps). La visite de c_1 amène a deux sous cas :

- L'agent rentre dans la pièce. Dans ce cas, l'agent visitera la pièce mais bloque ainsi temporairement l'entrée aux autres agents. D'autres agents pourront éventuellement entrer pour aider à explorer la pièce lorsque c_1 sera le minimum local de ses voisines. Toutefois, à chaque fois qu'un agent entre dans la pièce il en rebloque temporairement l'entrée.
- L'agent choisit de ne pas entrer dans la pièce. Dans ce cas les pièces ne seront pas visitées avant que c_1 ne redevienne le minimum local de ses voisines.

Dans les deux cas, il ne s'agit pas d'un comportement très intéressant pour l'exploration.

4.3.3 Les algorithmes fourmi et la couverture

Nous avons vu que la perception locale des agents peut les conduire à effectuer des choix sous-optimaux faisant baisser leur efficacité pour la couverture et que l'impact de ces choix

sous-optimaux sur la couverture varie en fonction de la topologie de l'environnement. Certains algorithmes réussissent toutefois à résoudre partiellement ce problème, soit grâce à la diffusion d'informations (comme CLInG) soit par le choix d'une sémantique du marquage adaptée au problème (LRTA*). Il est par ailleurs intéressant de noter que le gain dû à la propagation d'oisiveté de CLInG est assez faible par rapport à LRTA* (surtout en considérant les complexités relatives de ces deux algorithmes). Les agents EVAP ont quant à eux tendance à se piéger temporairement dans des zones déja explorées, ce qui conduit à une exploration de l'environnement en retrait par rapport aux deux autres approches étudiées.

Paramétrage de CLInG CLInG est le meilleur des trois algorithmes pour la couverture mais nous rappelons qu'il doit être configuré pour fonctionner correctement. Les paramètres utilisés dans cette section (α=10, β=20, OP_{min}=30) maximisent l'appel des agents par les cellules, *via* une faible atténuation de la propagation. Ce jeu de paramètres produit cependant une patrouille particulièrement peu efficace en comparaison d'EVAP (l'oisiveté moyenne atteint, en moyenne sur l'ensemble des environnements présentés, une valeur supérieure à 2 fois l'oisiveté optimale théorique), comme présenté sur la figure 4.11.

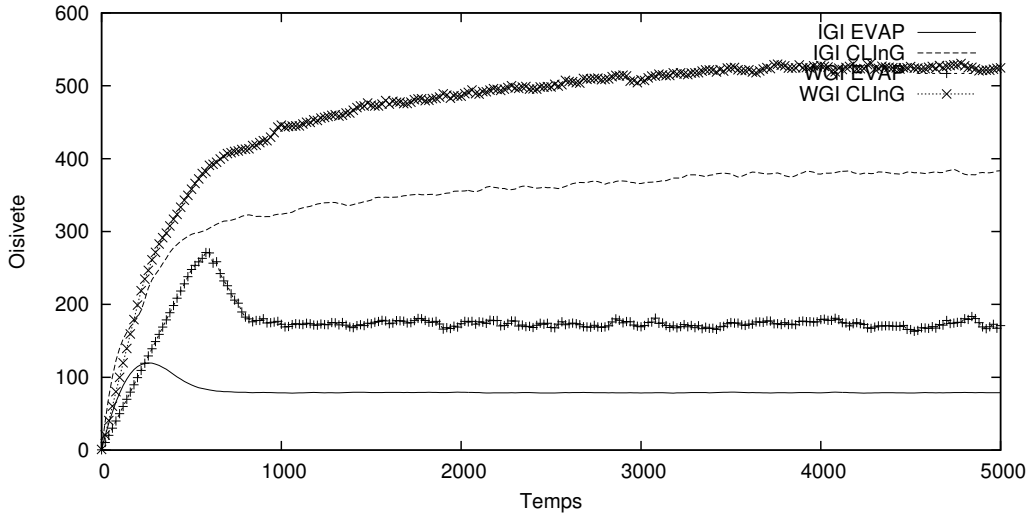
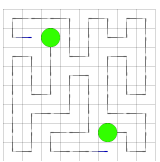


FIGURE 4.11 – Comparaison des oisivetés d'EVAP et de CLInG utilisant un paramétrage favorisant l'exploration.

4.4 Etude de la patrouille (phase d'organisation)

Nous proposons ici d'étudier la patrouille en elle-même. Nous comparons à nouveau les performances d'EVAP, de LRTA* et de CLInG sur les environnements utilisés en section précédente. Nous mettons en avant les changements que nous pouvons observer concernant l'efficacité des algorithmes par rapport à la couverture. En effet, alors qu'EVAP produit une couverture en net retrait par rapport à LRTA* et CLInG, il présente des performances analogues à CLInG pour la patrouille. LRTA*, qui couvre l'environnement plutôt efficacement, réalise pour sa part une patrouille médiocre.



4.4.1 Comparaison des patrouilles d'EVAP, de LRTA* et de CLInG

Nous avons montré précédemment l'intérêt que représente la propagation de l'information pour l'exploration de l'environnement. Là où EVAP réalise une exploration en aveugle [27], CLInG est informé des cellules ayant besoin d'être visitées. De la même manière, les agents LRTA* disposent toujours d'une estimation de la distance à une cellule encore inexplorée qui leur permet de couvrir efficacement l'environnement.

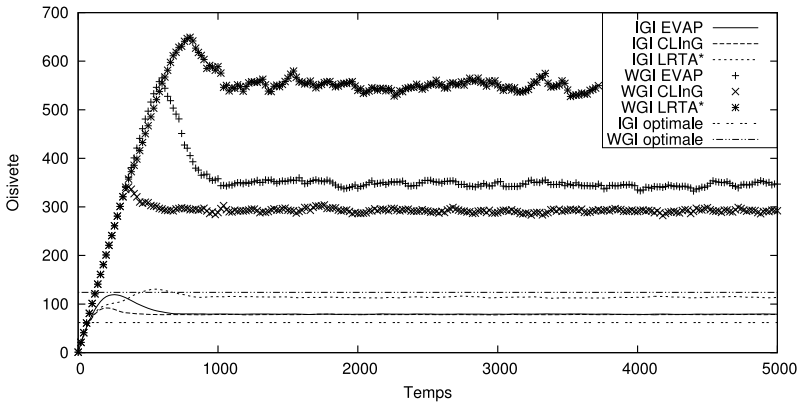


FIGURE 4.12 – Comparaison EVAP/LRTA*/CLInG sur l'environnement 50x50 cellules avec 20 agents

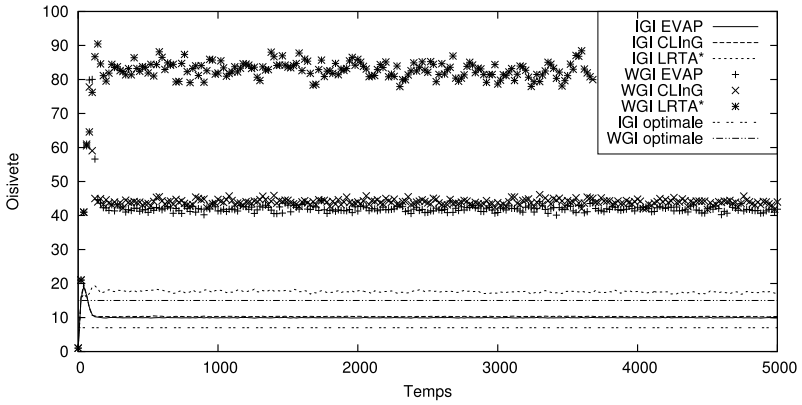


FIGURE 4.13 – Comparaison EVAP/LRTA*/CLInG sur l'environnement Densité avec 20 agents

Lors de la phase de patrouille, les agents EVAP utilisent l'information déposée préalablement pour effectuer leur tâche plus "intelligemment". Nous étudions dans cette section le comportement des trois modèles pour la patrouille.

Les figures 4.12 à 4.16 présentent les courbes d'oisiveté (IGI pour l'oisiveté moyenne et WGI pour la pire oisiveté) pour les 5 environnements présentés par la figure 4.3.

27. Les agents choisissent une destination aléatoirement mais leur exploration reste dirigée par les phéromones qu'ils déposent et qui les empêchent de revenir tout de suite sur leurs pas.

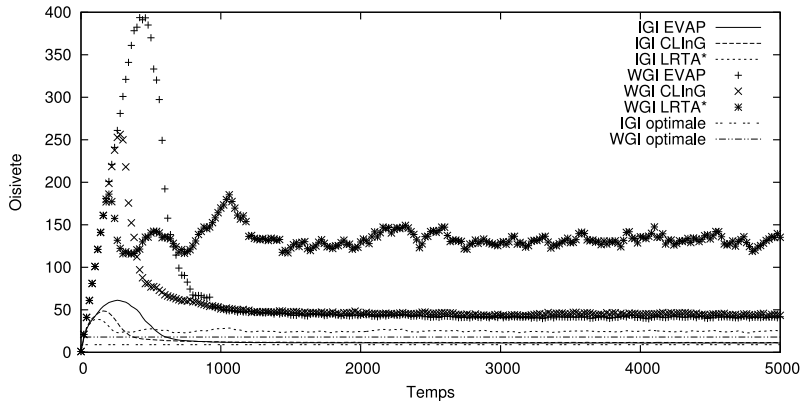


FIGURE 4.14 – Comparaison EVAP/LRTA*/CLInG sur l'environnement pièces imbriquées avec 20 agents

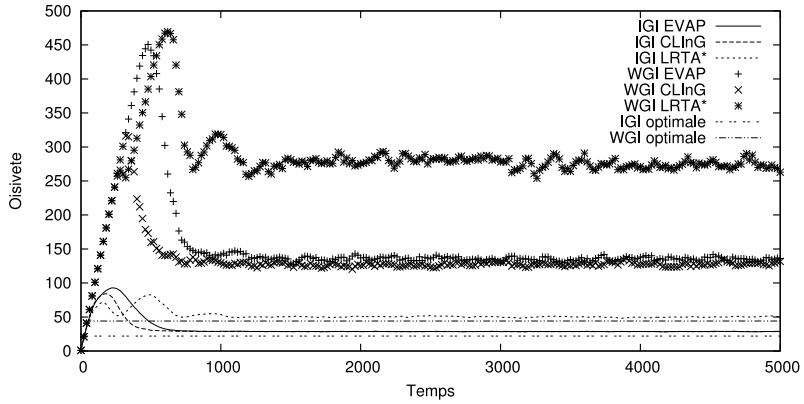
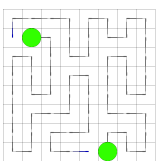


FIGURE 4.15 – Comparaison EVAP/LRTA*/CLInG sur l'environnement Koenig avec 20 agents

Nous pouvons constater que le "classement" des modèles est profondément modifié. LRTA* est systématiquement derrière EVAP et CLInG [28], aussi bien en oisiveté moyenne (IGI) qu'en pire oisiveté (WGI). La différence sur l'IGI est variable selon les environnements mais consiste en une augmentation de 20 à 50% par rapport aux oisivetés d'EVAP et CLInG.

Ce phénomène peut s'expliquer par la sémantique du marquage de LRTA*. En effet, l'algorithme ne considère pas l'oisiveté des nœuds, ce qui déconnecte le comportement de LRTA* de la tâche à accomplir. La figure 4.17 représente un environnement patrouillé par 20 agents LRTA*. La figure 4.17.a représente le champ scalaire créé par les agents (qui a été coloré pour être plus lisible). La figure 4.17.b présente la carte d'oisiveté correspondante (les cellules claires correspondent aux plus hautes oisivetés). Nous pouvons remarquer la corrélation relativement faible entre la valeur du marquage et l'oisiveté. Il est par exemple possible de trouver des cellules cyan (dans le centre de l'image, valeur du marquage = 3) ayant une oisiveté plus élevée que des cellules vertes (en haut à droite, valeur du marquage = 2). Les cellules vertes seront

28. CLInG utilise désormais les paramètres α=60, β=20, OP_{min}=4 et les fera varier légèrement en fonction de l'environnement et du nombre d'agents.



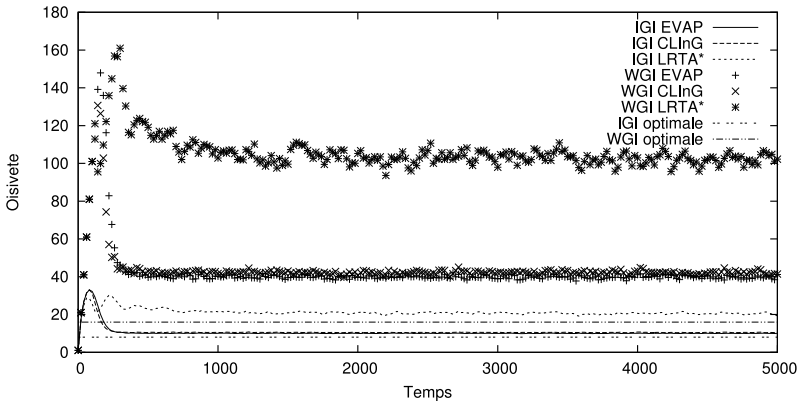
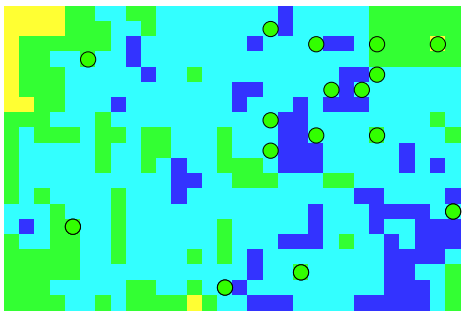
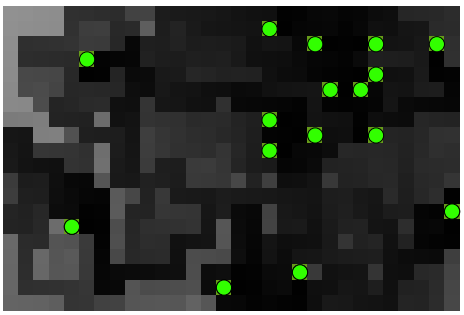


FIGURE 4.16 – Comparaison EVAP/LRTA*/CLInG sur l'environnement n-pièces avec 20 agents

cependant visitées préférentiellement par rapport aux cellules cyan. A partir des informations dont ils disposent, il est impossible pour les agents de discriminer finement l'oisiveté. LRTA* a de plus tendance à créer des "plateaux" (un ensemble de nœuds de même valeur) présentant un gradient nul. Lorsqu'un agent se trouve sur un tel plateau, il adopte un comportement proche de celui des agents d'EVAP lors de la phase d'exploration (à cause du gradient nul sur ces plateaux). Il en résulte une patrouille erratique qui ne cherche pas à minimiser l'oisiveté. Le marquage de LRTA*, intéressant pour la couverture (une estimation de la distance à la cellule inexplorée la plus proche), perd une grande partie de son sens pour la patrouille.



a. plateaux de couleurs présentant un gradient nul. (jaune=1, vert=2, cyan=3, bleu=4)



b. carte d'oisiveté correspondant à l'image a. (sombre = oisiveté faible)

FIGURE 4.17 – Représentation par niveaux de couleur du champ scalaire de LRTA* et de l'oisiveté pendant la patrouille.

Concernant EVAP et CLInG, il est surprenant de constater qu'après l'exploration, les deux modèles présentent des IGI quasiment équivalentes, avec parfois un avantage en faveur d'EVAP. Un point les distingue cependant. La pire oisiveté est meilleure (c'est-à-dire plus basse) pour CLInG. Le gain sur la WGI provient de l'appel des cellules de très forte oisiveté. Toutefois, ce gain reste assez limité. Compte tenu de la différence entre les IGI des deux modèles, le nombre de cellules de forte oisiveté pour EVAP est faible. Ce changement de comportement provient des phéromones déposées lors de la couverture qui permettent à EVAP

de réaliser une patrouille plus efficace.

4.4.2 Impact du nombre d'agents sur la patrouille

Nous nous intéressons ici au comportement des algorithmes fourmis lorsque le nombre d'agents augmente. Les interactions entre agents peuvent mener à des comportements sous-optimaux (cf. 4.3.2). L'augmentation du nombre d'agents peut alors conduire à des phénomènes d'encombrement influant sur la qualité du comportement du système (stagnation ou baisse des performances passée une masse critique d'agents). LRTA* étant en net retrait par rapport à EVAP et CLInG, il ne présente pas d'intérêt pour cette étude. Nous choisissons de nous concentrer sur EVAP et CLInG.

La figure 4.18 présente l'impact du nombre d'agents patrouilleurs sur l'oisiveté. L'expérience a été réalisée sur l'environnement 50x50 cellules pendant 10 000 itérations et a été répétée 100 fois. La métrique utilisée correspond à la moyenne des IGI sur la durée des expériences.

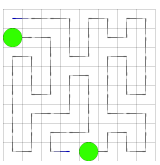
La figure 4.18.a présente l'oisiveté comparée d'EVAP, de CLInG et l'oisiveté optimale pour un nombre d'agents donné. Nous voyons que les deux modèles proposent, à l'image des expériences précédentes, des performances proches quel que soit le nombre d'agents présents. Nous pouvons cependant noter qu'EVAP est légèrement meilleur lorsque le nombre d'agents augmente.

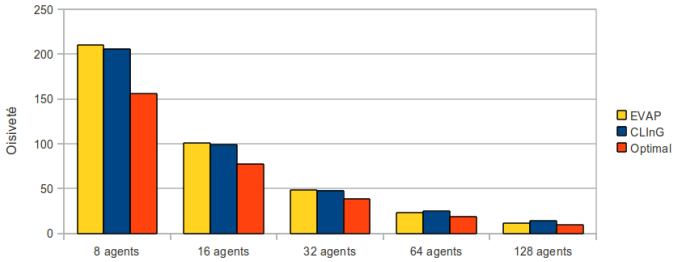
La figure 4.18.b présente le rapport de l'IGI moyenne observée sur l'IGI optimale. Ce rapport permet de savoir si l'IGI du système s'approche ou s'éloigne de l'IGI optimale lorsque le nombre d'agents varie.

La figure 4.18.c présente le *coefficient d'amélioration* de l'oisiveté lors du doublement du nombre d'agents (c'est-à-dire l'IGI pour n agents sur l'IGI pour $2n$ agents). Ce coefficient permet d'observer le gain de performance lorsque le nombre d'agents double, c'est-à-dire de déterminer si doubler le nombre d'agent permet de diviser l'oisiveté moyenne du système par deux.

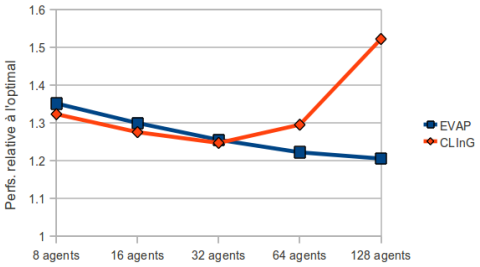
Nous pouvons voir que pour le cas d'EVAP, coefficient d'amélioration de l'oisiveté est supérieur à 2 et à peu près constant quel que soit le nombre d'agents. La variation de l'oisiveté, lorsqu'on augmente le nombre d'agents, n'est donc pas linéaire et plus le système comporte d'agents, plus il est efficace. Les agents, lorsqu'ils sont peu nombreux à patrouiller, réalisent souvent des choix sous-optimaux à cause de leur vision locale. Lorsque le nombre d'agents augmente, si un agent effectue un de ces choix sous-optimaux, un autre agent passera rapidement derrière lui en faisant l'autre choix.

Dans le cas de CLInG, nous observons, à partir de 32 agents, une importante réduction du gain de performance lors de l'ajout d'agents supplémentaires. Passer par exemple de 32 à 64 agents ne permet pas de diviser l'IGI moyenne du système par 2. Ce phénomène provient de la propagation même de l'information qui provoque un agglutinement des agents autour des cellules de forte oisiveté et un délaissement du reste de l'environnement (les agents se déplacent en petits groupes au lieu de se répartir de façon homogène).

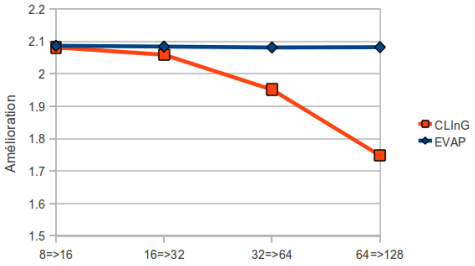




a. Oisiveté moyenne
classement par nombre d'agents



b. Rapport oisiveté/oisiveté optimale



c. Amélioration lors du doublement du nombre d'agents (oisiveté n agents/oisiveté $2n$ agents)

FIGURE 4.18 – Comparaison des moyennes des IGI sur la durée des simulations des modèles EVAP et CLInG et des valeurs optimales théoriques sur l'environnement 50x50 cellules avec un nombre croissant d'agents.

4.4.3 Conclusion sur la patrouille par les algorithmes fourmis

Nous nous sommes intéressés aux problèmes de la couverture et de la patrouille par des algorithmes fourmis. En plus d'EVAP, nous avons utilisé l'algorithme LRTA* qui repose sur un marquage n'utilisant pas le concept d'oisiveté et CLInG qui propose de compenser les perceptions locales des agents par la propagation de l'oisiveté par les cellules elles mêmes.

Selon que l'on s'intéresse à la couverture ou à la patrouille, les résultats sont différents.

Concernant la couverture, EVAP est handicapé par un comportement peu exploratoire résultant de la signification du marquage (les agents ne (re)visitent pas les cellules visitées récemment) et des choix sous-optimaux réalisés par les agents. EVAP est donc sur ce point en très net retrait par rapport à LRTA* et CLInG. Il est intéressant de noter le faible écart de performance entre CLInG et LRTA* malgré la différence de complexité entre les deux approches.

EVAP se rattrape cependant lors de la patrouille. Les phéromones déposées lors de la couverture permettent de guider les agents vers les zones les plus anciennement visitées. En pratique, nous pouvons observer la formation de *chemins* plus ou moins stables dans le temps qui seront réutilisés par les agents (Nous reparlerons de ces chemins en chapitre 6 dans le cadre de la formation des comportements cycliques). La patrouille d'EVAP est alors très proche de celle de CLInG (qui garde généralement l'avantage sur la WGI). La propagation de l'oisiveté ne permet pas de creuser un écart significatif avec le simple marquage de l'environnement d'EVAP. Dans certains cas, notamment lorsque le nombre d'agents augmente, EVAP est même capable de prendre l'avantage sur CLInG. La propagation de l'oisiveté agit alors comme une

perturbation qui a tendance à rassembler les agents en petits groupes. LRTA* réalise une patrouille médiocre, principalement à cause de son marquage qui ne repose pas sur l'oisiveté. Un agent LRTA* ira toujours visiter la cellule de plus basse valeur. Cette cellule ne correspond cependant pas nécessairement à la cellule de plus haute oisiveté de son voisinage.

En conclusion, EVAP est une approche simple qui est capable de rivaliser avec des algorithmes plus complexes même si ses performances sur les premiers instants de la patrouille sont en retrait.

4.5 Etude de la robustesse d'EVAP

Les systèmes auto-organisés présentent souvent une certaine robustesse aux perturbations mais peuvent également être mis en difficulté lors de changements dans les conditions d'exécution. Nous nous penchons donc sur la robustesse d'EVAP à travers l'étude de deux situations :

Robustesse à la perte d'agents Il peut arriver, dans l'hypothèse d'une implémentation robotique, que des agents doivent s'arrêter de patrouiller. Une panne matérielle, un blocage dans l'environnement ou un besoin de ravitaillement en énergie sont autant d'évènements qui peuvent venir perturber le système. Dans le cadre d'une approche *offline* par planification, des tels évènements risquent d'impacter fortement la qualité de la patrouille (une zone peut par exemple ne plus être visitée par les agents) et obligerait une replanification de la patrouille pour revenir à un comportement efficace. Nous étudions l'effet de ce type de perturbations sur le comportement d'EVAP sur un ensemble d'environnements de référence.

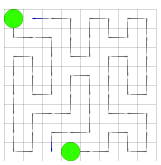
Robustesse au changement d'échelle Jusqu'à présent, nous avons toujours examiné le comportement d'EVAP sur des instances relativement réduites du problème de la patrouille. Une difficulté particulière lors de la conception des systèmes artificiels concerne le passage à l'échelle. Typiquement, certaines approches sont limitées par l'explosion combinatoire de l'espace d'état lors de l'augmentation de la taille des instances du problème à résoudre, d'autres sont limitées par des contraintes physiques, telles que la bande passante des canaux de communication. Nous proposons donc d'étudier le comportement d'EVAP sur des environnements de grande taille avec un plus grand nombre d'agents.

4.5.1 Robustesse à la perte d'agents

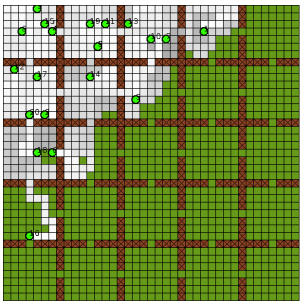
Pour étudier la robustesse du modèle EVAP à la perte d'agents, nous proposons d'utiliser l'environnement "25 pièces" présenté sur la figure 4.19.a, particulièrement problématique concernant la phase d'exploration.

L'expérience se déroule en deux temps :

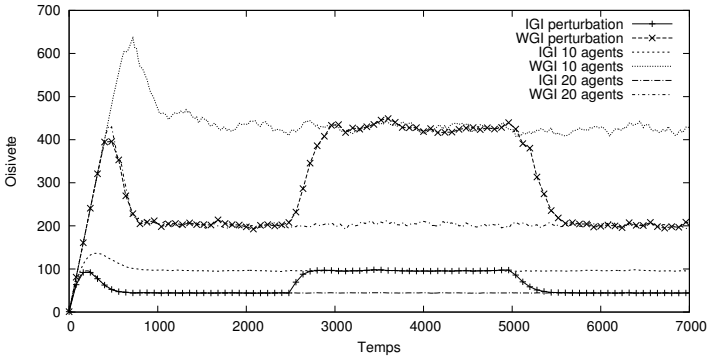
- Après un stabilisation des deux mesures d'oisiveté, nous supprimons la moitié des agents. Nous pourrons donc voir l'impact sur la patrouille de la défaillance d'un certain nombre d'agents, en particulier si les agents restants sont capables de se réorganiser rapidement pour se redistribuer équitablement la charge de travail.



– Après re-stabilisation des deux mesures d'oisiveté, nous ré-injectons dans la simulation les agents défaillants (que nous aurons réparés pendant ce laps de temps) par le point d'entrée initial (soit le coin supérieur gauche de l'environnement). Nous nous intéresserons alors à la capacité du système à retrouver le niveau de performances précédant la suppression des agents.



a. L'environnement 25 pièces



b. Oisiveté lors de l'ajout/suppression d'agents

FIGURE 4.19 – Robustesse à la perte d'agents : oisiveté sur l'environnement 25 pièces avec un passage de 20 à 10 agents puis réintroduction de 10 agents supplémentaires.

La figure 4.19 présente les valeurs moyennes d'oisiveté sur 100 répétitions de l'expérience présentée précédemment avec un passage de 20 à 10 agents à 2500 itérations puis réinjection de 10 agents dans la simulation à 5000 itérations. Nous comparons ces données à deux cas témoins, consistant en 100 répétitions d'un patrouille sans perturbations avec respectivement 10 et 20 agents sur le même environnement.

Nous pouvons voir que le système s'adapte très bien à la suppression et à l'ajout d'agents en cours de simulation puisque l'oisiveté (aussi bien en IGI qu'en WGI) s'aligne rapidement sur les cas témoins. Ceci illustre la capacité du système à se réorganiser pour s'adapter aux nouvelles conditions de patrouille en redistribuant les agents afin d'assurer leur bonne répartition dans l'environnement [29].

D'un point de vue qualitatif, nous pouvons observer sur la figure 4.19 que, suite à la suppression des agents, l'oisiveté (IGI et WGI) *ne dépasse pas* les valeurs de l'expérience témoin avec 10 agents. Cela signifie que la réorganisation spatiale des agents est effectuée rapidement et que les zones laissées "à l'abandon" lors de la disparition de la moitié des agents ont immédiatement été prises en charge par le reste de l'essaim. D'autre part, l'ajout d'agents permet de rétablir les valeurs d'oisiveté observées avant la suppression des agents dans un délai inférieur au temps nécessaire à l'exploration initiale de l'environnement et à la stabilisation des valeurs d'oisiveté. Ceci montre à nouveau l'intérêt du champ de phéromones qui guide les agents vers les zones les plus anciennement visitées.

29. Nous rajouterons : de manière complètement décentralisée, sans intervention extérieure et par la seule action de l'évaporation des phéromones.

4.5.2 Passage à l'échelle

Nous étudions ici le comportement du modèle EVAP lors de l'augmentation de la taille de l'environnement et du nombre d'agents dans des proportions importantes. La preuve de patrouille nous assure que le modèle fonctionnera toujours correctement mais ne donne pas d'indications sur les performances qu'il sera capable d'atteindre. Pour cela, nous observons l'évolution de l'oisiveté (en IGI et en WGI) sur un environnement (rectangulaire sans obstacles) de 20 par 40 cellules avec 6 agents et sur un environnement de 200 par 400 cellules avec 600 agents. Ces deux expériences utilisent la même densité d'agents (soit 133,33 cellules par agents) et possèdent donc les mêmes oisivetés optimales théoriques. En théorie, nous devrions donc observer des IGI et WGI équivalentes pour les deux expériences.

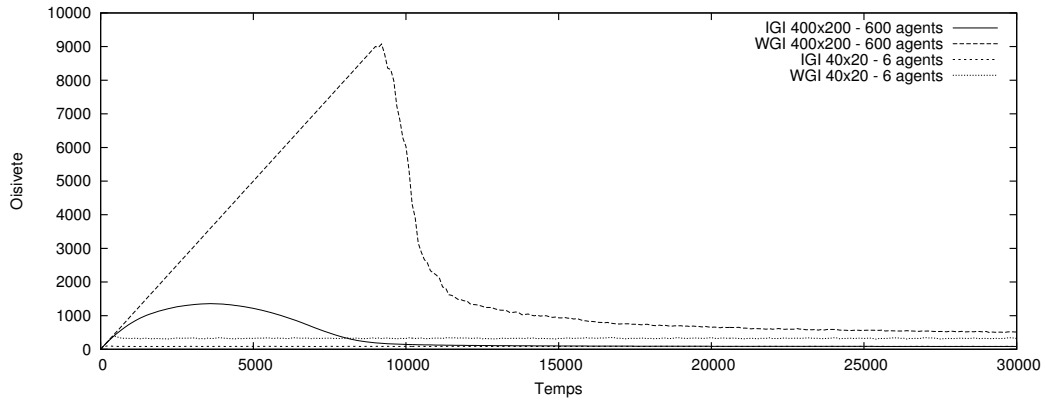
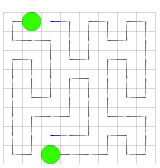


FIGURE 4.20 – Oisiveté (IGI et WGI) sur les environnements 400×200 avec 600 agents et 40×20 avec 6 agents (même densité d'agents)

La figure 4.20 présente les oisivetés (IGI et WGI) moyennes obtenues sur 50 répétitions de chaque expérience sur 70.000 itérations. Le premier point remarquable concerne l'explosion du temps nécessaire à la couverture de l'environnement qui s'établit à un peu moins de 9000 itérations pour le "grand" environnement (contre 350 pour le petit). Le second point intéressant concerne la convergence, à plus long terme, des valeurs d'oisiveté pour les deux environnements. Les courbes d'IGI convergent assez rapidement (compte tenu de la taille de l'environnement) pour se rejoindre autour d'une valeur proche de l'optimal vers 20.000 itérations. Les courbes d'IGI quant à elles semblent converger asymptotiquement vers la même valeur. En observant de plus près et à plus long terme le comportement de la pire oisiveté sur l'environnement 400×200 (voir figure 4.21), nous observons que celle-ci se stabilise autour de 475 pas de temps.

L'explosion du temps nécessaire pour réaliser la couverture provient de la forte concentration d'agents près du point de départ qui se "gènent" les uns les autres. Nous pouvons observer, en particulier sur les figures 4.22.a et b que la concentration d'agents est d'autant plus forte qu'on se rapproche du point de départ de la simulation (c'est-à-dire, le coin supérieur gauche). Le nombre d'agents présents dans cette zone provoque une saturation en phéromone. Toute la zone est proche de Q_{max} (la valeur de phéromone maximum déposée par les agents) la phéromone n'ayant pas le temps de s'évaporer entre deux visites par les agents. Les agents se trouvent alors dans une situation où il leur est impossible de discriminer la moindre information de gradient et adoptent un comportement similaire à une marche aléatoire (c'est-à-dire



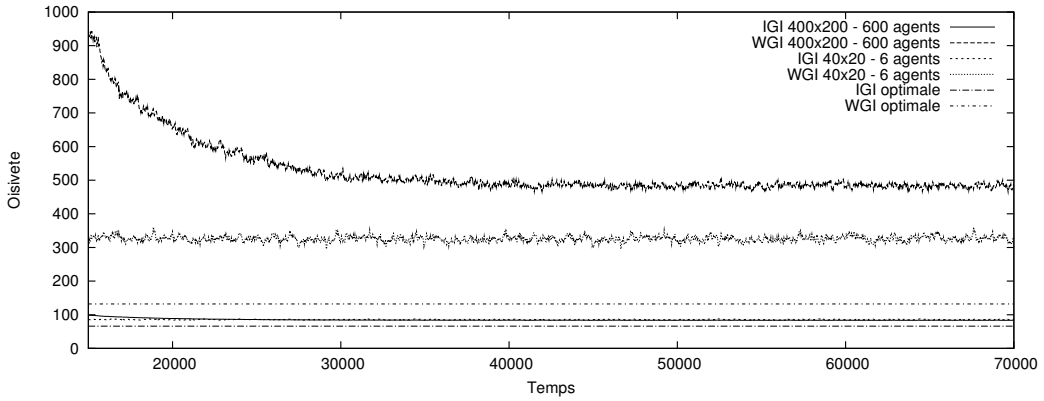


FIGURE 4.21 – Oisiveté (IGI et WGI) sur les environnements 400×200 avec 600 agents et 40×20 avec 6 agents. Zoom sur l'intervalle [15.000 ;70.000].

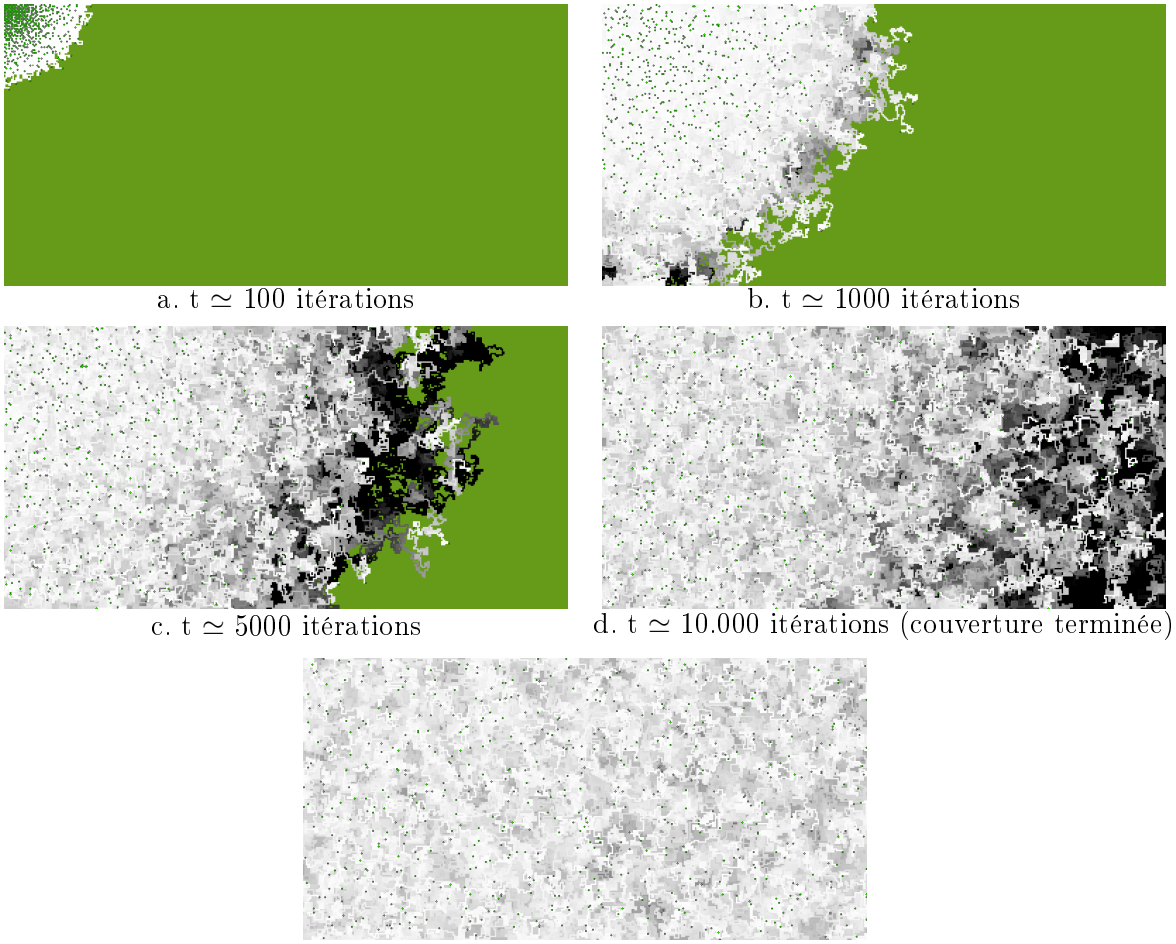
qu'ils ont une probabilité équivalente de se déplacer sur chacune des cellules voisines). La concentration en agents se réduisant à l'approche de la périphérie du champ de phéromones, les agents présents à cet endroit commencent à obtenir des informations sur le gradient et poursuivent l'exploration tandis que le reste de l'essaim continue à revisiter les nœuds proches du point de départ.

Les agents se "diffusent" lentement dans l'environnement jusqu'à sa couverture complète. Cependant, contrairement à ce qui peut être observé sur des instances plus petites, les agents ne sont pas répartis de manière homogène à travers l'environnement (voir figure 4.22.d). Ainsi, si la décroissance brutale de la WGI sur la figure 4.20 (entre 9000 et 11000 itérations) traduit la couverture de l'environnement, sa décroissance plus douce sur l'intervalle [11000, 40000] représente le temps nécessaire au système pour arriver dans un état globalement stable. La figure 4.22.e, qui représente l'état du système à environ 30.000 itérations, est encore légèrement plus sombre sur la droite[30], ce qui dénote une oisiveté un peu plus forte et donc une répartition encore imparfaite des agents à travers l'environnement. Concernant la WGI, il reste possible que les deux courbes se rejoignent à un horizon très grand. L'environnement 400×200 offre toutefois beaucoup moins de contraintes aux agents qui peuvent plus facilement "négliger" une zone pendant quelques centaines d'itérations.

Vers une amélioration de l'exploration — Si, d'après cette expérience, nous pouvons légitimement supposer qu'EVAP semble produire une patrouille d'une qualité comparable (au moins en IGI) à densité égale d'agents et quelle que soit la taille de l'environnement, nous ne pouvons pas transposer ces conclusions à l'exploration et à la couverture de l'environnement.

Notons toutefois que l'ensemble des expérimentations présentées dans ce chapitre sont réalisées dans des conditions particulières, notamment au niveau de l'ordonnancement qui utilise un ordonnanceur séquentiel cyclique (les agents agissent à chaque itération, chacun leur tour et dans un ordre fixe). Le passage à un ordonnancement synchrone (en utilisant par exemple le modèle influence réaction dont nous avons brièvement parlé en chapitre 1) pourrait produire un comportement exploratoire complètement différent (en fonction de la politique de

30. C'est un peu plus visible en s'éloignant un peu de l'image.



a. t ≃ 100 itérations

b. t ≃ 1000 itérations

c. t ≃ 5000 itérations

d. t ≃ 10.000 itérations (couverture terminée)

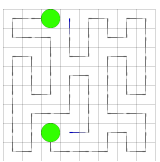
e. t ≃ 30.000 itérations

FIGURE 4.22 – Environnement de 400×200 cellules (80.000 nœuds) et 600 agents. Chaque petit point vert correspond à un agent.

résolution des conflits). Partant d'un même point de l'environnement et agissant à partir de perception identiques, tous les agents auront tendance à s'éloigner du point depuis lequel ils ont été déployés pour visiter les cellules inexplorées. Il en résulterait la formation d'un *front d'agents* qui couvrirait rapidement l'intégralité de l'environnement. Les résultats d'un tel ordonnancement sur la patrouille sont cependant difficiles à prévoir et peuvent potentiellement modifier en profondeur le comportement du modèle [31]. Il faudrait alors refaire une étude complète sur la patrouille selon ces nouvelles hypothèses.

Une autre possibilité consisterait à combiner (à la manière de ce qu'ont fait Wagner *et al.* pour VAW_0 et Node Counting [Wagner and Bruckstein, 1999]) EVAP et LRTA* pour compenser les défauts respectifs des deux approches.

31. positivement ou négativement. Bien que la preuve de patrouille ne tienne pas compte de l'ordonnancement et nous assure la patrouille de l'ensemble des nœuds de l'environnement en toutes situations, elle n'explique pas le comportement global du système ni ne donne d'indications sur ses performances.



4.6 Conclusion sur la patrouille d'EVAP

Nous avons abordé le problème de la patrouille multi-agent en environnement inconnu à l'aide d'EVAP. Nous avons étudié d'une part la couverture et d'autre part la patrouille en elle même. Pour cela, nous avons comparé EVAP à deux autres algorithmes fourmis pour la patrouille : LRTA* qui repose sur l'estimation de la distance au nœud à visiter le plus proche et CLInG qui utilise un processus actif de l'environnement pour attirer les agents vers les cellules à visiter.

Nous avons pu voir qu'EVAP propose une couverture peu performante en comparaison des deux autres approches. Cependant, après la première couverture de l'environnement, les performances d'EVAP dépassent celles de LTRA* pour atteindre un niveau proche de celles de CLInG. De plus, EVAP n'est pas limité à l'exploration d'environnements connus contrairement à CLInG (dont les cellules inexplorées doivent être capables d'attirer les agents vers elles).

EVAP est en outre capable de résister à l'ajout dynamique et à la perte d'agents en répartissant rapidement la charge de travail entre les agents pour s'adapter aux nouvelles conditions de la patrouille. Il peut aussi s'adapter à des instances de grande taille (en nombre de nœuds de l'environnement et en nombre d'agents), lui permettant d'intervenir sur des environnements inaccessibles à d'autres approches (par planification ou apprentissage notamment) du fait de la trop grande complexité du problème.

Nous continuons, dans les prochains chapitres, l'étude du modèle EVAP en mettant de côté la patrouille en elle-même pour nous intéresser, d'un point de vue à la fois théorique et expérimental, à l'émergence des comportements cycliques que nous avons déjà brièvement présentés dans le chapitre 3. La convergence d'EVAP, à long terme, vers ces comportements cycliques est intéressante pour la patrouille dans la mesure où ceux-ci permettent d'obtenir une oisiveté très proche de l'optimal, permettant à EVAP de concurrencer les approches par planification sur des environnements de taille modeste.

Chapitre 5

Outils et notations pour l'étude expérimentale des comportements cycliques

Sommaire

5.1 Description formelle de l'évolution des algorithmes fourmis 80
5.1.1 Représentation de l'état du système 80
5.1.2 L'évolution du système comme une chaîne de Markov 81
5.2 Définition des comportements cycliques 82
5.2.1 Sous-graphes transients 83
5.2.2 Sous-graphes récurrents 84
5.2.3 Cycles d'états du système 85
5.2.4 Preuve de convergence des algorithmes fourmi pour la patrouille vers des structures répétitives 85
5.3 Détection des sous-graphes récurrents dans les algorithmes fourmis 87
5.3.1 Détection de cycles dans les systèmes déterministes avec *the Tortoise and the Hare* 88
5.3.2 Détection de cycles d'états du système dans un système dynamique discret non déterministe 89
5.3.3 Détection des sous-graphes récurrents 90
5.3.4 Notre algorithme de recherche de sous-graphes 92
5.3.5 Problèmes pratiques et performances 93

Nous avons fait référence, à plusieurs reprises dans les chapitres précédents au cours de la thèse, à la convergence d'EVAP vers des "cycles". Les structures émergentes que nous pouvons observer sont cependant plus subtiles et compliquées que de simples cycles. Ce chapitre se concentre sur la détection et la qualification de ces comportements cycliques.

Nous avons vu au chapitre 3 que, lorsque plusieurs agents patrouillent l'environnement simultanément, le système n'est pas nécessairement déterministe puisqu'il peut arriver que les agents se trouvent face à un choix entre plusieurs destinations possibles (Le système peut alors évoluer vers deux états différents en fonction du choix des agents). Lorsqu'un tel événement se produit pendant la phase cyclique d'EVAP, la structure que nous pouvons observer ne correspond plus à un cycle — une séquence déterministe qui se répète indéfiniment — mais à un *sous-graphe récurrent*, une structure répétitive non déterministe (qui possède des propriétés spatiales proches de celles des cycles et permet au système d'atteindre une oisiveté comparable).

Mettre ces structures en évidence nécessite d'adopter un point de vue global, extérieur au système. Nous allons utiliser pour cela le fait que les algorithmes fourmis tels que nous les avons décrits au chapitre 2 sont des systèmes dynamiques discrets et peuvent donc être représentés par une chaîne de Markov.

Cette formalisation nous permet également de nous intéresser au problème de la détection des sous-graphes récurrents. Il existe des méthodes permettant de détecter des cycles dans des systèmes dynamiques déterministes mais il n'existe pas à notre connaissance de méthode permettant la détection des sous-graphes récurrents dans un cadre non déterministe. Une des contributions de cette thèse à été d'étendre l'algorithme de détection de cycles dans des systèmes déterministes, dit du *lièvre et de la tortue*[Floyd, 1967], à un cadre non déterministe.

5.1 Description formelle de l'évolution des algorithmes fourmis

Nous voulons représenter les algorithmes fourmis sous la forme d'une chaîne de Markov. Pour cela, nous devons en premier lieu poser la définition d'un *état du système* contenant l'ensemble des données permettant de faire évoluer le système d'un état s à un état s'.

5.1.1 Représentation de l'état du système

Nous définissons ici l'environnement $\mathcal{E}$ comme un graphe fortement connecté [32] dans lequel :

- le marquage d'un nœud $v_j \in V$ de voisinage $vois(v_j)$ par une phéromone φ est donné par $m_\varphi(v_j) \in M$, M étant l'ensemble des marquages possibles ;
- l'état d'un agent $a_i \in A$ est représenté par sa position courante $s(a_i) = v_j$.

A partir de cela, nous pouvons définir :

- un *état de l'environnement* $\mathrm{s}(\mathcal{E})$ comme l'ensemble des marques présentes sur les nœuds du graphe ;
- un *état du système* comme le tuple $\mathrm{s} = (s(\mathcal{E}), s(a_1), .., s(a_{|A|}))$, soit l'état de l'environnement augmenté des états respectifs des agents.

Etats équivalents — Afin de pouvoir comparer les états entre eux, nous introduisons le concept *d'états équivalents*.

32. Dans un graphe fortement connecté, il existe un chemin reliant toute paire de nœuds du graphe.

– Deux états de l'environnement $s(\mathcal{E})$ et $s(\mathcal{E}')$ sont dits *équivalents* si et seulement si leurs marquages m et m' diffèrent par une constante :

$$\exists k \in M, \forall c_j \in C, m(c_j) - m'(c_j) = k.$$

Dans le cas d'EVAW, nous avons $M = \mathbb{N}^+$. Nous identifierons donc $s(\mathcal{E})$ à l'état $s^*(\mathcal{E})$ dont le marquage minimum est égal à 0.

– Cette définition est extensible aux états du système qui seront équivalents si les deux *états de l'environnement sont équivalents* et si *les états respectifs des agents sont équivalents* (c'est-à-dire qu'ils se trouvent sur le même nœud).

Les agents pouvant être amenés à réaliser des choix entre plusieurs destinations, certains états du système peuvent présenter des transitions stochastiques. On nomme alors *état non déterministe* tout état pour lequel plusieurs transitions sont possibles (c'est-à-dire lorsqu'un agent a un choix aléatoire à effectuer ou lorsque les conflits entre deux agents sont réglés de manière non déterministe).

5.1.2 L'évolution du système comme une chaîne de Markov

Rappel sur les chaînes de Markov

Une *chaîne de markov* est un *processus de Markov* à temps discret. Elle est définie comme une suite de variables aléatoires $S_0, S_1, ..., S_t, ...$ définies sur un même espace d'états S (S_t représente l'état du système au temps t), cette suite vérifiant la *propriété de Markov* :

$$P(S_t|S_0, S_1, ..., S_{t-1}) = P(S_t|S_{t-1}),$$

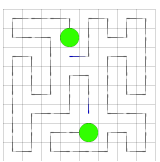
c'est-à-dire que la transition vers l'état suivant ne dépend que de l'état courant.

Nous ne considérons ici que des chaînes de Markov *homogènes dans le temps*, c'est à dire des chaînes dont la loi de transition est stationnaire :

$$P(S_t = s'|S_{t-1} = s) = P(S_1 = s'|S_0 = s),$$

La probabilité de passer d'un état s_i à un état s_j sera constante quelque soit l'instant t.

Ainsi, quand l'ensemble S des états du système est fini, la chaîne de Markov peut être vue comme un graphe orienté dont les nœuds sont les états de S et où un arc $s \mapsto s'$ représente la transition possible de s à s' et est valué par la probabilité $P(S_t = s'|S_{t-1} = s)$ (soit la probabilité au temps t d'être dans l'état s' sachant que le système se trouve dans l'état s au temps $t-1$). Afin de gagner en clarté, il est possible de ne pas dessiner tous les états déterministes du système (les nœuds avec une seule transition possible) comme présenté en figure 5.2 (par exemple, la transition entre les états s_1 et s_2 se fait via une séquence $(s_{1.1}, \ldots, s_{1.n})$ d'états déterministes). Un arc du graphe entre s et s' représentera alors l'ensemble des transitions déterministes entre ces deux états.



Représenter l'évolution des algorithmes fourmis par une chaîne de Markov

Dans le cadre des algorithmes fourmis discrets, le système suit une évolution à temps discret et la transition vers un nouvel état ne dépend que de l'état courant et du comportement (stochastique et stationnaire) des agents. Nous pouvons donc représenter son évolution par une loi de probabilité conditionnelle $p(X_t|X_{t-1})$ (qui décrit la probabilité de chaque valeur de la variable aléatoire X au temps t connaissant sa valeur au temps $t-1$, X représentant dans notre cas, l'ensemble des états possibles du système).

A condition que l'ensemble M des marquages d'une cellule soit dénombrable (ce qui est le cas puisque nous avons prouvé qu'EVAP couvre l'ensemble de l'environnement en temps fini, cf. chapitre 3) (et c'est aussi vrai pour VAW [Wagner *et al.*, 1999] et LRTA* [Koenig *et al.*, 2001]), nous avons un système aléatoire stationnaire[33] vérifiant *la propriété de Markov* telle que présentée précédemment.

Nous représentons alors l'évolution du système comme une chaîne de Markov E dont les nœuds représentent les états du système et les transitions, les évolutions possibles du système à partir d'un état. Nous ne précisons pas les probabilités des transitions entre les états qui peuvent varier d'une implémentation à l'autre selon les mécanismes de résolution de conflit. Dans le cadre de l'implémentation d'EVAP utilisée dans cette thèse, les transitions issues d'un même nœud sont équiprobables.

Ainsi, **un cycle d'états du système** sera représenté comme une séquence d'états du système se répétant indéfiniment.

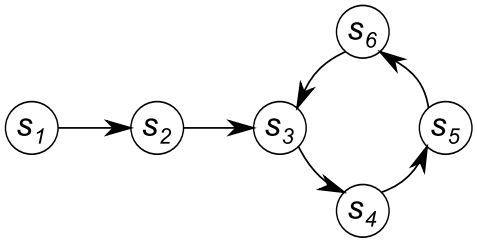


FIGURE 5.1 – Un cycle d'états du système. Le système converge à l'état s_3 vers un cycle de 4 états (s_3, s_4, s_5, s_6) qui se répétera indéfiniment.

Partant de cette formalisation de l'évolution du système, nous nous intéressons dans la section suivante à l'ensemble des structures répétitives pouvant être produites par EVAP.

5.2 Définition des comportements cycliques

Nous avons vu, en section précédente, que nous modélisions l'évolution du système par une *chaîne de Markov* représentée sous la forme d'un graphe orienté des états du système (figure 5.2).

En effet, si, dans le cadre mono-agent, l'exécution d'EVAP devient complètement déterministe après la première couverture de l'environnement (où le marquage de chaque cellule est

33. Le système est stationnaire puisque l'évolution depuis un états s vers un état s' ne dépend pas du temps.

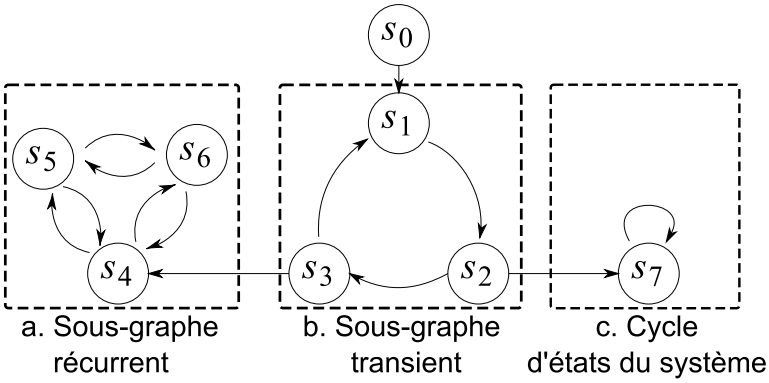


FIGURE 5.2 – Chaîne de Markov homogène représentées par un graphe orienté (dans le cas d'EVAP, les transitions à la sortie d'un nœud sont équiprobables, cf. section 5.1.2).

unique à un instant t donné), dans le cadre multi-agent, il est possible pour un agent de se retrouver dans une situation où il aura à choisir entre deux nœuds d'égales valeurs (pour n agents présents, le même marquage pourra être retrouvé jusqu'à n fois dans l'environnement, cf. conflits et hésitations, section 5.2.4). Il est ainsi possible de voir apparaître des structures contenant un ou plusieurs *états non déterministes* qui, si elles en semblent proches, ne sont pas des cycles.

Nous caractérisons alors les différentes *structures répétitives* (auxquelles nous avons référé jusqu'ici sous le terme abusif de *cycles)* pouvant être produites par EVAP comme des *sous-graphes* d'états du système plutôt que comme les chemins (des séquences de nœuds) suivis par les agents.

Nous définissons trois classes d'équivalence de sous-graphes $E' \subset E$ (soit des sous-chaînes E' de la chaîne de markov E représentant toutes les évolutions possibles du système à partir d'un état initial s_0 donné).

5.2.1 Sous-graphes transients

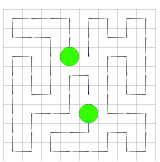
Les sous-graphes transients (cf. 5.2.b) sont des sous-graphes dans lesquels le système peut *boucler* un certain temps mais dont il **sortira nécessairement**. Ainsi, pour tout s et $s' \in E'$, il y a une probabilité non nulle (mais pas une certitude) d'atteindre s' à partir de s. Soit que :

$$\exists\ t > 0, \forall\ s \in E',\ 0 < P(S_{t+\Delta t} = s | S_t = s) < 1,$$

$$\exists\ t > 0, \forall\ s, s' \in E', P(S_{t+\Delta t} = s' | S_t = s) > 0.$$

Les états s_1, s_2, s_3 sur la figure 5.2 représentent un sous graphe transient ; à chaque visite des nœuds s_2 et s_3, le système a (selon le comportement stochastique des agents) une probabilité non nulle de sortir de ce sous-graphe transient respectivement vers les nœuds s_4 et s_7.

La figure 5.3 représente deux agents dans un sous-graphe transient. Selon le choix de l'agent 1 (qui agit en premier) le graphe peut persister (figure 5.3.b) ou disparaître (figure 5.3.c).



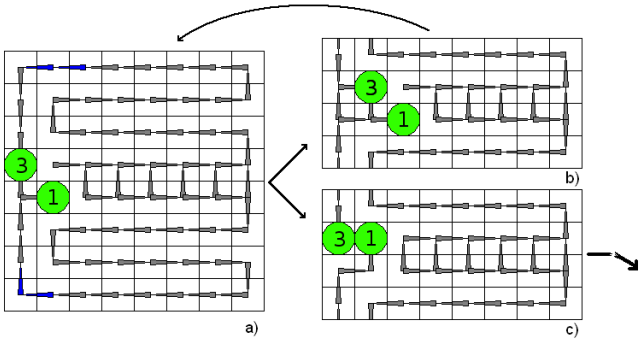


FIGURE 5.3 – Un exemple de sous-graphe transient. L'agent 1 agit toujours en premier ; tant que celui ci va vers la droite, le sous-graphe persiste ; s'il choisit de se déplacer vers le haut, le système sort du sous-graphe.

5.2.2 Sous-graphes récurrents

Les sous-graphes récurrents (cf. 5.2.a) sont des sous-graphes dont certaines transitions peuvent être non déterministes mais dont le système ne peut ressortir une fois qu'il y est entré. Ils sont définis tels que :

$$\exists\ t > 0, \forall \Delta t > 0, \forall\ s \in E \setminus E',\ s' \in E',\ P(S_{t+\Delta t} = s | S_t = s') = 0.$$

Ces sous-graphes sont des attracteurs du système. Les états s_4, s_5, s_6 sur la figure 5.2 sont les états suivis de transitions non déterministes d'un sous-graphe récurrent.

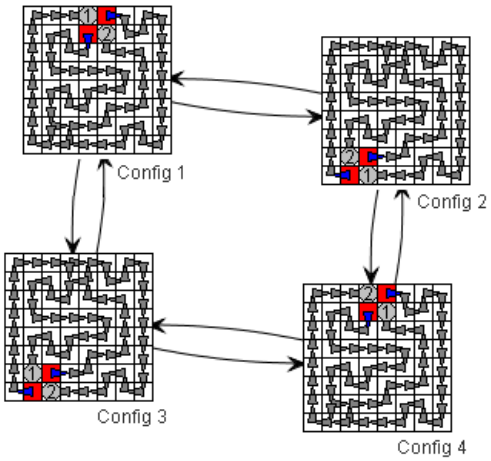


FIGURE 5.4 – Un exemple de sous-graphe récurrent. Les nœuds du graphe représentent les états non déterministes du sous-graphe. récurrent

La figure 5.4 représente un sous-graphe récurrent à 4 états non déterministes. Le système évoluera librement entre ces 4 états selon les transitions représentées par des flèches. On notera que les transitions issues d'un même nœud sont équiprobables.

5.2.3 Cycles d'états du système

Les cycles d'états du système (cf. 5.2.c) sont des *sous-graphes récurrents particuliers* dont tous les états sont suivis de transitions déterministes (c'est-à-dire que les agents ne sont confrontés à aucun conflit ni hésitation) tels que :

$$\exists\ t > 0, \forall s,\ s' \in E',\ P(S_{t+\Delta t} = s' | S_t = s) = 1$$

L'état s_7 de la figure 5.2 correspond au *premier* état de la séquence d'états du système composant le cycle.

La figure 5.5 présente un cycle d'états du système (de 64 états) patrouillé par deux agents. Le système évoluera de façon déterministe jusqu'à retrouver cet état. Durant ce cycle, chaque cellule de l'environnement sera parcourue deux fois. Il est cependant difficilement possible d'évaluer la qualité du cycle (l'oisiveté du système au cours du cycle) à partir du point de vue systémique que nous avons adopté dans ce chapitre (il est nécessaire pour cela d'étudier le chemin emprunté par les agents).

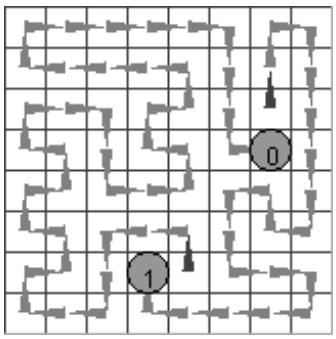


FIGURE 5.5 – Deux agents dans un cycle d'états du système. Toutes les transitions sont déterministes.

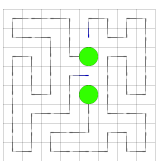
5.2.4 Preuve de convergence des algorithmes fourmi pour la patrouille vers des structures répétitives

Le formalisme posé en section 5.1 et la mise en évidence des structures répétitives pouvant être rencontrées par les algorithmes fourmi pour la patrouille nous permettent de prouver leur convergence systématique vers des structures répétitives stables (c'est à dire des sous-graphes récurrents ou des cycles d'états du système).

Les preuves suivantes reposent en partie sur le fait que les agents couvrent l'environnement en temps fini et borné. Elles s'appliquent donc à plusieurs algorithmes fourmis. Le détail du calcul de la borne sur le temps de couverture T_{vis} pour EVAP est disponible en section 3.3. Cette preuve est également applicable à LRTA* et VAW pour lesquels des bornes de couverture sont données respectivement dans [Koenig *et al.*, 2001] et [Wagner *et al.*, 1999].

Dans un premier temps, nous présentons le cas déterministe [34] pour lequel nous prouvons la convergence systématique vers des *cycles d'états du système*. Dans un deuxième temps, nous

34. Il s'agit du cas mono-agent pour EVAP/EVAW et VAW.



nous intéressons au cas non déterministe dans lequel les agents sont confrontés à des choix même après la première couverture [35]. Le système pourra alors converger vers des *sous-graphes récurrents*.

Cas déterministe

Théorème 5.2.1 (Convergence dans le cas déterministe)
Etant donné un environnement $\mathcal{E}$ et a agents au comportement déterministe [36]*, le système converge vers un* ***cycle d'états du système*** *en temps fini.*

Preuve: La preuve repose sur deux faits : 1) le nombre d'états accessibles du système est fini et 2) le comportement des agents est déterministe et stationnaire.

1) Pour tout nœud n, le temps entre deux visites successives de n est majoré par T_{vis}. Avec des états "équivalents" $s^*(\mathcal{E})$, cela conduit à : $\forall n \in N,\ 0 \leq m(c) \leq T_{vis}$.

L'ensemble des états accessibles de l'environnement est donc fini puisqu'il peut être majoré par $|A|! \times (T_{vis}+1)^{|N|}$. En effet, le marquage seul permet de savoir sur quels nœuds se trouvent les $|A|$ agents (d'où le terme $(T_{vis}+1)^{|N|}$), mais il y a $|A|!$ façon de les permuter.

2) Parce que le système est déterministe [37] et que le nombre d'états atteignables du système est fini, cette séquence converge vers un **cycle d'états du système** en temps fini.

Afin de clarifier ce que nous venons d'exposer, nous proposons — à travers le schéma présenté en figure 5.6 — de poser plus simplement l'idée de la preuve de convergence vers des cycles.

La figure 5.6.a représente l'espace (fini) d'états admissibles du système — après une première couverture de l'environnement — où chaque état S du schéma correspond à l'ensemble des *états équivalents* [38]. Après avoir visité, au pire, l'ensemble des états équivalents admissibles (figure 5.6.b) et sachant que

- le comportement du système est déterministe après la première couverture,
- le nombre d'états équivalents admissibles du système est fini, et
- le système ne s'arrête jamais (cf. preuve de patrouille, section 3.3),

le système va nécessairement repasser par un état préalablement rencontré et dès lors converger vers un cycle (voir figure 5.6.c) puisque qu'à chaque état S, on associe une transition vers un unique état S'.

35. Pour EVAP/EVAW et VAW, il s'agit du cas multi-agent. Puisque tous les agents déposent la même marque à chaque itération (une valeur fixe de phéromone pour EVAP/EVAW et la date pour VAW), chaque valeur peut être retrouvée plusieurs fois dans l'environnement. Les agents sont alors susceptibles d'avoir à réaliser un choix aléatoire. Ce cas concerne également LRTA*, en mono-agent comme en multi-agent.

36. C'est le cas en mono-agent pour EVAP et VAW et pour des versions multi-agent déterminisées (c'est-à-dire que, lorsqu'ils sont face à un choix, les agents décident de leur destination par un processus déterministe) d'EVAP et VAW et des versions déterminisées de LRTA* (mono et multi-agent).

37. Note concernant EVAP et VAW en mono-agent : Les cycles couvrant l'ensemble de l'environnement, il n'est pas possible de converger vers un cycle avant la première couverture. Le non déterminisme du système n'est donc pas gênant jusqu'à ce point. Après au plus T_{vis} pas de temps, EVAP/EVAW et VAW génèrent de manière déterministe une séquence d'états du système par récurrence ($s_{t+1} = f(s_t)$).

38. Nous rappelons au lecteur qu'ils s'agit de l'ensemble des états du système (marquage de l'environnement plus positions respectives des agents) dont les marquages diffèrent à une constante près.

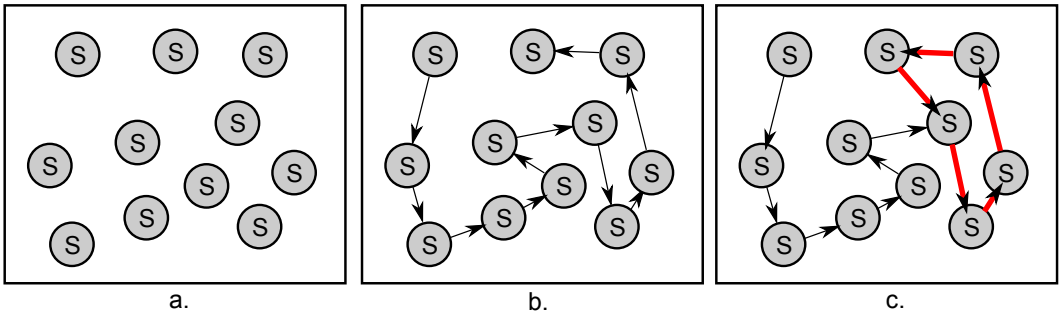


FIGURE 5.6 – Schématisation de l'idée de la preuve de convergence vers des cycles dans le cadre mono-agent

Cas non déterministe

Dans le cas non déterministes[39], il est toujours vrai que toutes les cellules sont visitées infiniment souvent et un majorant sur le nombre d'états du système accessibles reste $|A|! \times (T_{vis}+1)^{|N|}$.

Théorème 5.2.2 (Convergence dans le cas non déterministe)

Etant donné un environnement $\mathcal{E}$ et a agents au comportement non déterministe, le système converge vers un ***sous-graphe récurrent*** *en temps fini.*

Preuve: La preuve est proche de celle du cas déterministe à la différence que si les tirages aléatoires effectués sont indépendants, le comportement du système peut être modélisé par une chaîne de Markov *sans état terminal* sur un espace d'état fini. Il aboutit nécessairement dans un *sous-graphe récurrent*, les cycles d'états du système étant des cas particuliers de tels sous-graphes.

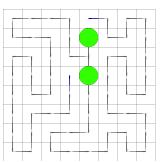
Ces deux théorèmes confirment les observations de cycles. Toutefois, ces preuves ne nous apprennent rien sur les structures produites en elles mêmes.

Ainsi, ces théorèmes ne donnent aucune indication sur la taille des structures répétitives pouvant être produites (combien d'états ils incluent). Par exemple la marche aléatoire de n agents sur un graphe fini et entièrement connecté évolue dans un sous-graphe récurrent qui est en fait l'espace d'états du système. Nous n'avons pas non plus d'indications sur la forme de ces sous-graphes récurrents (Les agents patrouillent-ils l'environnement efficacement ?).

5.3 Détection des sous-graphes récurrents dans les algorithmes fourmis

La détection automatique de comportements cycliques ou émergents est un point clé dans l'étude expérimentale des systèmes auto-organisés. La recherche de comportements cycliques

39. LRTA* en mono et multi-agent ainsi que VAW et EVAP en multi-agent



dans un système non déterministe reste, à notre connaissance, un problème ouvert. Nous présentons, dans cette section, l'algorithme du lièvre et de la tortue (The tortoise and the hare) de recherche de cycles dans des systèmes dynamiques discrets déterministes [Floyd, 1967]. Nous l'étendons ensuite à la détection de sous-graphes récurrents dans des systèmes dynamiques discrets non déterministes.

5.3.1 Détection de cycles dans les systèmes déterministes avec *the Tortoise and the Hare*

L'algorithme de détection de *cycles* proposé par Floyd [Floyd, 1967] est fondé sur le parcours d'une *séquence déterministe* d'éléments [40] par deux pointeurs en parallèle et à des vitesses différentes. Quand la tortue est "rattrapée" par le lièvre — qui parcourt la séquence deux fois plus vite que la tortue —, l'algorithme a détecté un **cycle** dans la séquence, cf. figure 5.7.

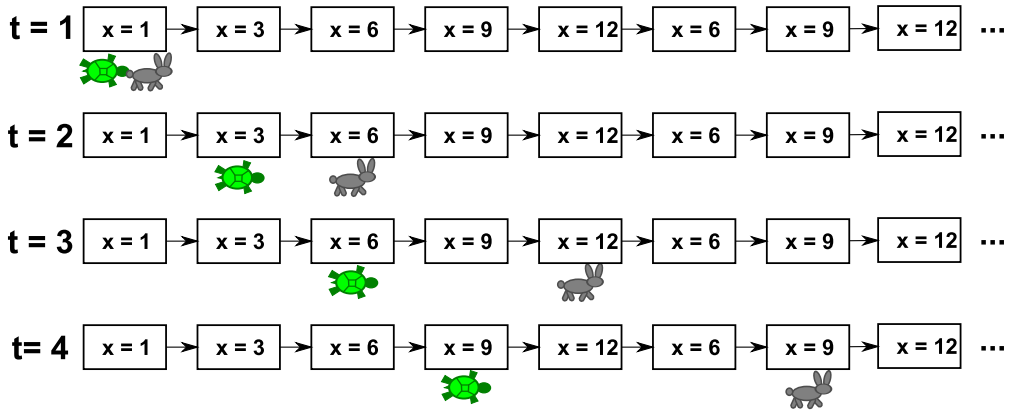


FIGURE 5.7 – Illustration de l'algorithme du lièvre et de la tortue

Considérons un ensemble X, une loi d'évolution représentée par la fonction $f : X \mapsto X$ et une suite u de premier terme $u_0 \in X$ et définie par $\forall n > 1,\ u_{n+1} = f(u_n)$.

L'algorithme du lièvre et de la tortue repose sur deux séquences déterministes u et v de même origine et parcourues à des vitesses différentes. La détection du cycle s'effectue en comparant l'évolution de u, *la tortue*, et de v, *le lièvre*, définie par $v_n = u_{2n}$. Un cycle est détecté lorsque $u_n = v_n,\ n > 0$, c'est à dire quand le lièvre rattrape la tortue, cf algorithme 12. Dans le cas des algorithmes fourmis, $u_n = v_n$ ssi u et v sont des *états équivalents*.

Algorithme 12 : L'algorithme du lièvre et de la tortue

```
1 u ← u_0;
2 v ← u_0;
3 répéter
4 |   u ← f(u);
5 |   v ← f(f(v));
6 jusqu'à u = v ;
```

40. c'est-à-dire une chaîne de Markov sans état terminal et sans transitions stochastiques

Calcul de la longueur des cycles

Un avantage de cet algorithme est sa capacité à calculer la longueur des cycles détectés. Il suffit en effet, après une première détection, de laisser se dérouler l'algorithme jusqu'à ce que le lièvre rencontre à nouveau la tortue. La longueur du cycle sera donné par le nombre d'itérations effectué entre les deux détections.

Coût de calcul et utilisation mémoire

Pour cet algorithme, le coût en mémoire pour la recherche de cycles est assez faible puisqu'il ne conserve pas d'historique de la séquence à analyser mais utilise seulement deux pointeurs sur les éléments courants. La consommation de mémoire pour la représentation du système (c'est-à-dire hors environnement de simulation) est donc seulement doublée (représentation de deux états du système au lieu d'un seul pour une simulation sans détection de cycles). Le coût de calcul est quant à lui triplé. En effet, lorsque la séquence u aura convergé vers un cycle après n itérations (la fonction f aura été appliquée n fois), la fonction f aura en plus été appliquée $2n$ fois à v.

Extension à des séquences non déterministes

Nous étendons ici l'algorithme du lièvre et de la tortue de Floyd pour gérer les situations non déterministes et donc détecter les sous-graphes récurrents présentés en section 5.2.

Concernant des versions "déterminisées" d'EVAP [41] et le cas mono-agent — la convergence ne peut survenir qu'après une première couverture de l'environnement, après quoi l'évolution du système devient déterministe —, seuls des cycles d'états du système peuvent être obtenus. On pourra donc utiliser directement l'algorithme 12. Notons également que le lièvre et la tortue représentent chacun une instance, une exécution du système, à partir des mêmes conditions initiales.

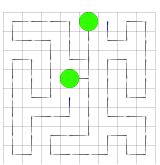
5.3.2 Détection de cycles d'états du système dans un système dynamique discret non déterministe

Evolution des séquences

Un premier point important concerne l'évolution des deux instances du système (le lièvre et la tortue) qui doivent évoluer de manière identique, en faisant les mêmes choix aléatoires afin d'éviter une divergence des deux séquences.

Pour cela, il suffit d'utiliser un générateur de nombre aléatoire propre à chaque séquence — un pour le lièvre, un pour la tortue — et utilisant tous les deux la même graine d'initialisation.

41. Des versions d'EVAP pour lesquelles les agents n'effectueraient pas de décision aléatoire en cas de multiples choix possibles mais sélectionneraient préférentiellement leur destination (par exemple, plutôt à droite qu'à gauche).



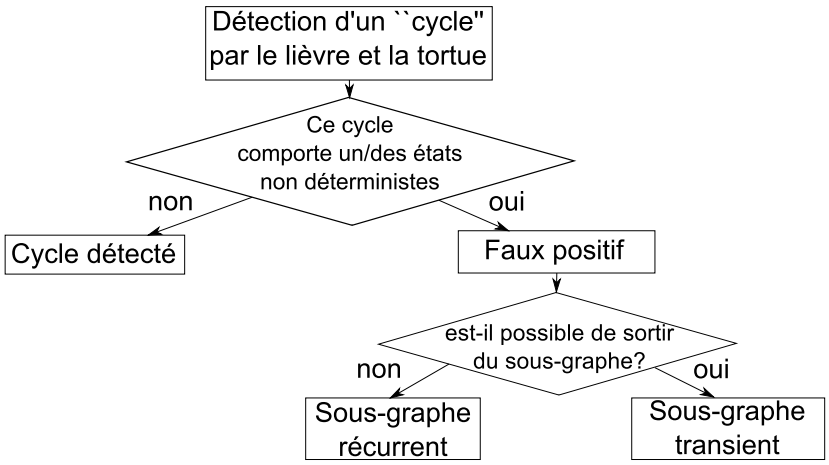


FIGURE 5.8 – Détection de faux positifs par la lièvre et la tortue.

Faux positifs

Après la première détection d'une répétition d'états du système, c'est-à-dire après que deux états équivalents aient été détectés, s'il reste des transitions non déterministes, l'algorithme du lièvre et de la tortue peut être trompé (voir figure 5.8) et s'arrêtera en cas de :

- cycle d'états du système, soit ce qui doit être détecté,
- sous-graphe récurrent — un premier type de faux positif qui, même s'il s'agit d'un comportement répétitif, ne doit pas être confondu avec un cycle par l'algorithme de détection. En effet, sans avoir étendu complètement le sous-graphe, il est impossible de savoir s'il s'agit d'un sous-graphe récurrent ou d'un sous-graphe transient—, l'algorithme ne détectant pas les transitions non déterministes entre deux visites du même état,
- sous-graphe transient — un second type de faux positif —, l'algorithme ne détectant pas les transitions non déterministes qui peuvent faire sortir le système de ce sous-graphe.

Pour éviter un arrêt de la détection en cas de faux positif, on laisse l'algorithme calculer la longueur du cycle obtenu en continuant la simulation jusqu'à rencontrer à nouveau des états équivalents (voir figure 5.9.a). Si durant cette phase :

- l'algorithme ne rencontre *que* des *états déterministes* ; nous sommes en présence d'un cycle d'états du système et l'algorithme s'arrête ;
- des *états non déterministes* sont rencontrés, nous sommes en présence soit d'un sous-graphe récurrent, soit d'un sous-graphe transient.

Si les sous-graphes récurrents correspondent effectivement à un comportement cyclique, stable dans le temps, les sous-graphes transitoires peuvent être considérés comme étant des faux positifs. Nous voyons ci-dessous comment différencier ces deux types de sous-graphes.

5.3.3 Détection des sous-graphes récurrents

L'algorithme de construction des sous-graphes récurrents que nous présentons ici étend l'algorithme de recherche de cycles de Floyd présenté précédemment.

Nous supposons que l'algorithme du lièvre et de la tortue de Floyd ait arrêté la simulation sur un état du système s identifié comme faisant partie d'un sous-graphe. Nous analysons ce sous-graphe pour déterminer sa nature (soit un sous-graphe récurrent, soit un sous-graphe transient).

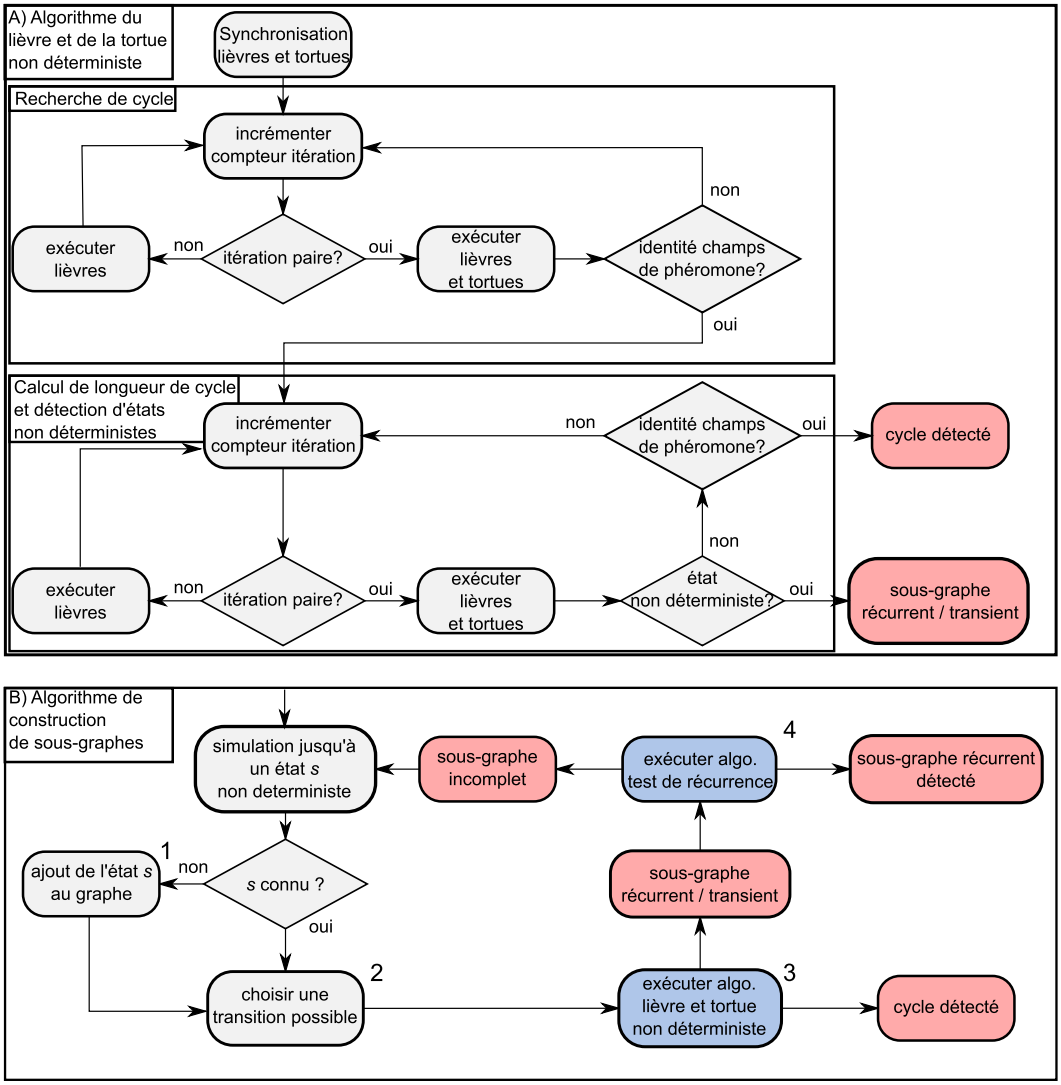
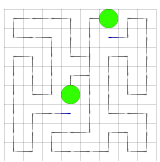


FIGURE 5.9 – Flowchart représentant l'algorithme de détection de cycles dans des séquences non déterministes

Il est possible de réaliser cette analyse de deux façons :

Développer le graphe de tous les futurs possibles

Nous construisons ici un graphe de tous les futurs possibles à partir de l'état s. Nous analysons ensuite le graphe obtenu pour déterminer :



- s'il est fait d'une seule classe d'équivalence, auquel cas il s'agit bien d'un sous-graphe récurrent ;
- s'il est fait de plusieurs classes d'équivalences, auquel cas s appartient à un sous-graphe transient.

L'objectif de la détection n'est cependant pas de générer toutes les possibilités depuis un point d'ambiguïté. Il s'agit de plus d'un processus particulièrement long et extrêmement gourmand en mémoire, le graphe construit pouvant atteindre plusieurs milliards d'états.

L'objectif est, dans le cadre d'expérimentations par simulation (sous entendu par la méthode de Monte Carlo), d'arrêter automatiquement la simulation lorsqu'un cycle ou un sous-graphe récurrent est atteint afin de pouvoir faire des statistiques sur le comportement du système en répétant les expériences.

Construire le graphe des états visités

Pour construire ce graphe, l'idée est de laisser la simulation se dérouler normalement (afin d'éviter de biaiser le résultat) mais en surveillant son déroulement. Le graphe est donc construit au fur et à mesure de l'évolution du système. Il ne s'agira donc pas du graphe de tous les futurs possible mais seulement des états visités par les agents. Ainsi, pendant la simulation, nous allons vérifier, à intervalles réguliers, si l'état courant fait parti :

- d'un sous-graphe complet, c'est-à-dire dont toutes les branches ont été visitées ; dans ce cas, nous sommes dans un sous-graphe récurrent et la simulation peut être arrêtée ;
- d'un sous graphe incomplet, c'est-à-dire dont toutes les branches n'ont pas encoré été visitées. Dans ce cas, la simulation doit se poursuivre.

C'est la combinaison de l'algorithme du lièvre et de la tortue et de cette approche qui a été retenue pour la détection des sous-graphes récurrents.

5.3.4 Notre algorithme de recherche de sous-graphes

Notre approche se décompose en trois temps. Tout d'abord, la simulation est exécutée en la supervisant avec l'algorithme du lièvre et de la tortue gérant le non déterminisme (voir figure 5.9.a).

Si un cycle est trouvé, l'exécution de la simulation est arrêtée. Dans le cas contraire (rencontre d'un état non déterministe), nous exécutons l'algorithme de construction du sous-graphe par simulation à partir de l'état renvoyé par le lièvre et la tortue (voir figure 5.9.b et algorithme 13).

Généricité

Un point intéressant de cet algorithme de détection de comportements cycliques est sa généricité puisqu'il peut être appliqué à tout système dynamique pour lequel il est possible :

- de modéliser son évolution comme une chaîne de Markov homogène ;

Algorithme 13 : Simulation et construction simultanées

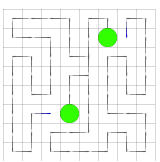
```
1  SimulationEtConstruction(s);
2  S_connus ← ∅;
3  stop ← faux;
4  répéter
5      si s ∉ S_connus alors
6          S_connus ← S_connus ⋃ {s};
7          fils(s) ← liste des transitions possibles depuis s;
8          /* Les destinations ne sont pas encore connues.                    */
9      τ ← échantillon(fils(s));
10     si τ.s =? alors
11         s' ← s;
12         répéter
13             s' ← successeur(s');
14             /* Employer dans cette boucle un algorithme de détection d'un
                  éventuel cycle.                                              */
15             ;
16         jusqu'à s' non-déterministe ;
17         s ← s';
18         τ.s ← s;
19     sinon
20         s ← τ.s;
21     stop ←EstRécurrent(s);
22 jusqu'à stop ;
```

- de définir des états équivalents,
- d'exécuter deux instances du système suivant une *évolution identique* à des vitesses différentes. En dehors de la gestion des décisions des agents, il est nécessaire de prendre en compte tous les facteurs pouvant ajouter de l'indéterminisme dans l'exécution du système — par exemple un ordonnanceur stochastique qui donne une probabilité d'activation à chaque agent à chaque itération ou qui active les agents dans un ordre différent à chaque itération.

5.3.5 Problèmes pratiques et performances

Resynchronisation du lièvre et de la tortue — Le temps de calcul nécessaire à faire évoluer le système ayant triplé (calcul de deux itérations du lièvre pour chaque itération de la tortue), il peut être intéressant de resynchroniser la tortue sur le lièvre pour accélérer le processus de détection. En effet, si le lièvre a convergé après n itérations (voir figure 5.10), il faudra attendre n itérations supplémentaires avant que la tortue ne le rattrape et que la convergence soit détectée.

Le temps séparant le lièvre et la tortue augmente avec le temps écoulé depuis le début de la simulation. Ainsi si le lièvre converge après une heure de simulation, il faudra attendre une



Algorithme 14 : Test de récurrence du sous-graphe contenant le nœud courant

```
1  EstRécurrent(s);
2  S_vus ← ∅;
3  retourner TestRécursif(s);
   ---------------------------------------------
4  TestRécursif(s);
5  si s ∈ S_vus alors retourner vrai
6  S_vus ← S_vus ⋃ {s};
7  pour chaque τ ∈ fils(s) faire
8      si τ.s =? alors retourner faux;
9      si ¬TestRécursif(τ.s) alors retourner faux;
10 retourner vrai;
```

heure supplémentaire pour que la convergence soit détectée.

Nous proposons donc de resynchroniser régulièrement (toutes les n itérations) la tortue sur le lièvre, c'est-à-dire de recommencer la simulation à partir de l'état courant du lièvre. Cette méthode présente cependant deux inconvénients :

- Une décorrélation de la date de convergence et de la date de détection de la convergence. En effet, si la resynchronisation des deux séquences se fait t itérations après la convergence du lièvre, la détection aura lieu au mieux à la date $t + longueur\_du\_cycle$. Si la perte de précision de la date de détection est problématique, il est préférable de ne pas utiliser ce *hack*.
- Ensuite — pour une resynchronisation intervenant toutes les n itérations — il devient impossible de détecter un cycle de longueur supérieure à n, la resynchronisation s'effectuant avant que la tortue ne puisse rattraper le lièvre.

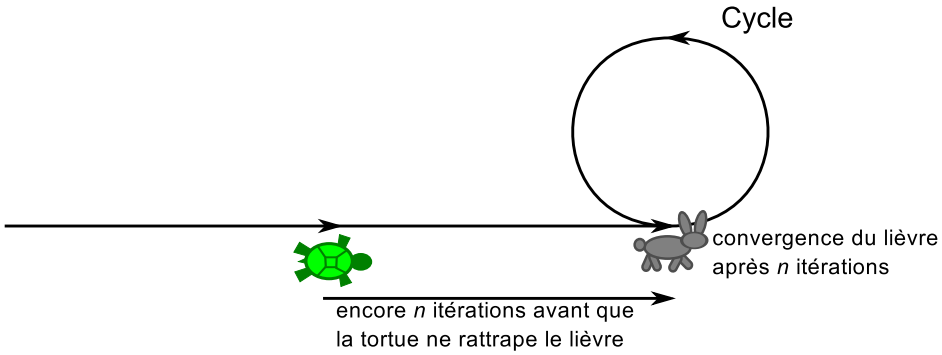


FIGURE 5.10 – Etat d'avancement des deux séquences lorsque le lièvre a convergé vers un cycle

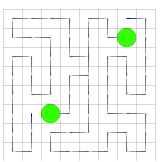
Profondeur de recherche et espace mémoire — Une autre préoccupation importante concerne l'espace mémoire utilisé lors de la construction du sous-graphe récurrent. Prenons l'exemple d'EVAP qui peut converger à des horizons importants selon les paramètres de simulation (le chapitre 6 est, entre autres, consacré à l'étude expérimentale de la convergence

d'EVAP. Selon la taille et la topologie de l'environnement ainsi que le nombre d'agents présents, cet horizon peut se situer à plusieurs millions d'itérations). Faisons ensuite l'hypothèse qu'un sous-graphe transient ait été détecté par l'algorithme du lièvre et de la tortue après quelques milliers d'itérations. La construction du sous-graphe récurrent peut ainsi s'étaler sur des millions d'itérations, l'algorithme de détection gardant en mémoire un grand nombre d'états. Afin d'éviter un encombrement de la mémoire, nous proposons d'arrêter la construction du graphe si une profondeur maximale de recherche est atteinte. Le graphe est alors effacé et la recherche — de cycle, avec l'algorithme du lièvre et de la tortue — est reprise à partir de l'état courant.

Un effet secondaire de ce *hack* est l'impossibilité de détecter un sous-graphe récurrent de taille supérieure à la profondeur de recherche maximale. Notons cependant que cette limitation apparaît également quand le système converge vers un sous-graphe d'une taille mémoire supérieure à celle disponible pour l'algorithme de détection.

Mise en application

L'algorithme de détection de comportements cycliques dans des séquences non déterministes — étendant l'algorithme du lièvre et de la tortue de Floyd — que nous avons présenté dans cette section est utilisé extensivement pour l'étude quantitative de la convergence d'EVAP vers des comportements cycliques présentée au chapitre suivant.



Chapitre 6

Etude de l'auto-organisation en cycles

Sommaire

6.1 Stabilité et instabilité des cycles d'états du système 98
6.1.1 Les sous-graphes d'un point de vue agents 98
6.1.2 Instabilité des cycles de longueurs différente 99
6.1.3 Stabilité des cycles agent de même longueur 101
6.2 Améliorer la convergence vers des cycles 105
6.2.1 Visualisation 105
6.2.2 Des motifs particuliers 105
6.2.3 EVAW+ : réduire les temps de convergence 107
6.3 Étude expérimentale 109
6.3.1 Cadre expérimental 109
6.3.2 Résultats avec des grilles simples 109
6.3.3 Topologies irrégulières 113
6.3.4 Oisiveté des cycles 114
6.3.5 Robustesse des comportements cycliques 115
6.4 Conclusion sur l'auto-organisation en cycles 115

Nous avons prouvé dans le chapitre précédent qu'EVAP converge systématiquement vers des sous-graphes récurrents (nous rappelons que les cycles sont un cas particulier de sous-graphes récurrents). Ces preuves ne donnent cependant aucune information sur l'intérêt de ces comportements cycliques pour la patrouille. Nous proposons dans ce chapitre de nous intéresser à ce problème.

Nous prouvons d'abord que lors de la convergence du système vers des cycles distincts, ceux-ci sont de même longueur. Cette propriété garantie une bonne répartition de la charge de travail entre les agents, c'est-à-dire une fréquence de visite homogène des nœuds de l'environnement.

Nous avons observé expérimentalement que le temps de convergence vers des cycles est important et explose avec l'augmentation de la taille de l'environnement et du nombre d'agents. Nous proposons une amélioration du comportement des agents accélérant significativement la convergence du système.

Nous concluons enfin ce chapitre par une étude expérimentale de la convergence vers des comportements cycliques afin de mesurer le temps de convergence, les longueurs des cycles (afin d'évaluer leur oisiveté) ainsi que les taux de cycles (hamiltoniens et non hamiltoniens) et de sous-graphes récurrents obtenus.

6.1 Stabilité et instabilité des cycles d'états du système

6.1.1 Les sous-graphes d'un point de vue agents

La représentation des cycles et sous-graphes récurrents sous forme de séquences d'états du système, si elle est utile pour décrire formellement ces structures, ne permet pas d'évaluer leur intérêt pour la patrouille (c'est-à-dire l'oisiveté du système après la convergence du système vers un sous-graphe). En effet, la seule information accessible sur la qualité de la patrouille à partir d'un sous-graphe d'états du système concerne sa longueur. Si en première approximation il peut sembler logique que les cycles d'états du système les plus courts sont les plus efficaces, nous pouvons montrer à travers un exemple simple que ce n'est pas toujours vrai.

Prenons l'exemple d'un environnement grille de 8x8 cellules patrouillé par deux agents. Le cycle d'états du système le plus court pouvant être obtenu est de longueur 32 itérations et présente des performances de patrouille optimales (voir figure 6.1.a). Pour autant, un cycle d'états du système de 64 itérations (figure 6.1.b), donc deux fois plus long, peut présenter la même oisiveté (optimale) que le cycle d'états du système de longueur 32.

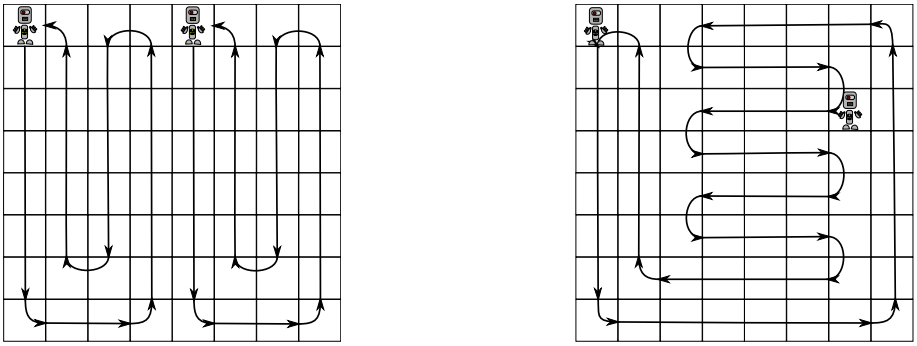
En effet, durant le cycle d'états du système de 32 états (ou 32 itérations de l'algorithme), chaque cellule sera visitée exactement une fois. Durant le cycle d'états du système de 64 états, chaque cellule sera visitée exactement deux fois (le cycle se termine quand les agents sont revenus à leur *places respectives*), c'est-à-dire avec une fréquence de visite égale au cycle de 32 états. Les informations fournies par un *cycle d'états* ne permettent donc pas de juger de sa *qualité* en termes d'*oisiveté* et donc de son efficacité pour la patrouille.

C'est pourquoi nous devons prendre en compte un point de vue centré agent afin de pouvoir étudier théoriquement la qualité des cycles et sous-graphes récurrents.

Nous définissons un *cycle agent* comme la séquence de nœuds visitée par un agent au cours d'un cycle d'états du système. Nous définissons également un *sous-graphe récurrent agent* comme l'ensemble des séquences de nœuds pouvant être suivies par un agent entre deux états non déterministes d'un sous-graphe récurrent.

Dans les deux cas, ils s'agit de séquences de nœuds visités par les agents ; pour les cycles d'états du système, il s'agira de séquences cycliques de nœuds et pour les sous-graphes récurrents, de séquences de nœuds visités par les agents entre deux états non déterministes du système.

Nous utilisons, dans cette section, cette représentation individu centrée des cycles pour étudier la stabilité et la structure spatiale des cycles d'états du système.



a. cycle de 32 itérations b. cycle de 64 itérations

FIGURE 6.1 – Deux cycles optimaux pour les mêmes paramètres de simulation : a - cycle de 32 états, b - cycle de 64 états

6.1.2 Instabilité des cycles de longueurs différente

Nous avons observé lors des expérimentations que, dans le cadre multi-agent, tous les cycles d'état du système ne sont pas stables (c'est-à-dire que le système est en fait dans un sous-graphe transient dont la structure spatiale est semblable à un cycle d'états du système). C'est notamment le cas lorsque le système converge vers de multiples *cycles agent* distincts [42], patrouillés chacun par un agent et dont les longueurs sont différentes.

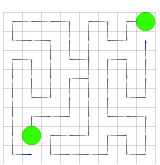
Ce comportement est particulièrement intéressant lorsqu'il s'agit de patrouiller un environnement avec plusieurs agents. En effet, des cycles agent de même longueur garantissent une répartition spatiale homogène des agents. On évite ainsi des situations dans lesquelles un sous-ensemble réduit de nœuds de l'environnement est visité avec une fréquence nettement supérieure au reste de l'environnement. Nous rappelons au lecteur qu'une fréquence de visite homogène est un critère d'optimalité de la patrouille (voir section 2.3.2).

Nous cherchons à démontrer dans cette section que seuls les cycles agent de longueurs égales peuvent persister, c'est-à-dire que lorsque les cycles agents de longueurs différentes le système est dans un sous-graphe transient . Les cycles agent sont en interaction deux à deux. Nous considérons donc seulement le cas de 2 cycles. Nous nous plaçons dans le cadre formel d'EVAW et considérons dans cette section la notation $m_t(v_i)$ comme la valeur de marquage du nœud v_i au temps t.

Soit deux cycles agent distincts ζ_1 et ζ_2 de longueur l_1 et l_2. Nous faisons l'hypothèse que les cycles ζ_1 et ζ_2 sont patrouillés respectivement par les agents a_1 et a_2. Ces cycles sont connectés par au moins un couple de nœuds (v_1, v_2).

Nous supposons que $l_1 < l_2$. A chaque visite de v_1, a_1 continue son cycle vers le nœud v'_1 (voir figure 6.2), ce qui définit un sens de rotation de l'agent a_1. Le sens de rotation de l'agent. Nous faisons l'hypothèse que $v'1$ n'est visitée qu'une seule fois au cours du cycle (ce qui est vérifié en particulier dans le cadre des cycles hamiltoniens). En conséquence, au temps t, lorsque a_1 se trouve en v_1, nous avons :

42. Les cycles agent sont distincts lorsque qu'ils ne partagent pas de nœud commun



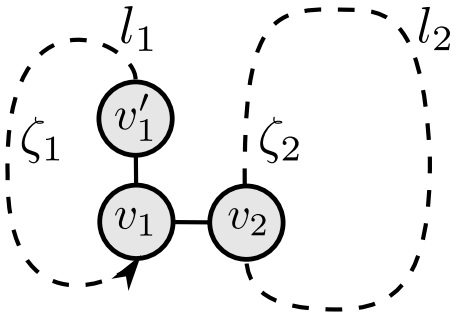


FIGURE 6.2 – Deux cycles distincts de longueur différentes connectés par les nœuds (v_1, v_2)

$$m_t(v'_1) = m_t(v_1) - l_1 + 1 = t - l_1 + 1. \tag{6.1}$$

Théorème 6.1.1 *Dans ces conditions, deux cycles agent distincts patrouillés chacun par un agent EVAP ne peuvent pas persister s'ils sont de longueur différente.*

Preuve: Si l'agent a_2 casse son cycle en premier, le problème est résolu. Considérons alors que ce n'est pas le cas et observons ce qu'il se passe pour l'agent a_1.

L'agent a_1 se déplace vers le cycle ζ_2 (déplacement vers le nœud v_2) si et seulement si il se trouve sur le nœud v_1 au temps t et

$$m_t(v'_1) \quad \geq \quad m_t(v_2). \tag{6.2}$$

Cette inégalité repose sur le comportement des agents EVAP qui garantie le déplacement de l'agent vers le nœud du voisinage de valeur minimale.

Nous devons donc montrer que l'inégalité (6.2) deviendra vraie en temps fini.

La propriété selon laquelle les deux agents visitent v_1 et v_2 alternativement et infiniment souvent s'écrirait :

$$t_2 \leq t_1 \leq t_2 + l_2 \leq t_1 + l_1 \leq \cdots \leq t_2 + k \cdot l_2 \leq t_1 + k \cdot l_1,$$

avec t_2 et t_1, deux *dates de visite de référence* telles que t_2 marque la première visite de v_2 par a_2 , t_1 la première visite de v_1 par a_1 et $t_2 < t_1$ (a_2 visite v_2 avant que a_1 ne visite v_1).

Cette inégalité ne peut être vraie que si $l_1 = l_2$, c'est-à-dire lorsque les cycles ζ_1 et ζ_2 sont de même longueur ($\forall l_1 \neq l_2, \forall t_2 \leq t_1, \exists k$ tel que $t_1 + k \cdot l_1 \leq t_2 + k \cdot l_2$).

Il existe donc deux dates t_1 et t_2 des visites de a_1 en v_1 (où $m_{t_1}(v_1) = t_1$) et de a_2 en v_2 (où $m_{t_2}(v_2) = t_2$) telles que

$$t_2 \leq t_1 < t_1 + l_1 < t_2 + l_2.$$

c'est-à-dire que l'agent a_1 passera deux fois par v_1 entre deux visites de v_2 par a_2.

A partir de l'équation 6.1, nous pouvons écrire :

$$\begin{aligned} m_{t_1}(v'_1) &= t_1 - l_1 + 1, \\ m_{t_1+l_1}(v'_1) &= (t_1 + l_1) - l_1 + 1 = t_1 + 1, \\ m_{t_1}(v_2) &= m_{t_2}(v_2) = t_2 \quad (\text{car } t_1 < t_2 + l_2) \text{ et} \\ m_{t_1+l_1}(v_2) &= m_{t_2}(v_2) = t_2 \quad (\text{car } t_1 + l_1 < t_2 + l_2). \end{aligned}$$

Alors, à l'instant $t_1 + l_1$, nous avons (en utilisant l'équation 6.2) :

$$\begin{aligned} m_{t_1+l_1}(v'_1) &= t_1 + 1 \\ &> t_2 \\ &= m_{t_1+l_1}(v_2). \end{aligned}$$

Dans ces conditions, l'agent a_1 saute sur le cycle ζ_2.

Notons que, comme nous ne prenons en compte que le nœud v_2 du cycle ζ_2, le résultat précédemment énoncé ne dépend pas du sens de rotation de l'agent a_2. Une seconde remarque concerne la stabilité d'un système de n cycles agent distincts patrouillés par n agents. La stabilité d'un tel système ne peut être obtenue que dans le cas où tous les cycles sont de même longueur.

6.1.3 Stabilité des cycles agent de même longueur

Nous nous plaçons dans les mêmes conditions que dans la section précédente (deux cycles distincts patrouillés chacun par un agent) mais en posant $l_1 = l_2$.

Tous les cycles (de même longueur) obtenus ne sont pas nécessairement stables. Nous montrons ici que certains motifs sont des points fixes du système mais que d'autres sont instables (des sous-graphes transients d'états du système).

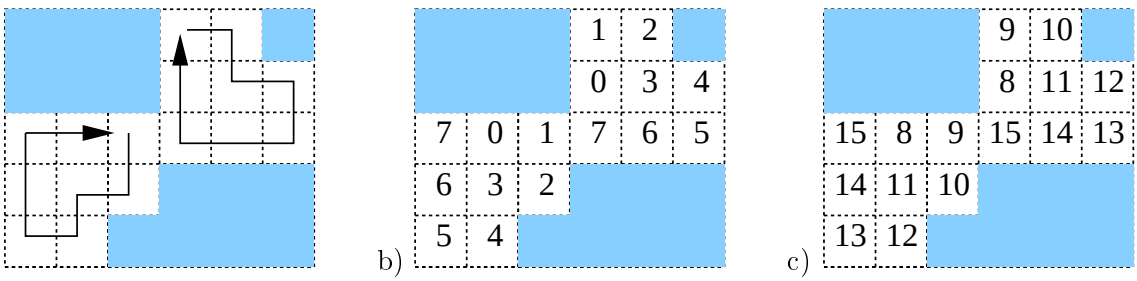
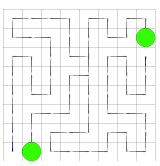


FIGURE 6.3 – Un cycle d'états du système composé de deux cycles de longueurs égales.

Commençons avec un exemple illustré. La figure 6.3 présente un environnement dans lequel deux cycles agent stables ont émergé. Notons au passage que ces cycles sont une solution optimale, chaque cellule étant visitée exactement une fois au cours du cycle.



La figure 6.3.b montre l'état du système à l'itération 7 et la figure 6.3.c l'état du système à l'itération 15 (soit un *tour* de cycle après la figure 6.3.b). Nous pouvons noter que la différence des marques entre les cellules adjacentes des deux cycles est constante (ce qui prouve la stabilité du système).

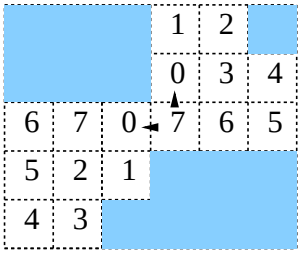


FIGURE 6.4 – Un cycle instable (transitoire) d'états du système composé de deux cycles de longueur égale

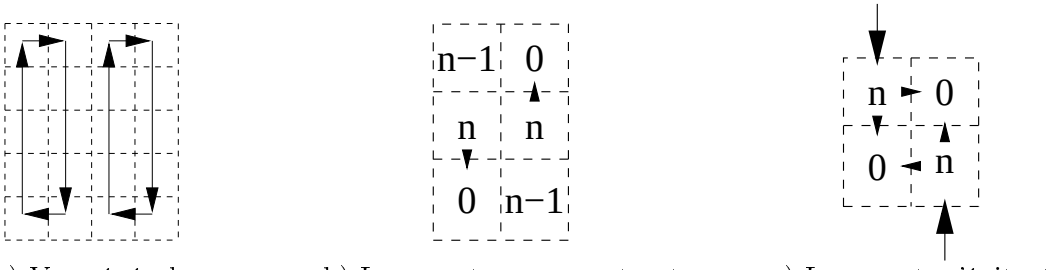
La figure 6.4 présente un cycle transitoire (un cycle instable). En effet, en considérant que l'agent du cycle de droite agit en premier, celui-ci a le choix entre suivre son propre cycle et se déplacer vers le cycle de l'autre agent (les deux cellules possèdent la même valeur de marquage), brisant ainsi la structure du champ de phéromones.

Nous allons exposer dans quelles situations un tel choix est possible (et donc dans quelles configurations ces cycles de longueurs égales ne sont pas stables). Nous commençons par étudier un cas particulier dans lequel les deux cycles sont adjacents sur la moitié de leur longueur (voir figures 6.5.a et 6.6.a). Nous distinguons deux cas selon les sens de rotation relatif des agents.

Agents de déplaçant dans des sens opposés — La longueur de la frontière correspondant à la moitié de la longueur de leurs cycles, a_1 et a_2 se rencontrent toujours à un même point précis de cette frontière. Selon l'environnement considéré (et donc la taille des cycles), les agent peuvent soit toujours se rencontrer sur un couple (c_1, c_2) de cellules (voir figure 6.5.b) — dans ce cas, ils restent chacun sur leur cycle —, soit toujours se "rater" (voir figure 6.5.c) — dans ce cas, ils voient chacun la "queue" du cycle [43] et peuvent donc choisir de "sauter" sur le cycle de l'autre agent. Notons que dans ce cas précis, les cycles ne sont pas cassés mais que le système se trouve en fait dans un graphe récurrent d'états du système (voir section5.2). Ainsi, dans le cas de deux cycles avec des agents se déplaçant dans des sens opposés, le système conservera toujours sa structure (sauf que dans un cas, cette structure sera un cycle d'états du système et dans l'autre, un graphe récurrent d'états du système).

Agents se déplaçant dans le même sens — Les agents tournent ici dans le même sens, ils se "suivent" donc le long de la frontière. Selon l'écart entre les deux agents (qui reste constant puisque les chemins qu'ils suivent sont de même longueur), le système réagira différemment. Si les agents sont distants de plus d'une cellule (figure 6.6.b), aucun des agents ne verra jamais la *queue* du cycle de l'autre. Dans ce cas, la structure du système reste stable.

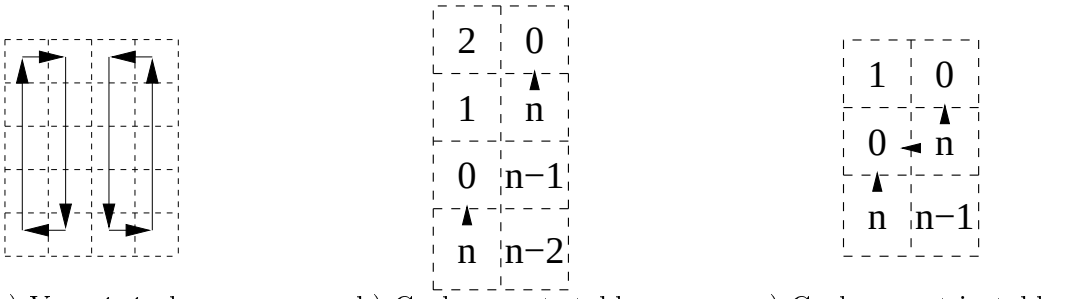
43. la queue du cycle est le nœud du cycle le plus anciennement visité. Il s'agit donc de la valeur la plus basse du cycle et par définition du comportement des agents, de leur prochaine destination.



a) Vue générale b) Les agents se rencontrent c) Les agents s'évitent

FIGURE 6.5 – Agents se déplaçant dans des sens opposés le long de la frontière entre leurs cycles

Si l'agent a_1 (l'agent activé en premier selon nos hypothèses de travail) a une cellule d'avance sur l'agent a_2 (voir figure 6.6.c), a_1 aura, à chaque itération, le choix de continuer sur son cycle ou de sauter sur le cycle de a_2, brisant ainsi la structure.



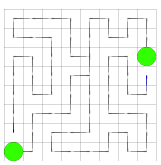
a) Vue générale b) Cycles agent stables c) Cycles agent instables

FIGURE 6.6 – Agents se déplaçant dans le même sens le long de la frontière entre leurs cycles

Cycles à frontières non continues — Le même raisonnement peut être appliqué à des situations plus complexes dans lesquelles la frontière est "découpée" en plusieurs segments. Il faut alors étudier les sens de rotation des agents ainsi que les distances les séparant pour chaque segment pour déterminer si la construction est stable.

a) b)

FIGURE 6.7 – Cycles d'états du système avec a) une frontière non continue et b) plus de deux agents



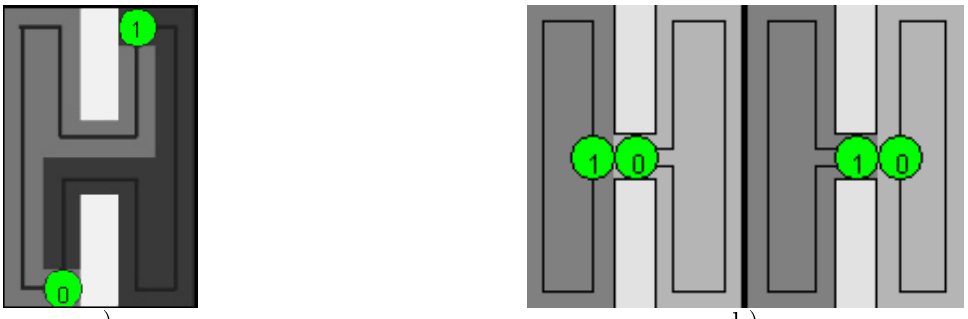
Au delà de deux agents — Le même raisonnement peut être appliqué aux situations mettant en jeu plus de deux agents. Il suffit de considérer les cycles deux à deux. Par exemple, sur la figure 6.7.b, il suffirait d'étudier les frontières entre les cycles des agents a_0 et a_1, a_0 et a_2 et a_1 et a_2.

Cycles partagés

Lors de l'étude de la stabilité des cycles de même longueur, nous avons également observé des cycles de plus complexes. Dans les exemples présentés jusqu'à présent, chaque cellule de l'environnement n'était visitée que par un unique agent. Nous proposons de montrer quelques exemples de cycles dans lesquels les agents se partagent la visite d'une ou plusieurs cellules.

Cycle commun — Un premier cas correspond à plusieurs agents patrouillant l'environnement sur un cycle commun (voir figure 6.8.a). Dans ce cas, il est évident que les agents décrivent des cycles de même longueur. Nous avons également observé expérimentalement que l'écart entre les agents se suivant est généralement égal (sauf dans le cas où le cycle est de longueur impaire). par exemple, sur un cycle de 16 cellules avec 4 agents, chaque agent sera séparé du suivant par 4 cellules.

Cycles à cellules partagées — Un second cas correspond à des cycles partageant seulement un sous-ensemble de cellules. La figure 6.8.b présente un exemple de cycle a cellules partagées. Dans ce cas, les agents semblent également toujours parcourir des cycles de même taille (observation expérimentale).



a) b)

FIGURE 6.8 – Cycles d'états du système avec a) un cycle commun aux deux agents and b) une cellule partagées par deux agents

Conclusion sur la stabilité des cycles de même longueur

Nous avons pu observer que les cycles produits par EVAP peuvent adopter une grande variété de formes mais que leur équilibre est précaire.

La stabilité de l'ensemble des topologies rencontrées dépend fortement des positions relatives des agents et seul un sous-ensemble très réduit des cycles sont stables. Il s'avère que

les cycles que nous avons pu observer expérimentalement sont de bonnes solutions pour la patrouille multi-agent (la fréquence de visite des cellules est homogène et l'oisiveté est proche de l'optimale). Toutefois, la rareté de ces structures peut être une des raisons à l'origine des temps de convergence importants observées expérimentalement dans le cadre multi-agent.

6.2 Améliorer la convergence vers des cycles

Cette section est motivée par l'explosion combinatoire du temps de convergence en fonction de la taille de l'environnement et du nombre d'agents. De façon à proposer un algorithme amélioré, nous introduisons d'abord un outil de visualisation approprié nous permettant d'analyser la formation de chemins. A partir de cette analyse, nous concevons une nouvelle heuristique qui repose sur la détection de motifs particuliers.

6.2.1 Visualisation

Un problème pratique dans l'étude de la formation des cycles produits par EVAP concerne la visualisation de l'état du système. En effet, la visualisation classique du champ de phéromones par gradient de couleur (figure 6.9.a) ne permet pas de visualiser efficacement le chemin parcouru par les agents. Nous cherchons donc une autre représentation, plus lisible, quitte à perdre certaines informations sur l'état courant.

Les marques numériques déposées par les fourmis forment un champ de potentiels. Une idée intuitive est de visualiser le champ vectoriel associé, c'est-à-dire de dessiner, pour chaque cellule c, des flèches dans les directions de *plus grande pente descendante*. Toutefois, comme illustré en figure 6.9.b ceci ne donne pas une représentation directe des chemins pris par les agents mais plutôt la direction que prendrait un agent présent sur une cellule donnée à cet instant.

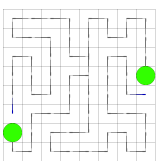
Nous proposons pour cette raison une nouvelle représentation. A chaque instant, nous dessinons des flèches entre une cellule c et ses voisines en suivant la *pente ascendante la plus faible* (Dans le cas, ou plusieurs destinations sont possibles, on dessine plusieurs flèches). Comme illustré en figure 6.9.c, le résultat exhibe clairement les chemins pris par les agents et les cycles quand ceux-ci se forment. Ce résultat vient du fait que la plus petite pente ascendante représente le chemin pris par les agents lors de la dernière visite d'une cellule (la pente générée par l'évaporation de la phéromone).

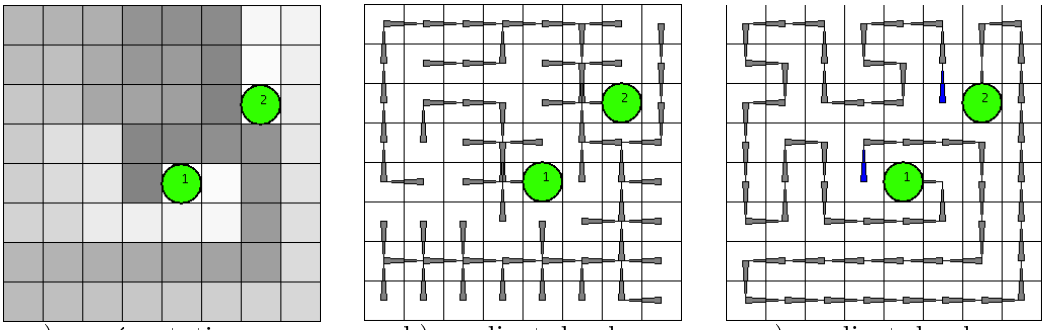
Nous utiliserons désormais cette représentation.

6.2.2 Des motifs particuliers

queues de cycles et sous chemins Hamiltoniens

On peut voir expérimentalement, et c'est plus visible sur de grands environnements, qu'EVAW construit des cycles à partir de chemins *hamiltoniens partiels* apparaissant pendant le processus de convergence. Nous nous focalisons sur les nœuds qui correspondent à des minima locaux





a) représentation par gradient de couleur

b) gradient de plus grande pente descendante

c) gradient de plus faible pente ascendante

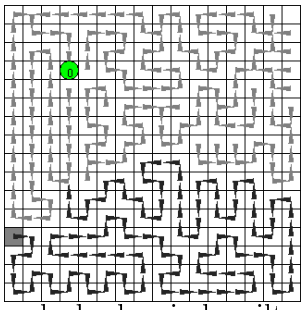
FIGURE 6.9 – Deux représentations à base de champs de vecteurs d'un même état du système

($m(queue) \leq \min_{v \in Vois(queue)} m(v)$) et que nous appelons *queue* puisqu'elles correspondent à la plupart des débuts de chemins observés.

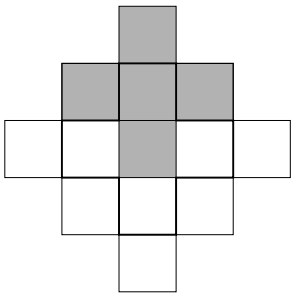
Une propriété intéressante est que, quand un agent entre dans un chemin existant par sa queue (cellule sombre sur la figure 6.10.a), il suit le chemin jusqu'au bout sans réorganiser le champ de phéromones, favorisant ainsi la convergence vers des cycles. Mais, du fait de leur connaissance limitée de l'environnement (perceptions limitées aux cellules voisines, pas de mémoire), les agents ne sont pas capables d'identifier ces chemins et :

- n'entrent pas nécessairement dans un chemin par sa queue quand c'est possible ;
- détruisent les chemins quand ils n'y entrent pas par leur queue.

S'il est difficile pour les agents d'identifier des chemins du fait de leur perceptions locales, ils peuvent facilement détecter une queue parmi ses voisins en utilisant une zone de perception étendue (13 cellules au lieu de 4 dans nos exemples sur des environnements grille, soit le voisinage de Moore étendu), comme illustré par la figure 6.10.b.



a. Un exemple de chemin hamiltonien partiel (représenté par les flèches sombres) ainsi que sa *queue de cycle* (représentée par la cellule sombre).



b. La croix grise représente le voisinage utilisé pour déterminer si la cellule au nord de l'agent est une queue ou non (voisinage de Moore étendu).

FIGURE 6.10 – Mise en évidence d'un chemin hamiltonien partiel et la zone de perception nécessaire pour détecter son entrée.

flip-flops

Les queues sont un premier type de motif intéressant qui apparaît quand des fourmis EVAW patrouillent. Quand les chemins suivis par les fourmis se stabilisent, on peut observer que la direction de sortie de certains nœuds reste instable. Nous appelons ce phénomène un *flip-flop*. La figure 6.11 montre une fourmi parcourant un cycle non-hamiltonien avec deux flip-flops. Dans le flip-flop de droite (situé sur la cellule dont le marquage passe de 6 à 14), la flèche sortante alternera entre les directions haut et gauche.

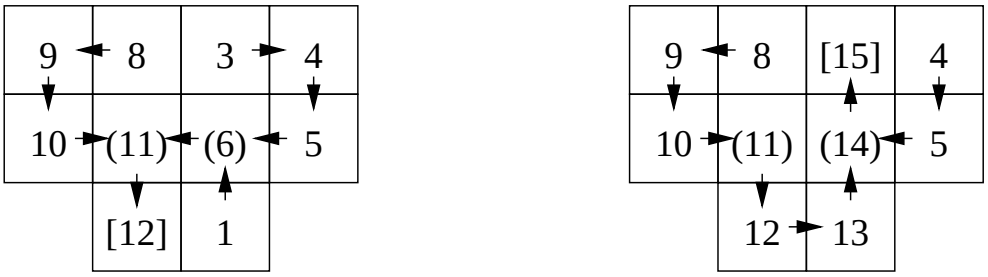


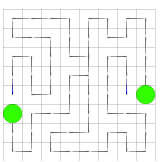
FIGURE 6.11 – Une fourmi (notée avec des crochets) parcourant un cycle non-hamiltonien avec deux flip-flops (notés par des parenthèses). Les figures montrent les pas de temps 12 et 15.

Les flip-flops sont des structures indésirables dans le cadre des environnements admettant des cycles hamiltoniens puisque leur présence implique la convergence du système vers un comportement cyclique sous-optimal.

Il serait alors tentant d'éliminer ces motifs pour permettre au système de re-converger vers une meilleure solution. Les flip-flops sont cependant **nécessaires** à la formation des cycles non-hamiltoniens puisqu'ils permettent la revisite de certains nœuds au cours d'un cycle. Chercher à supprimer ce motif impliquerait donc d'empêcher le système de converger vers une patrouille cyclique sur les environnements non-hamiltoniens. A titre d'information, la plupart des environnements présentés dans cette thèse n'admettent pas de cycles hamiltoniens et d'une manière générale, les environnement hamiltoniens sont beaucoup plus rares et se rencontrent rarement dans la réalité. De plus un environnement admettant un cycle hamiltonien pour n agents n'en admet pas nécessairement pour $n+1$ agents. Nous ne chercherons donc pas à empêcher l'apparition de flip-flops.

6.2.3 EVAW+ : réduire les temps de convergence

Sur la base de ces observations, nous étendons EVAW en forçant les agents à se déplacer vers une queue lorsqu'ils en détectent une. Cette variante — non limitée à des environnements grilles — est appelée EVAW+ et présentée dans l'algorithme 15, où $queues(Vois(v))$ est l'ensemble des nœuds voisins de v dans lesquelles on reconnaît le motif caractéristique d'une queue.



Algorithme 15 : Comportement d'un agent dans la version améliorée d'EVAW

```
1 t = t + 1;
2 si queues(Vois(v)) ≠ ∅ alors
3     v ← arg min_{d∈queues(Vois(v))} m(d) ; /* Règle de visite des queues de cycle */
4 sinon
5     v ← arg min_{d∈Vois(v)} m(d) ;                    /* Comportement normal */
6 m(v) ← t ;
```

Maintenir la propriété de patrouille

Si cette amélioration de l'algorithme EVAW accélère la convergence (voir section 6.3), elle peut conduire à la perte de la propriété de patrouille. En effet nous avons observé de rares cas dans lesquels, après quelques temps, certaines cellules de l'environnement ne sont plus visitées. On peut voir sur les figures 6.12 a), c) et e) que l'agent n'a pas d'autre choix — en appliquant l'algorithme 15 — que de suivre des queues de chemins (flèches foncées). En conséquence, la cellule du coin en haut à gauche et ses deux voisines ne peuvent plus être visitées.

Ce défaut est corrigé simplement en ajoutant une règle de plus haute priorité disant que, si la différence entre la valeur de la queue candidate et la valeur de la cellule voisine la plus âgée est plus grande qu'un seuil donné T_{thr} (par exemple $10 \times |V|$ pour éviter d'inhiber l'heuristique EVAW+, avec V le nombre de nœuds de l'environnement), alors l'agent doit se déplacer vers cette cellule la plus âgée. Avec cette modification, on peut à nouveau prouver que les agents EVAW+ couvrent leur environnement en temps borné (cf. section3.3) en utilisant $\Delta = T_{thr}$. Borner le temps de couverture d'EVAW+ permet alors d'étendre directement à EVAW+ l'ensemble des résultats théoriques établis précédemment sur EVAP et EVAW.

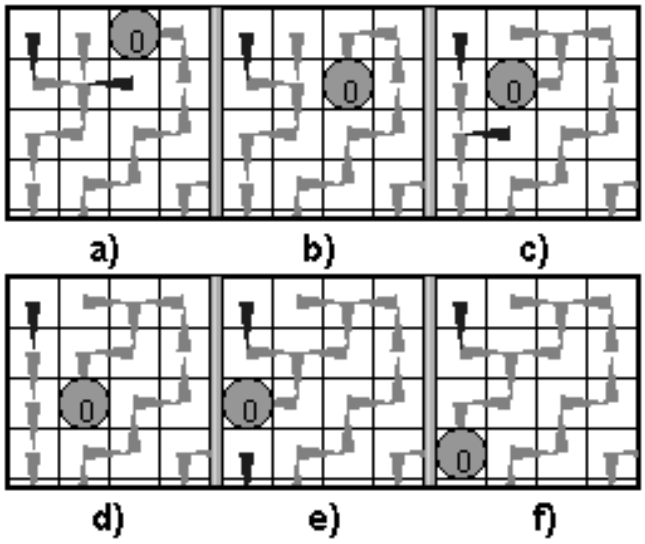


FIGURE 6.12 – Perte de la propriété de patrouille

6.3 Étude expérimentale

Cette section présente les expérimentations conduites pour illustrer les résultats théoriques vus en section 6.1 et pour étudier quantitativement le comportement de convergence du système, en mesurant le temps avant convergence et la probabilité de converger vers des cycles. Nous observons d'abord les effets de l'emploi d'un nombre croissant d'agents et d'une taille croissante de l'environnement, d'où l'utilisation d'une topologie de grille qui peut facilement passer à l'échelle. Nous regardons ensuite comment se comporte le phénomène de convergence dans le cas de grilles avec obstacles ou de topologies diverses.

6.3.1 Cadre expérimental

Avant de commencer cette étude, nous avons besoin de détailler comment *conflits* et *hésitations* sont gérés dans les implémentations d'EVAW et EVAW+ utilisées dans nos expérimentations.

D'abord, nos premières implémentations d'EVAW reposaient sur le framework de simulation Madkit/turtlekit [Michel *et al.*, 2005] avec l'ordonnancement par défaut de turtlekit qui contraint les agents à agir séquentiellement et toujours dans le même ordre (c'est-à-dire avec un ordonnancement séquentiel cyclique). Nous avons conservé cet ordonnancement (EVAP/EVAW étant maintenant implémentés en JAVA dans un simulateur maison qui n'est pas détaillé dans cette thèse). Les conflits sont résolus en donnant une priorité supérieure aux agents d'ordre supérieur.

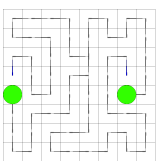
Ensuite, selon leur comportement en cas d'hésitation, nous distinguons deux types d'agents : 1) les agents non-déterministes qui agissent aléatoirement et 2) les agents déterministes qui préfèrent aller vers la droite plutôt qu'en arrière, à gauche et finalement en avant. Les premiers sont employés par défaut dans nos expérimentations (pour observer l'effet du non-déterminisme) alors que les seconds ne sont employés que pour vérifier certains résultats théoriques spécifiques.

La détection des comportements cycliques (cycles et sous-graphes récurrents) est réalisée à l'aide de la version étendu aux cas indéterministes de l'algorithme du lièvre et de la tortue présenté dans le chapitre précédent.

6.3.2 Résultats avec des grilles simples

Ici, nous considérons des grilles carrées sans obstacles. La plupart des expérimentations sont notées (n, s, α), où n est le nombre d'agents, $s \times s$ est la taille de la grille carrée vide et α dénote l'algorithme ('-'= EVAW, 'd'= EVAW déterministe, '+'= EVAW+). Au moins 1300 (et jusqu'à 10 000) simulations d'au plus 3×10^6 itérations ont été utilisées. Le positionnement initial des agents est aléatoire.

Nous avons effectué des expérimentations avec divers paramètres ($n \in \{1, 2, 3, 4, 5, 6, 8, 10\}$, $s \in \{8, 14\}$, $\alpha \in \{-, +, d\}$). Les tableaux 6.1 à 6.3 fournissent une partie des statistiques collectées.



Comportement de convergence

Notre premier ojbectif est de verifier expérimentalement les résultats établis en section 6.1 avec quatre paramétrages : $(1, 8, -)$, $(1, 8, d)$, $(3, 8, -)$ et $(3, 8, d)$. Comme prévu, les statistiques recueillies dans le tableau 6.1 montrent que le seul cas où la convergence vers des cycles n'est pas garanti (c'est-à-dire quand des sous-graphes récurrents sont trouvés) est celui où on emploie plusieurs agents non-déterministes ($(3, 8, +)$ et $(3, 8, -)$).

	# runs	cycles ham.	*lg.* (ham.)	cycles non-h.	$\overline{lg.}$ (non-h.)	graphes récurrents	non convergence	$\overline{tps\ de\ conv.}$	$\sigma(tps\ de\ conv.)$
1,8,d	20000	18,53 %	64	81,47 %	68	,00 %	,00 %	615	428
1,8,-	10000	26,79 %	64	73,21 %	66	,00 %	,00 %	596	401
1,8,+	10000	61,00 %	64	39,00 %	66	,00 %	,00 %	350	148
3,8,d	5000	,00 %	0	100,00 %	40	,00 %	,00 %	31283	30971
3,8,-	10000	,00 %	0	72,83 %	31	27,17 %	,00 %	71063	80072
3,8,+	10000	,00 %	0	80,83 %	84	13,31 %	5,86 %	3590	29000

TABLE 6.1 – Résumé des expérimentations sur la *déterminisation* d'EVAW.

	# runs	cycles ham.	*lg.* (ham.)	cycles non-h.	$\overline{lg.}$ (non-h.)	graphes récurrents	non convergence	$\overline{tps\ de\ conv.}$	$\sigma(tps\ de\ conv.)$
1,8,-	10000	26,79 %	64	73,21 %	66	,00 %	,00 %	596	401
1,8,+	10000	61,00 %	64	39,00 %	66	,00 %	,00 %	350	148
1,14,-	10000	10,06 %	196	89,94 %	211	,00 %	,00 %	116200	115605
1,14,+	10000	15,82 %	196	84,18 %	199	,00 %	,00 %	2264	1485
2,8,-	10000	34,08 %	32	53,16 %	56	12,76 %	,00 %	4356	5728
2,8,+	10000	24,81 %	32	61,24 %	68	13,95 %	,00 %	644	824
2,14,-	2800	1,78 %	98	69,57 %	186	2,25 %	26,39 %	1194667	884023
2,14,+	9992	5,41 %	98	88,03 %	197	6,05 %	,50 %	7219	70516
3,8,-	10000	,00 %	0	72,83 %	31	27,17 %	,00 %	71063	80072
3,8,+	10000	,00 %	0	80,83 %	84	13,31 %	5,86 %	3590	29000
3,14,-	1300	,00 %	0	2,38 %	184	,23 %	97,38 %	1869325	851906
3,14,+	9991	,00 %	0	90,64 %	186	6,44 %	2,91 %	7558	23258

TABLE 6.2 – Résumé des expérimentations conduites avec EVAW et EVAW+.

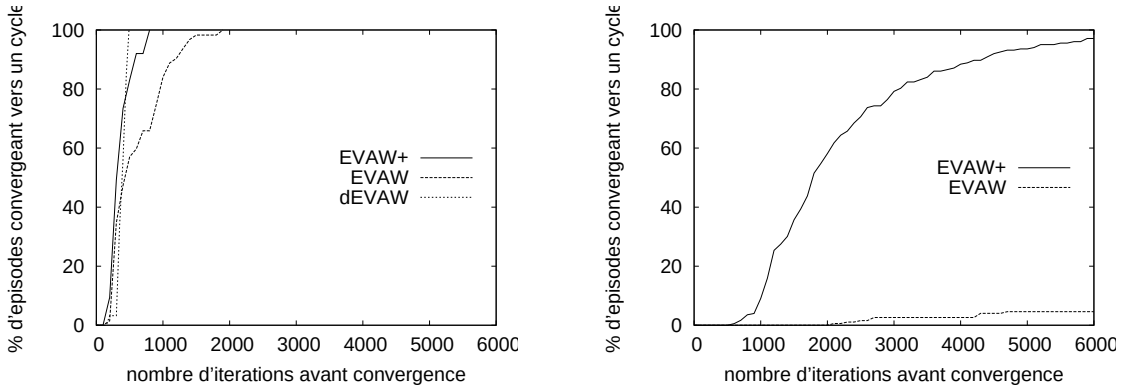
EVAW vs EVAW+

Nous considérons dans ce paragraphe les résultats de diverses expériences (voir tableau 6.2). Les figures 6.13, 6.14 et 6.15 représentent les courbes de convergence (la probabilité d'avoir convergé vers un cycle après N itérations) en ne tenant compte que des cycles, de sorte qu'elles n'atteignent pas toujours 100%.

On peut observer qu'EVAW+ converge toujours plus vite qu'EVAW, la tâche étant plus difficile s'il y a plus d'agents ou en cas d'environnements plus grands. En fait, il n'est pas possible d'expérimenter avec l'algorithme EVAW dans des cadres plus complexes.

	# runs	cycles ham.	$lg.$ (ham.)	cycles non-h.	$\overline{lg.}$ (non-h.)	graphes récurrents	non convergence	$\overline{tps\ de\ conv.}$	$\sigma(tps\ de\ conv.)$
1,8,+	10000	61,00 %	64	39,00 %	66	,00 %	,00 %	350	148
2,8,+	10000	24,81 %	32	61,24 %	68	13,95 %	,00 %	644	824
3,8,+	10000	,00 %	0	80,83 %	84	13,31 %	5,86 %	3590	29000
4,8,+	10000	13,27 %	16	38,99 %	43	47,74 %	,00 %	4204	4924
5,8,+	3100	,00 %	0	58,77 %	95	41,22 %	,00 %	252878	278600
6,8,+	450	,00 %	0	42,88 %	46	52,88 %	4,22 %	561121	485924
8,8,+	3450	9,73 %	8	2,11 %	19	88,14 %	,00 %	88195	85185
10,8,+	125	,00 %	0	,00 %	0	4,80 %	95,20 %	993875	517571

TABLE 6.3 – Résumé des expérimentations conduites avec EVAW+ sur une grille 8x8 et avec un nombre croissant d'agents.



a- sur une grille 8x8 b- sur une grille 14x14

FIGURE 6.13 – Courbes de convergence pour un agent ("vrais" cycles seulement)

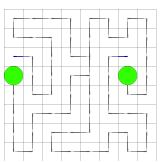
Forme des courbes de convergence

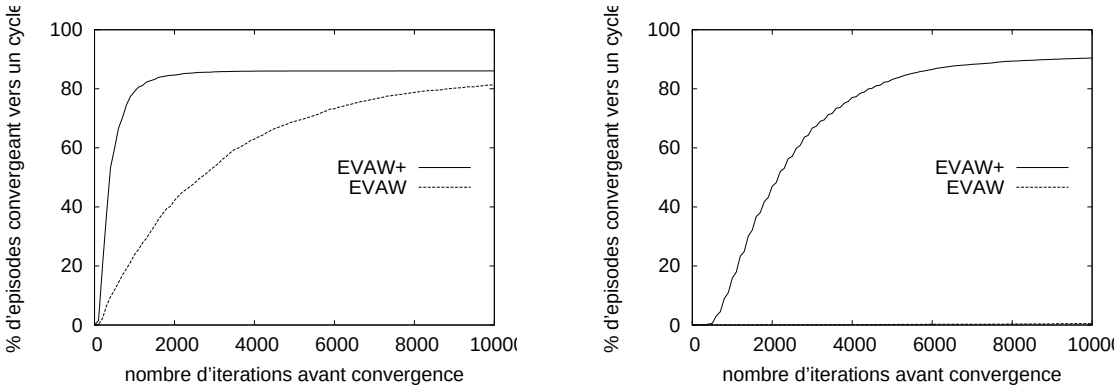
Comme on pouvait s'y attendre les diverses courbes de convergence ressemblent à des fonctions de répartition de distributions exponentielles. Cela signifie que, en moyenne, la probabilité de converger (de se retrouver sur un cycle) est la même à chaque instant. Elles diffèrent de ces fonctions de répartition

- au début, pendant la phase d'exploration initiale,
- du fait que des "marches" apparaissent sur les courbes (figures 6.13.b et 6.14.b), c'est-à-dire qu'il y a des périodes où le système converge rarement vers des cycles. Ce phénomène périodique reste inexpliqué pour l'instant.

Densité d'agents croissante

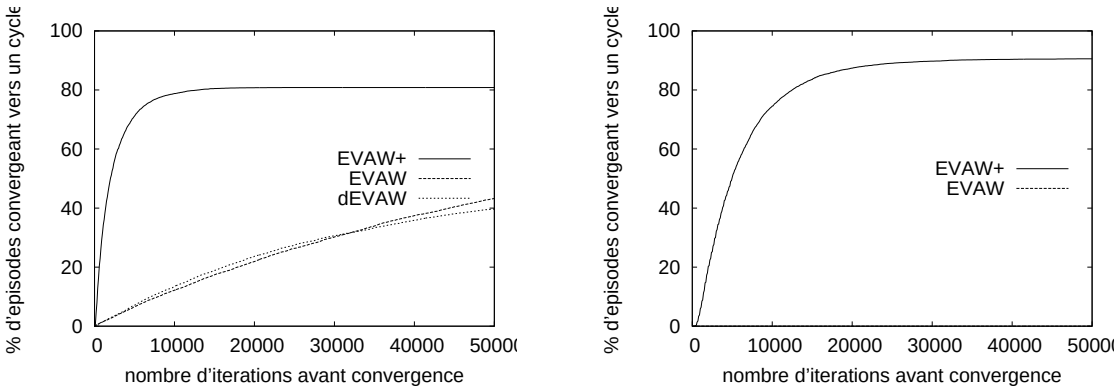
Finalement, nous avons mené des expérimentations avec un nombre croissant d'agents. Elles concernent seulement EVAW+ — parce que les autres algorithmes ne passent pas à l'échelle — utilisés sur une grille 8x8.





a- sur une grille 8x8 b- sur une grille 14x14

FIGURE 6.14 – Courbes de convergence pour deux agents ("vrais" cycles seulement)



a- sur une grille 8x8 b- sur une grille 14x14

FIGURE 6.15 – Courbes de convergence pour trois agents ("vrais" cycles seulement)

Les résultats sont présentés sur le tableau 6.3. L'observation principale est que aussi bien le pourcentage de convergence vers des sous-graphes récurrents que le temps de convergence augmentent avec le nombre d'agents, mais de manière irrégulière. Ces irrégularités coïncident clairement avec les situations où la taille de la grille n'est pas divisible par le nombre d'agents, c'est-à-dire des situations dans lesquelles l'environnement ne peut pas être facilement divisé en cycles de longueurs égales. Ceci force les agents à "trouver" des organisations plus compliquées comportant un ou plusieurs *flip-flops* permettant d'allonger artificiellement la longueur des cycles parcourus par certains agents. Typiquement, avec 3 agents sur un environnement de 64 cellules, nous pourrons trouver des cycles de longueur 22, visitant 66 cellules par cycle (22 cellules $\times$ 3 agents). Pour atteindre cette configuration, deux des trois agents sont obligés de revisiter deux fois une des cellules de leur cycle, *via* un *flip-flop*.

On peut noter que, dans les cas $(3, 8, +)$ et $(6, 8, +)$, EVAW+ peut ne pas converger dans le temps imparti. Ce phénomène est lié à l'algorithme de détection de sous-graphes récurrents. Pour ces expérimentations, nous avons limité la profondeur de recherche à 3 000 nœuds, restreignant la taille des sous-graphes récurrents détectables à 3 000 sommets. Ainsi, EVAW+ a effectivement convergé vers un sous-graphe récurrent mais l'algorithme de détection n'a pas été en mesure de le trouver. Afin d'estimer la taille des sous-graphes récurrents produits,

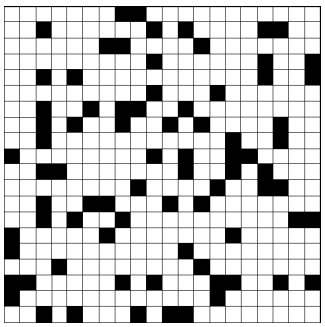
nous avons également lancé des expérimentations avec les mêmes paramètres expérimentaux mais avec des profondeurs croissantes allant jusqu'à 2 000 000 de noeuds. Nous observons effectivement une décroissance du nombre de non convergence avec l'augmentation de la profondeur de recherche, sans toutefois réussir à éliminer le phénomène. A titre d'exemple, pour les paramètres $(3, 8, +)$ et avec une profondeur de recherche de 500 000 noeuds, le taux de non convergence baisse pour atteindre 3,6%. Comme EVAW qui, dans des cas identiques, n'a pas produit de tels résultats (pour $(3, 8, -)$, voir tableau 6.2), nous pouvons supposer que l'heuristique a de fortes chances de produire des sous-graphes récurrents de très grande taille dans certaines situations.

Décroissance du nombre de cycles hamiltoniens

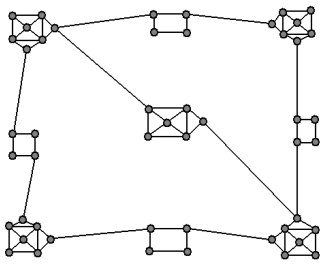
Finalement, on peut aussi observer que la probabilité de converger vers des cycles hamiltoniens décroît quand la taille de la grille croît. Nous ne savons pas si cela reflète une propriété de l'environnement — par exemple qu'il existe moins de cycles hamiltoniens parmi les sous-graphes récurrents possibles — ou une propriété d'EVAW — que son comportement d'exploration est moins susceptible de trouver des cycles hamiltoniens du fait des interférences entre agents qui ont tendance à détruire ou raccourcir les chemins hamiltoniens partiels créés précédemment.

6.3.3 Topologies irrégulières

Nous nous intéressons ici au comportement d'EVAW sur des environnements de formes plus diverses tirés ou adaptés de la littérature afin d'apprécier l'influence de topologies irrégulières sur le comportement de convergence de l'algorithme.



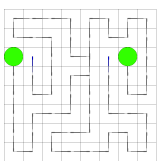
a) Environnement *crosswords*



b) Environnement *islands*

FIGURE 6.16 – Deux environnements irréguliers

Nous avons mené des expérimentations avec un nombre d'agents croissant (1,4 et 8 agents), avec et sans l'heuristique, sur les environnements *islands* (Fig. 6.16-a [Almeida *et al.*, 2004]) et *crosswords* (Fig. 6.16-b [Chu *et al.*, 2007]), et selon les mêmes hypothèses d'ordonnancement et de durée que pour les expérimentations précédentes. Nous avons observé que ces environnements se révèlent particulièrement problématiques. En effet, l'environnement *densité* n'a pas permis à EVAW de converger vers un cycle dans le temps imparti et nous n'avons pu observer de cycle pour un nombre d'agents supérieur à 3 sur l'environnement *islands* qui ne comporte



pourtant que 50 nœuds. L'efficacité, en termes de temps de convergence vers des cycles, de l'algorithme EVAW dépend donc fortement de la topologie de l'environnement.

Dans certains cas l'heuristique n'apporte aucun gain concernant le temps de convergence, sans pour autant dégrader les performances de la patrouille. Ainsi, la figure 6.17 présente une comparaison en termes d'oisiveté entre EVAW et EVAW+. Les courbes d'IGI — *Instantaneous Graph Idleness* : oisiveté moyenne du graphe à un instant donné — et de WGI — *Worst Graph Idleness* : pire oisiveté sur le graphe à un instant donné — représentant la qualité de patrouille ne semblent pas varier de manière significative entre les deux versions de l'algorithme.

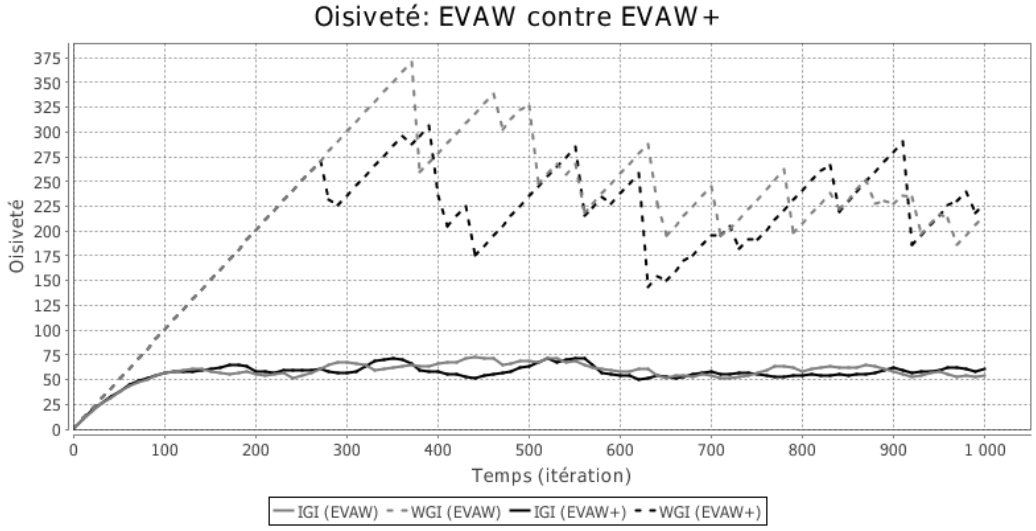


FIGURE 6.17 – Influence de l'heuristique EVAW+ sur la qualité de la patrouille

6.3.4 Oisiveté des cycles

L'utilité des structures cycliques dans la cadre de la patrouille dépend directement leur capacité à maintenir une oisiveté basse. des deux types de structures cycliques que nous avons étudié, seuls les cycles sont facilement évaluables. La nature stochastique des sous-graphes récurrents ne permet pas de leur associer une oisiveté fixe. Nous avons néanmoins pu observer expérimentalement que les sous-graphes récurrents de petite taille correspondentà des cycles avec des points d'indéterminisme permettant aux agents de passer d'un cycle à l'autre. Nous pouvons donc légitimement étendre les résultats suivants aux sous-graphe récurrents les plus courants.

La table 6.4 présente les moyennes des IGI observées [44] sur 1000 répétitions de l'expérience pour chaque paramètre.

Nous pouvons observer que les IGI sont en moyenne très proches de la borne d'optimalité théorique qui n'est pas atteignable dans les cas $3, 8, +$ et $3, 14, +$. Le plus mauvais cycle dans chacun des trois cas permet au système d'atteindre une IGI bien meilleure que pendant la

44. L'IGI des cycles pouvant être instable légèrement sur la durée d'un cycle, du fait des flips-flops apparaissant dans les cycles non-hamiltoniens, nous présentons la moyenne des IGI sur la durée du cycle.

	Oisiveté opti.	Oisiveté max.	Rapport opti./max.	Oisiveté moy.
3,8,+	10,16666	11,51562	+13%	10,31682
3,14,+	32,16666	34,22418	+6%	32,47514
4,14,+	24	25,05214	+4%	24,17038

TABLE 6.4 – Oisiveté moyenne (IGI) des cycles.

phase précédant la convergence (où l'IGI se stabilise à environ 25% au dessus de l'optimale théorique).

Notons également que la longueur des cycles d'états du système n'est pas un facteur déterminant quant à leur qualité. Ainsi, sur l'expérience $3, 8, +$, le cycle le plus long, d'une durée de 462 itérations (sur un environnement de 64 nœuds), atteint une IGI de 10,17, soit moins que la moyenne de l'IGI de l'ensemble des cycles détectés sur les 1000 répétitions de l'expérience. Le plus mauvais cycle détecté sur cette expérience, donne une IGI de 11,52 pour une durée de 32 itérations (un des cycles les plus courts ayant été détecté).

6.3.5 Robustesse des comportements cycliques

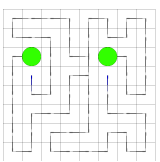
Un aspect important lorsqu'on s'intéresse à ces structure concerne leur robustesse aux perturbations. Il s'agit là d'un gros problème pour EVAW (et EVAP, dans leur versions normales et avec heuristique) puisque la moindre erreur de la part d'un agent (qui visiterait une mauvaise cellule) déstabilise l'organisation en cycle. L'agent brisera son cycle et ira rapidement visiter les nœuds des cycles des autres agents, renvoyant le système vers une phase de réorganisation similaire à la phase précédent la convergence.

Dans le cas général, nous ne notons pas d'accélération de la convergence vers un nouveau cycle/sous-graphe récurrent à partir des traces du cycle précédent par rapport à une nouvelle simulation.

6.4 Conclusion sur l'auto-organisation en cycles

Le comportement d'auto-organisation en sous-graphes récurrents est particulièrement intéressant pour la tâche de patrouille puisqu'il permet d'obtenir une patrouille dont la stabilité est garantie et avec une oisiveté très basse sinon optimale. Notons aussi que si les expérimentations menées dans ce chapitre ont été réalisées sur des environnements grille, ces résultats ne sont pas limités à cette topologie particulière et s'applique sur tous types de pavages réguliers, sur des graphes généraux et même sur des topologies évolutives, comme un réseau ou un site web dont les relations entre éléments sont amenées à changer au cours du temps (les cycles seront rendus instables par ces changements mais l'algorithme se réorganisera de lui même vers une nouvelle structure cyclique).

L'identification des motifs particuliers dans le champ de phéromones nous a également permis de proposer une amélioration du comportement des agents afin de favoriser la construction de chemins hamiltoniens partiels. Ces "portions de cycles" permettent une augmentation substantielle de la vitesse de convergence vers des cycles et permet d'observer leur formation sur des instances trop complexes pour la version originale d'EVAP.



L'utilisation des cycles dans le domaine robotique reste cependant difficile puisque le temps nécessaire à la convergence combiné à la sensibilité au bruit des structures cycliques ne permet pas leur exploitation dans ce cadre. Notons cependant que d'une part, d'après l'étude menée au chapitre 4, EVAP propose une patrouille robuste, aux performances acceptables et capable d'opérer sur des environnements de très grande taille avec un nombre d'agents conséquent. D'autre part, les cycles trouvés en simulation peuvent servir de plan de patrouille à une flottille d'agents (il faudrait alors comparer la qualité des solutions et le temps nécessaire à la convergence vers des cycles d'EVAP à d'autre approches) ou être utilisés pour initialiser le champ de phéromones avant de commencer la patrouille. Dans ce cas là, la patrouille suivrait le cycle jusqu'à ce qu'une perturbation survienne et conduise le système à se réorganiser.

Chapitre 7

Influence des modèles d'exécution dans l'étude de systèmes auto-organisés

Sommaire

7.1 Modèles d'exécution 119
7.1.1 Gestion de l'indéterminisme 119
7.1.2 Représentation et granularité du temps 120
7.1.3 Hypothèses d'ordonnancement 121
7.2 Implémentation robotique 122
7.3 Effet des modifications du modèle d'exécution 124
7.3.1 Hypothèses de simulation 124
7.3.2 Propriété de patrouille 124
7.3.3 Auto-organisation en cycles 125
7.3.4 Des robots et des cycles 127
7.4 Conclusion 128
1 Synthèse des contributions 131
2 Perspectives 134
2.1 Amélioration du comportement exploratoire d'EVAP 134
2.2 Extension d'EVAP à d'autres problèmes 134
2.3 Algorithmes fourmi et programmation dynamique : des phéromones pour apprendre 134
2.4 Résolution de problèmes par des approches auto-organisées . . . 135

Quand nous parlons de systèmes multi-agent situés, les différents aspects relatifs à son étude, à savoir, les aspects théorique, expérimental et robotique, sont habituellement considérés de manière indépendante. Cependant, chaque aspect ayant ses hypothèses et contraintes

propres, il n'est pas évident que des résultats théoriques puissent être transposés à une implémentation en simulation ou sur des robots. En effet, l'utilisation d'hypothèses d'exécution légèrement différentes peuvent mener, de par la nature complexe des systèmes étudiés, vers des comportements imprévus.

Alors que les systèmes multi-agent sont implicitement construits sur les interactions entre agents, les travaux réalisés sur ces approches ignorent habituellement la problématique des modèles d'exécution. Nous considérons les modèles d'exécution comme l'ensemble des paramètres des lois d'évolution du système. La manière de gérer l'ordonnancement, les interactions entre agents ou la représentation du temps et de l'espace définissent un modèle d'exécution. Par exemple, [Axtell, 2001] montre l'impact des modèles d'exécution sur les résultats de simulation. Ainsi, en fonction du modèle simulé, la modification de la fonction d'activation des agents (l'ordre des actions des agents) mène à des résultats statistiquement différents et à des comportements irréalistes (par exemple, en modifiant la fonction d'activation des agents, le même modèle de simulation d'un système économique produira soit un comportement proche de la réalité, soit un comportement complètement abérant). Le manque d'informations sur les modèles d'exécution (par exemple, dans les publications scientifiques) peut aussi empêcher la reproductibilité des expériences à cause des biais d'implémentation. Rectifier ces biais demande alors un réalignement du modèle [45] sur des données expérimentales déjà connues [Edmonds and Hales, 2003]. Le réalignement de modèles, en plus d'être fastidieux, ne garantit pas la fidélité du modèle réaligné au modèle original. Un modèle d'exécution donné peut en effet permettre d'obtenir les mêmes résultats sur les expérimentations connues (publiées) mais produire des comportements différents sur d'autres jeux de données.

Les algorithmes fourmi en environnement discret font généralement l'hypothèse d'une exécution synchrone des agents et ne spécifient pas comment résoudre les conflits entre agents. Cependant, les simulations reposent habituellement sur un ordonnancement asynchrone (les agents agissent, au cours d'une itération, les uns après les autres), souvent par ignorance du problème ou à cause des difficultés techniques à adapter le modèle à une exécution synchrone (en utilisant par exemple le modèle influence/réaction [Ferber and Muller, 1996] dans lequel il est nécessaire de distinguer la production d'influences (ce que souhaitent faire les agents) de la réaction de l'environnement (la réalisation des actions)). De plus, si de plus en plus de simulateurs multi-agent [Michel *et al.*, 2001] implémentent des ordonnanceurs synchrones, il proposent cependant souvent un ordonnanceur asynchrone par défaut.

Généralement au cours de l'implémentation d'un modèle conceptuel [46], les questions relatives aux modèles d'exécution sont résolues à partir des contraintes techniques de la plateforme sur laquelle le modèle va être implémenté ("Nous justifions le choix d'un ordonnancement asynchrone séquentiel parce qu'il est disponible sur le simulateur") plutôt que par un choix argumenté ("Nous justifions le choix d'un ordonnancement événementiel car c'est celui qui correspond le mieux à l'implémentation robotique que nous prévoyons de réaliser dans le futur").

Nous proposons dans ce chapitre, d'explorer l'influence des modèles d'exécution sur le

45. Lorsque l'on souhaite implémenter un modèle déjà publié dont le modèle d'exécution n'est pas donné, il est nécessaire d'effectuer un réalignement des modèles. On modifiera les paramètres d'exécution du modèle réimplementé pour que ses sorties correspondent aux données expérimentales (déjà connues) du modèle original.

46. Un modèle conceptuel décrit les composants du système sans spécifier ses lois d'évolution.

comportement des systèmes multi-agent réactifs discrets situés et plus particulièrement sur EVAP que nous avons étudié extensivement dans cette thèse, à la fois d'un point de vue théorique et expérimental.

Nous commençons par présenter les multiples éléments composant les modèles d'exécution dans les systèmes multi-agent réactifs en environnement discret. Dans un second temps, nous présentons la plateforme robotique utilisée pour l'implémentation du modèle EVAP.

7.1 Modèles d'exécution

L'implémentation d'un modèle multi-agent est soumis au choix d'un certain nombre de paramètres que nous regroupons sous le terme de *modèles d'exécution*. Cependant, les travaux sur les modèles multi-agent réactifs ne présentent généralement que le comportement des agents, considérant les hypothèses sur la résolution des interactions entre agents ou l'ordonnancement comme implicites ou comme des détails peu importants.

Prenons l'exemple d'EVAP. L'algorithme 7, présenté en chapitre 3 décrit effectivement le modèle mais ne donne aucune indication d'implémentation. Que doit-t-il se passer, par exemple, lorsque deux agent souhaitent se déplacer vers un même nœud ? En l'absence d'indications explicites (comme celles données dans l'algorithme 9), on effectue un choix au hasard. Ce choix pourra mener à une divergence entre l'implémentation du modèle étudiée et la réimplementation. Les agents risquent de montrer des comportements légèrement différents qui pourront mener à une organisation macroscopique très différente. Dans tous les cas, nous n'avons pas l'assurance que l'implémentation originale et la réimplémentation correspondent au même objet d'étude.

7.1.1 Gestion de l'indéterminisme

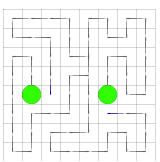
Le premier point important concerne la gestion de l'indéterminisme du comportement des agents qui peut être divisé en deux sous catégories. Ces deux catégories ont déjà été présentées au chapitre 6 pour la preuve de convergence vers des cycles. Nous les rappelons néanmoins dans un cadre plus général.

Hésitations

Le premier concerne les hésitations, c'est à dire les choix stochastiques qu'un agent peut-être amené à réaliser en cours d'exécution. Dans le cadre des systèmes multi-agent réactifs discrets, ils consistent généralement à choisir la prochaine destination de l'agent. Les hésitations faisant généralement explicitement partie du comportement des agents, elles ne posent pas de problèmes particuliers.

Interactions

[Weyns and Holvoet, 2003] proposent une classification complète des interactions pouvant être recontrées dans les systèmes multi-agent en général (les actions concurrentes, les actions



s'influençant indirectement et les actions jointes). Les capacités des agents réactifs dans les modèles discrets étant particulièrement limitées, nous pouvons réduire cette liste aux seuls interactions concurrentes directes : les conflits.

Les conflits proviennent des interactions directes entre agents lorsqu'ils doivent partager une ressource. Ces interactions n'étant pas présentes dans les algorithmes, la gestion des conflits est rarement explicitée.

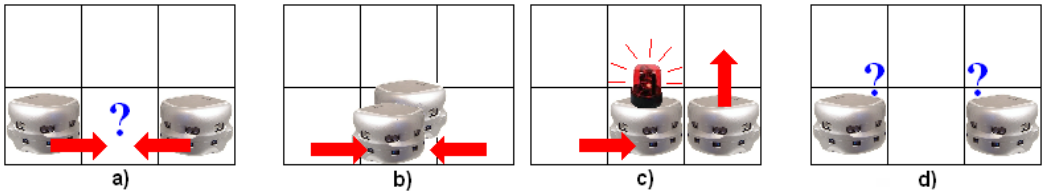


FIGURE 7.1 – Conflit entre agents pour la cellule centrale.

Prenons encore une fois l'exemple d'EVAP. Deux agent souhaitent visiter un même nœuds simultanément. Pour un tel conflit, ils peuvent :

- visiter le nœud ensemble (figure 7.1.b). Cette solution est clairement sous optimale et peut poser problème dans le cadre d'une implémentation physique.
- résoudre le conflit par un système de priorité (figure 7.1.c). Par exemple, un système de file d'attente (premier arrivé, premier servi), une priorité reposant sur leur position (comme la priorité à droite aux croisements), une priorité basée sur les identifiants des agents (priorité des identifiants les plus bas sur les identifiants les plus hauts) ou une combinaison de plusieurs règles lorsque l'une d'elle ne permet pas de régler seule les conflits (que fait on lorsque 4 voitures arrivent simultanément à un carrefour avec 4 priorités à droite ?).
- ne pas bouger et ne pas résoudre le conflit (figure 7.1.d), ce qui amènera le système à une situation de blocage si le comportement est déterministe.

Nous pouvons voir que la manière de gérer les conflits peut influer sur le comportement global du système, ce qui pose problème lorsqu'ils sont résolus implicitement par des contraintes d'implémentation.

7.1.2 Représentation et granularité du temps

Le deuxième aspect des modèles d'exécution auquel nous nous intéressons concerne la représentation du temps. Il s'agit d'un point à mettre en parallèle avec l'ordonnancement puisque le choix d'une représentation discrète ou continue du temps influe sur le choix de l'ordonnanceur.

Une des possibilités est de choisir une représentation continue du temps, ce qui implique le choix d'un ordonnancement événementiel. Cependant, dans le cadre des systèmes multi-agent réactifs discrets, ont considère généralement que le temps est divisé en unités de durées égales durant lesquelles chaque agent est capable d'effectuer une itération de sa boucle perception-décision-action.

Nous pouvons clairement voir que cela pose problème lorsque la durée de ces actions ne sont pas égales (que ce soit la différence de durée entre deux actions différentes, comme par

exemple un déplacement et un ramassage d'objets ou même la variabilité de durée d'une même action). D'une certaine façon, les agents sont alors synchronisés dans leur prise de décision, les agents dont les actions sont les plus courtes attendant ceux qui prennent plus de temps. Implémenter un tel système dans un cadre robotique demande alors de :

Implementer le système simulé sur les robots. Cette approche du problème suppose d'utiliser un processus de synchronisation des actions pour que les agents rapides ne commencent pas de nouvelles actions avant que les agents les plus lents aient fini les leurs. Une telle synchronisation, en plus de forcer l'introduction d'un processus centralisé dans le système, peuvent avoir un effet sur les performances, les agents rapides *attendant* les plus lents.

Adapter le modèle aux spécificités de l'implémentation robotique. Dans ce cas, les agents agissent dès que possible. Cependant, lorsque les durées des actions diffèrent, l'accumulation de ces dérives fait perdre au système sa synchronisation. Nous verrons plus loin (section 7.3.4) ce que peut impliquer un tel changement d'hypothèses.

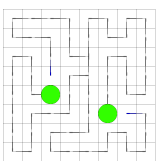
7.1.3 Hypothèses d'ordonnancement

Le choix d'activer les agents de façon synchrone ou séquentielle peut changer le comportement global du système et une fois de plus, les choix d'ordonnancement sont rarement spécifiés. De plus, alors que les systèmes multi-agent sous souvent considérés comme synchrones (parce que tous les agents agissent au cours de chaque itération), les simulateurs implémentent assez rarement de vrais ordonnanceur synchrones (qui séparent les phases de perception/décision et d'action des agents). Nous présentons ici les types d'ordonnancement les plus courants pour une représentation discrète du temps.

Approches synchrones

L'ordonnancement synchrone semble être un choix intelligent. Les interactions directes entre les agents (c'est-à-dire les conflits) ne se produisent que lorsque deux agents (ou plus) souhaitent réaliser des actions incompatibles au même moment. Une simulation synchrone nécessite donc d'identifier et de résoudre *explicitement* ce type d'interactions.

Ce type d'ordonnancement est cependant plus difficile à implémenter qu'un ordonnancement asynchrone. Le modèle Influence/Réaction [Ferber and Muller, 1996] explique comment approcher ce problème pour obtenir une simulation cohérente : Tout d'abord, tous les agents produisent des *influences*, c'est-à-dire qu'ils effectuent leurs décisions à partir de leurs perceptions mais sans essayer de les réaliser. Cette phase permet de s'assurer que les agents agissent tous à partir des mêmes perceptions. Dans un second temps, l'environnement effectue une phase de *réaction* qui consiste à combiner les influences des agents et à résoudre les éventuels conflits pour mettre à jour la simulation avant de passer à l'itération suivante. Il n'existe cependant pas de méthode générique pour détecter et résoudre les conflits, ce travail est laissé à la charge du concepteur du modèle.



Approches asynchrones

Il existe plusieurs types d'ordonnancements asynchrones. Nous présentons ici les plus courants.

L'α-synchronisme assigne aux agents une probabilité α d'activation à chaque itération. Ce type d'asynchronisme est principalement utilisé dans le cadre des automates cellulaires.

L'ordonnancement séquentiel active les agents chacun leur tour au cours de chaque itération. Cet ordonnancement peut sembler présenter une forme de synchronisme (un agent doit attendre que tous les autres agents aient effectué leur action avant de pouvoir agir à nouveau) mais il s'agit bien un ordonnancement asynchrone puisqu'il n'est pas possible pour deux agents de réaliser des actions concurrentes.

On peut distinguer deux sous-catégories d'ordonnancement séquentiels :

L'ordonnancement séquentiel cyclique où l'activation des agents se fait toujours dans le même ordre.

L'ordonnancement séquentiel acyclique où l'ordre d'activation des agents est aléatoire.

Les ordonnancements séquentiels présentent l'avantage (où le désavantage, si le concepteur du n'en est pas conscient) de résoudre implicitement la plupart des conflits puisque les accès concurrents à une ressource n'existent plus lors d'une exécution séquentielle. Le premier agent à agir "gagne" le conflit qui existerait avec un ordonnancement synchrone.

Nous pouvons aussi noter que, si les deux types d'ordonnancements séquentiels résolvent les conflits entre agents, l'ordonnancement séquentiel cyclique les résout de manière déterministe (l'ordre d'action étant fixé, le vainqueur dans un conflits enter deux agents donnés sera toujours le même) alors qu'il est impossible de prédire le résolution d'un conflit avec l'ordonnancement séquentiel acyclique.

7.2 Implémentation robotique

L'implémentation robotique des modèles multi-agent réactifs est un problème complexe. Plus précisément, les principales difficultés sont :

- Les robots sont autonomes, c'est-à-dire que leurs prises de décisions ne sont pas synchrones. L'utilisation d'un ordonnancement externe n'a pas de sens puisque l'activation des comportements des robots est régi par leur architecture interne (le processeur du robot). Les robots peuvent éventuellement être synchronisés mais dans ce cas, le système perd la propriété d'être décentralisé.
- Deux robots (du même type) ne peuvent pas être considérés comme des agents identiques. L'usure de certaines pièces ou la qualité variables des capteurs peut mener, pour un même comportement, à deux résultats légèrement différents. De plus, les robots évoluent dans un monde non déterministe.
- Les approches nécéssitant un environnement actif (capable de réaliser un calcul comme l'évaporation de phéromones par exemple) sont difficiles à implémenter du fait de la

nature même des processus considérés. Cependant, les avancées technologiques récentes permettent d'envisager l'implémentation de ces mécanismes à travers des environnements augmentés de réseaux de capteurs [Mamei and Zambonelli, 2005].

Dans ce chapitre, nous analysons les hypothèses d'exécution des modèles multi-agent réactifs discrets afin de pouvoir effectuer le passage de la simulation à l'implémentation robotique.

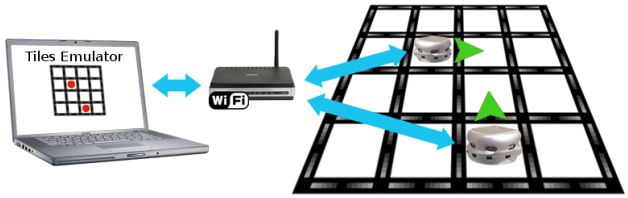


FIGURE 7.2 – Schéma de l'émulateur iTiles

Pour étudier l'implémentation robotique d'EVAP, nous nous penchons sur le modèle iTiles [Pépin *et al.*, 2009] (voir Figure 7.2) qui définit un moyen de marquer l'environnement. Ce modèle fait les hypothèses suivantes :

- L'environnement est pavé de dalles rectangulaires contenant chacune une unité de calcul autonome capable de réaliser des calculs simples et de communiquer avec les robots.
- Chaque dalle est connectée à ses quatre voisines, définissant un réseau de communication.
- Les robots peuvent lire et écrire de l'information sur la dalle sur laquelle ils se trouvent et peuvent propager ces informations à travers l'environnement via les dalles voisines.

Le modèle EVAP a été implémenté sur des robots Khépera 3 et un environnement reposant sur un émulateur de dalles. Les agents communiquent avec un serveur qui leur répond comme s'il s'agissait des dalles sur laquelle ils se trouvent. L'avantage d'un tel choix est que les robots conservent leur autonomie et ne sont pas synchronisés avec l'environnement actif. Une vidéo de cette implémentation du modèle EVAP est disponible à l'adresse "http ://www.loria.fr/∼simoniol/EVAP.html".

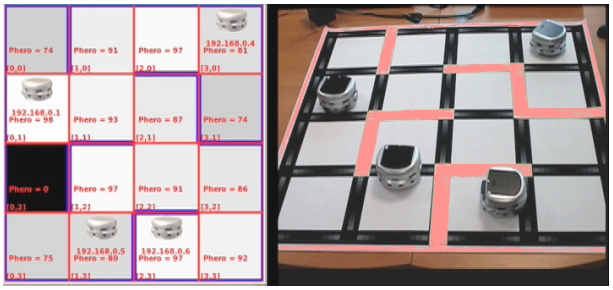
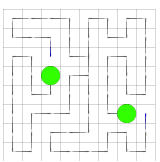


FIGURE 7.3 – Implémentation robotique d'EVAP sur l'émulateur de dalles



7.3 Effet des modifications du modèle d'exécution

Cette section est consacrée à l'étude de l'impact de différents modèles d'exécution (et de l'implémentation robotique) sur le comportement d'EVAP, en particulier sur deux propriétés majeures que nous avons démontrées dans les chapitres 3 et 6, la propriété de patrouille et la convergence vers des cycles stables.

7.3.1 Hypothèses de simulation

Comme expliqué en section 7.1, le comportement des agents seul ne permet pas de définir une implémentation. Nous précisons donc les hypothèses de simulation.

Ordonnancement : L'utilisation d'un ordonnancement séquentiel, cyclique ou acyclique, semble être un bon choix. En effet, la séquentialité des actions résout directement le problème des conflits dans EVAP. Considérons deux agents dans une situation de conflit dans laquelle ils souhaitent tous les deux accéder à un même nœud. Par définition du comportement des agents EVAP, après son déplacement, un agent marque son nœud d'arrivée. Ainsi, quand l'agent a_1 se déplace et marque le nœud conflictuel, l'agent a_2 choisira nécessairement une autre destination. Cet ordonnancement est donc particulièrement utile pour simuler une implémentation physique du modèle dans laquelle deux agents ne peuvent occuper la même position simultanément.

Transitions : Les transitions sont contraintes par l'ordonnancement. Nous considérons que chaque agent effectue un cycle perception/action par itération.

7.3.2 Propriété de patrouille

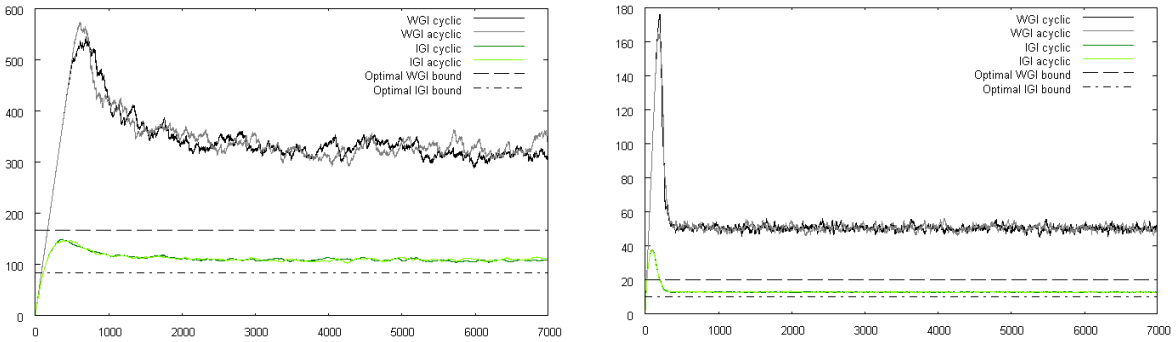
Nous rappelons que la propriété de patrouille garantit que tous les nœuds de l'environnement seront (re)visités infiniment souvent. Notons que cette démonstration est particulièrement intéressante puisqu'elle de ne fait aucune hypothèse sur l'ordonnancement, la résolution de conflits entre agents ou la forme de l'environnement.

Expérimentations en simulation

Hormis les transitions, ces hypothèses semblent être en accord avec celles de l'implémentation robotique présentée en section 7.2. Nous pouvons attendre une simulation assez précise de l'implémentation robotique.

Nous lançons nos expérimentation sur l'environnement *n-pièces* présenté en chapitre 4 avec 2 et 16 agents, une fois avec un ordonnanceur séquentiel cyclique et une fois avec un ordonnanceur séquentiel acyclique. Chaque expérience est répétée 50 fois. Les agents sont initialement répartis aléatoirement au sein de l'environnement. Les figues 7.4.a et 7.4.b présentent les oisivetés moyennes sur les 50 répétitions des expériences.

Nous pouvons observer que l'ordonnancement ne change pas significativement le comportement général du modèle et n'a pas d'impact sur les performances de patrouille. Les phases d'exploration et de patrouilles mises en évidence au chapitre 4 sont toujours présentes quel que soit l'ordonnancement choisi et le nombre d'agents.



a. environnement n-pièces, 2 agents

b. environnement n-pièces, 16 agents

FIGURE 7.4 – Comparaison de l'oisiveté d'EVAP entre un ordonnancement séquentiel cyclique et ordonnancement séquentiel acyclique

Expérimentations robotiques

Pour éviter les situations de conflit, c'est-à-dire d'éviter que deux agents visitent la même cellule simultanément, nous avons mis en place un mécanisme de réservation. Quand un agent choisit sa destination, il demande à la dalle sur laquelle il se trouve de lui réserver sa dalle destination et attend une confirmation. Si la dalle est déjà occupée ou réservée, l'agent doit alors choisir une nouvelle destination. Ce mécanisme de résolution de conflit est identique à celui qu'implique le choix d'un ordonnancement *séquentiel acyclique*.

Cependant, comme le comportement des agents est décentralisé et qu'ils n'attendent pas que les autres aient terminé de se déplacer avant de choisir leur prochaine action, il est difficile de mesurer une oisiveté comparable à l'oisiveté "par itération" relevée en simulation.

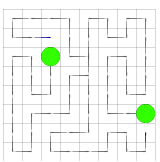
Une solution possible consisterait à synchroniser le déplacement des robots Dans ce cas, le comportement de l'implémentation robotique serait exactement le même que celui du système simulé mais avec des performances en retrait dues à l'inactivité des agents les plus rapides attendant que le dernier ait terminé son mouvement.

Les agents ne s'attendant pas après avoir bougé, les performances du système robotique devraient être légèrement meilleures que celles du système simulé (ce qui revient, nous l'avons dit, au système robotique avec une synchronisation des agents).

L'implémentation robotique, malgré les divergences de modèle d'exécution peut être jugée fidèle aux simulations. De plus, l'implémentation robotique et les modifications du modèles d'exécution des simulations n'infirment pas la preuve de la propriété de patrouille.

7.3.3 Auto-organisation en cycles

Nous avons prouvé au chapitre 6 que le modèle EVAP montre un comportement d'auto-organisation en cycles permettant d'obtenir une patrouille de très bonne qualité, proche de l'optimale (voir optimale dans le cas particulier des cycles hamiltoniens).



Cycles et ordonnancement en simulation

A la différence de la propriété de patrouille, l'ordonnancement à des effets visibles sur le processus de convergence vers des cycles. EVAP garde la capacité à converger vers des cycles quel que soit l'ordonnancement mais certaines topologies de cycles peuvent devenir instables.

Nous reprenons ici les exemples des ordonnancements séquentiels cycliques et acycliques.

La figure 7.5.a représente l'état initial de cet exemple. Les flèches (grises) sur chaque cellule représentent les destinations possibles lors de sa dernière visite par un agent (les flèches représentent en fait la plus petite pente ascendante, voir section 4.2) Les flèches rouges pointent vers la ou les cellules de plus faible valeur parmi le voisinage des agents, c'est-à-dire les cellules qui pourront être visitées par les agents lors de l'itération suivante.

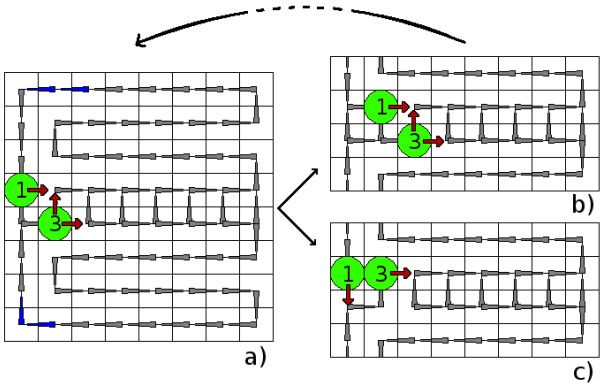


FIGURE 7.5 – Exemple de graphe absorbant avec deux agents. Les flèches rouges représentent les destinations possibles des agents.

Cas de l'ordonnancement séquentiel cyclique — Dans ce cas (voir figure 7.5.b), les agents agissent toujours dans le même ordre (selon l'identifiant des agents). L'agent a_1 a pour seul choix d'aller sur la cellule à sa droite (puisque les cellules du haut et du bas ont été visitées à l'itération précédente,). l'agent a_3 n'a alors plus d'autre choix que de visiter la cellule à sa droite. Ce comportement se répète à l'identique pour toutes les cellules "frontières" entre les deux cycles. Le système cyclera alors enter les états a et b.

Cas de l'ordonnancement séquentiel acyclique — Dans ce cas, les agents sont activés dans un ordre aléatoire. Il y a donc une chance que l'agent a_3 agisse avant l'agent a_1. Dans ce cas, a_3 peut choisir sa destination entre deux cellules (en haut ou à droite). S'il va à droite, le cycle persiste temporairement (on se retrouve dans la situation présentée en figure 7.5.b). S'il se déplace vers le haut, a_1 visite la cellule du bas (voir figure 7.5.c) et l'organisation cyclique est détruite. Le "cycle" que nous avons observé ici est en fait un sous-graphe transient.

Nous pouvons donc voir qu'un cycle avec un ordonnancement donné correspond à un sous-graphe transient avec un autre ordonnancement (et inversement). Nous avons observé expérimentalement que, pour une même configuration (pour un nombre d'agent et un environnement donné), les simulations réalisées avec un ordonnanceur séquentiel acyclique ne produit pas la

même variété de cycles. De plus, comme le nombre de cycles vers lequels il est possible de converger est plus faible qu'avec l'ordonnancement séquentiel cyclique, EVAP prend plus de temps pour converger. Notons cependant que la qualité des cycles produits, en terme d'oisiveté, ne dépendent pas du type d'ordonnancement.

7.3.4 Des robots et des cycles

Nous avons expliqué au chapitre 6 que la preuve de convergence vers des cycles repose sur l'existence d'un espace d'états fini. Toutefois, lorsque nous passons d'un modèle théorique discret (avec un espace et un temps discrets) à une implémentation réelle (avec un espace et un temps continu, même si les agents se déplacent de dalle en dalle) avec des robots, cette hypothèse n'est plus vérifiée. Le temps de déplacement des agents de dalle en dalle n'est alors plus constant. En plus des imprécisions inhérentes aux robots, la vitesse de transitions entre deux dalles peut dépendre de la destination (prendre un virage peut prendre un peu plus de temps qu'aller tout droit). Nous montrons ici, sur un cas particulier, que cette différence mène à un système comptant un nombre infini d'états.

Preuve: Soit a_1 et a_2, deux agents patrouillant une grille de taille 2×1 (nous appelons les cellules c_1 et c_2), les deux agents se trouvant initialement sur la cellule c_1. D'après le comportement décrit en algorithme 7, les deux agents alterneront indéfiniment entre les cellules c_1 et c_2.

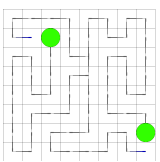
Supposons désormais que l'agent a_1 se déplace d'une cellule à l'autre en 1 unité de temps et a_2 en $\sqrt{2}$ unités de temps. Le temps peut toujours être discrétisé en considérant qu'un pas de temps commence quand un agent, a_1 ou a_2, arrive dans une cellule. Les pas de temps n'ont alors pas une durée constante et sont de la forme $t = k \times 1$ ou $t = k \times \sqrt{2}$, avec $k \in \mathbb{N}$. Cependant, comme 1 est rationnel et $\sqrt{2}$ est irrationnel[47], chaque nouveau pas de temps correspond à une nouvelle position des agents (un agent se trouvant sur une cellule et l'autre entre les deux cellules) et donc à un nouvel état du système.

Nous pouvons alors déduire qu'un tel comportement va générer un nombre infini de configurations du champ de phéromones (et donc un nombre infini d'états du système). Ainsi, même si le système est complètement déterministe, il ne sera jamais capable d'atteindre un comportement cyclique stable.

Les expérimentations robotiques montrent que, dans notre implémentation et avec plusieurs robots, le système perd effectivement la capacité de s'organiser en cycles stables. Toutefois, notons que le système ne perd la capacité à s'organiser en cycles mais que ceux-ci ne sont capable de persister que pour un temps limité (c'est-à-dire que le système ne produit que des sous-graphes transients).

Une possible solution que nous avons déjà évoquée précédemment consisterait à synchroniser les agents pour qu'ils s'attendent après chaque déplacement de façon à ce que le système se comporte conformément à ce qui a été observé en simulation et qu'il se conforme donc aux hypothèses (implicites) que nous avions faites en chapitre 6 pour la démonstration de la convergence en cycles. Notons qu'en plus augmentation de l'oisiveté du système due inactivité

47. On n'aura jamais $k \times 1 = n \times \sqrt{2}$, avec k et $n \in \mathbb{N}$



des agents qui attendent, la synchronisation implique de centraliser le système. Nous pouvons aussi attirer l'attention du lecteur sur les temps de convergence vers des cycles. Même sur un petit environnement (8×8 cellules) avec 4 agents et en supposant qu'un robot mette 5 secondes pour visiter une cellule, il faudrait attendre en moyenne 5.8 heures [48] avant d'observer la convergence. Les batteries des robots seront certainement déchargées avant qu'on puisse observer la formation d'un cycle. Toutefois, et nous rejoignons là les conclusions des expérimentations sur la robustesse des cycles du chapitre 6, il est possible d'initialiser l'environnement avec un cycle trouvé en simulation et qui persisteront pour un temps limité.

7.4 Conclusion

Dans ce chapitre, nous avons étudié les effets de différents modèles d'exécution sur le comportement d'EVAP. Nous avons présenté des aspects importants souvent ignorés durant la conception des systèmes multi-agent réactifs discrets :

- la gestion du non-déterminisme, c'est-à-dire comment réagissent les agents en cas de choix multiples ou de conflits ou encore le déterminisme des transitions entre les états du système et
- les hypothèses d'ordonnancement.

Ces point particuliers des modèles sont importants puisque qu'ils ont la capacité de changer potentiellement le comportement du modèle (voir section 7.3.4), lui faisant gagner ou perdre des propriétés selon les choix effectués. Nous avons aussi présenté une implémentation robotique reposant sur des dalles permettant l'utilisation d'un environnement actif nécessaire à l'implémentation des algorithmes en essaim.

Les systèmes robotiques utilisent cependant des hypothèses propres, très différentes (et généralement plus complexes) de celles utilisées en simulation ou dans les modèles théoriques.

Ces hypothèses sont cependant généralement choisies en fonction de contraintes implémentatoires. Par exemple, une grande partie des simulateurs multi-agent proposent par défaut un ordonnancement asynchrone qui résout implicitement certains conflits. Par ailleurs, le modèle EVAP a été implémenté et étudié dans le cadre des projets SMAART et SUSIE [49] visant à réaliser une patrouille avec un essaim de drones [Simonin, 2010]. Pour cela il a fallu adapter le modèle de déplacement des agents à un environnement continu [Legras *et al.*, 2008] (voir figure 7.6). La patrouille s'est avérée efficace mais la convergence vers des cycles n'a pu être observée, elle reste peu probable du fait des hypothèses de navigation et des perturbations provoquées par les opérateurs.

Nous pouvons dès lors nous poser la question de l'intérêt de telles études en simulation pour étudier le comportement d'une implémentation robotique future. Nous avons étudié le comportement de deux propriétés du modèle EVAP face à ces différentes modèles d'exécution. Alors que la propriété de patrouille n'est pas affectée par l'ordonnancement ou la résolution

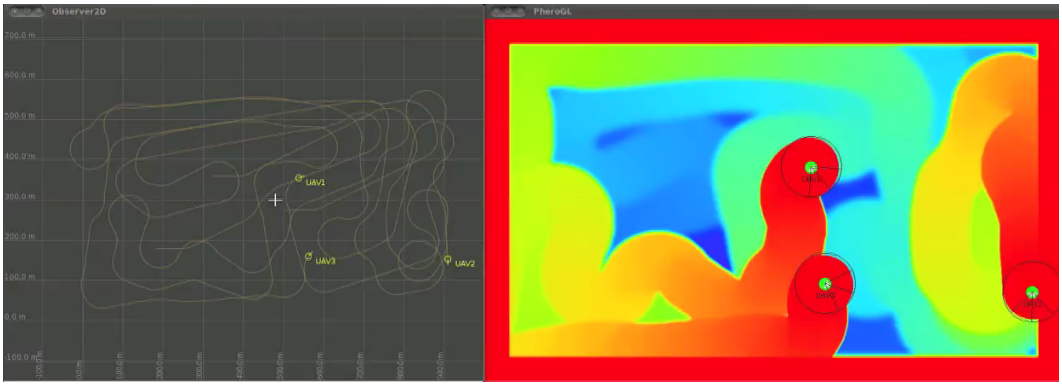
48. D'après les résultats exposés en section 6.3.5

49. Projets DGA - SMAART "Etude des SMA Adaptés à la Reconnaissance de Théâtre et à l'auto-organisation" 2006-2009 et SUSIE "SUpervision de Systèmes d'Intelligence en Essaim" 2009-2012, en partenariat avec Telecom Brest.

de conflits, la convergence vers des cycles subit des effets à la fois quantitatifs (en simulation) et qualitatifs (sur l'implémentation robotique).

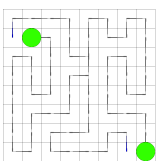
Le problème semble venir de la définition même des modèles conceptuels qui se résument souvent à un algorithme donnant le comportement d'un agent et qui ne forcent pas, faute d'une méthode adaptée, à spécifier un (ou des) modèle(s) d'exécution qui baliseront l'étude du système et poseront clairement les hypothèses de travail.

Il existe des méthodes de conception et de vérification de modèles qui forcent à spécifier toutes les interactions possibles entre agents (par exemple DEVS [Concepcion and Zeigler, 1988] et ses variations) mais elle sont complexes, non obligatoires et surtout détachés des modèles théoriques.



LightSim EVAP

FIGURE 7.6 – Simulateur du projet SUSIE : EVAP intégrant des contraintes de navigation de drones autonomes (connexion au moteur physique 2D LightSim).



Conclusion de la thèse

1 Synthèse des contributions

Dans cette thèse, nous avons approché la problématique de la résolution de problèmes à l'aide des systèmes multi-agent réactifs, une approche décentralisée et auto-organisée.

En particulier, nous avons cherché à évaluer l'intérêt de ce type d'approche pour le problème de la patrouille multi-agent en proposant le modèle EVAP. Il s'agit d'un problème complexe qui demande à un groupe d'agents de visiter le plus régulièrement possible l'ensemble des nœuds du graphe représentant l'environnement. Calculer une solution optimale à ce problème est un problème NP-complet.

Nous nous sommes intéressés, en parallèle à la résolution du problème de la patrouille, à la problématique de l'assurance des propriétés émergentes dans un système auto-organisé. Les systèmes multi-agent sont connues pour être imprévisibles et sujets à générer des comportements inattendus. Etre en mesure de prouver que les interactions entre agents ne mèneront pas à un état chaotique et pouvoir garantir les propriétés d'un système sont alors des points particulièrement importants du domaine.

EVAP, un modèle de patrouille auto-organisé

L'évaluation des performances d'EVAP pour la tâche de patrouille a été réalisée à travers la mesure, en simulation, de l'oisiveté du système sur un ensemble d'environnements de complexités variables. Nous avons pu observer dans le cas général un comportement pouvant être découpé en plusieurs phases distinctes proposant chacune des qualités propres.

Exploration de l'environnement — La première phase correspond à l'exploration de l'environnement. Durant cette phase, nous pouvons observer un pic d'oisiveté pouvant être expliqué par le comportement exploratoire sous-optimal des agents, incapables de détecter les cellules de forte oisiveté sur de grandes distance à cause de leur perception locale. Nous avons en outre identifié des topologies de *pièces imbriquées* particulièrement difficiles à explorer.

Ces difficultés d'exploration disparaissent cependant une fois la première couverture de l'environnement effectuée grâce aux phéromones qui guident les agents vers les zones de forte oisiveté. Dans le cas général, la patrouille d'EVAP se stabilise rapidement à une oisiveté

proche de l'optimal. La pire oisiveté n'est pas aussi intéressante, encore une fois à cause des perceptions locales des agents.

Les difficultés exploratoires d'EVAP nous ont mené à nous intéresser à CLInG [Sempé, 2004], un autre algorithme fourmi pour la patrouille reposant sur la propagation de l'oisiveté. CLInG repose sur un principe qui répond aux défauts d'EVAP puisque les nœuds appellent eux même les agents lorsque leur oisiveté devient importante. L'effet sur la phase d'exploration est remarquable mais ce processus ne permet pas d'obtenir une patrouille significativement meilleure que celle d'EVAP malgré les hypothèses plus fortes réalisées sur les capacités calculatoires de l'environnement (ce qui empêche également CLInG de réaliser l'exploration d'un graphe à priori inconnu).

Robustesse de la patrouille — La bonne répartition spatiale des agents au travers de l'environnement permet aux agents de maintenir une patrouille stable, même sur de grands environnement (80.000 nœuds avec 600 agents et plus) où l'oisiveté reste comparable à des instances plus réduites et plus simples du problème. Cette répartition homogène des agents dans l'environnement combinée à l'exécution *online* et décentralisée de la patrouille permet à EVAP de répondre efficacement à la suppression ou à l'ajout dynamique d'agents en redistribuant les agents dans l'environnement, la phéromone les attirant vers les zones peu visitées. EVAP est également assez résistant au bruit, puisqu'il est possible de paramètrer le comportement de la phéromone pour limiter l'influence du bruit sur la perception de l'oisiveté des agents. De plus, l'influence du bruit sur leur comportement semble être linéaire (l'oisiveté augmente de $k \times bruit$, avec $k \in \mathbb{R}+*$). L'augmentation du bruit ne change pas brutalement le comportement global du modèle ce qui permet de maintenir une patrouille acceptable même avec un fort niveau d'incertitude sur les mesures. La robustesse affichée par EVAP semble faire de ce modèle fourmi très simple un bon candidat à l'implémentation robotique.

Comportement cyclique à long terme — Nous avons pu observer lors des simulation que sur certaines configurations, le système montre un comportement à long terme de convergence vers des attracteurs cycliques stables qui produisent une patrouille très performante avec un oisiveté stable et proche de l'optimal (et même optimale dans le cas des cycles hamiltoniens).

Ce résultat avait déjà été observé dans le cadre de l'algorithme VAW de Wagner qui converge uniquement vers des cycles hamiltoniens. EVAP à la capacité de converger également vers des cycles non hamiltoniens et est donc capable d'adopter ce type de comportement cyclique sur tout type d'environnement. Les expérimentations nous ont montré que ces cycles permettent d'obtenir une meilleure oisiveté que lors de la phase non cyclique mais que le temps nécessaire à la convergence vers de telles solutions est important et que le processus de convergence est très sensible à la forme de l'environnement et au nombre d'agents.

Etude de l'auto-organisation

L'étude formelle et théorique du modèle EVAP est rendue difficile par la nature dynamique du champ de phéromones. Nous avons donc proposé EVAW, *un modèle équivalent à EVAP* utilisant le marquage statique de l'environnement, introduit dans VAW [Wagner *et al.*, 1999],

qui facilite l'étude formelle du modèle. EVAW nécessite cependant des hypothèses d'exécution différentes de celles d'EVAP. Nous remplaçons l'environnement actif, capable d'effectuer l'évaporation, par un marquage statique plus simple mais nécessitant la synchronisation des horloges des agents. Cette hypothèse peut poser problème en particulier dans le cadre de l'ajout dynamique d'agents dans une patrouille.

Preuves de comportements auto-organisés — Nous avons dans un premier temps étendu à EVAP les résultats théoriques déjà établis sur VAW, en particulier, la preuve de couverture de l'environnement en temps fini. Nous avons par ailleurs établi la *preuve de patrouille* qui assure, tant qu'un agent patrouille l'environnement, que l'ensemble des nœuds seront (re)visités infiniment souvent.

L'observation approfondie des structures cycliques nous a permis d'identifier des classes d'équivalence d'attracteurs du système. D'une part, les cycles qui sont des structures déterministes et d'autre part les sous-graphes récurrents qui sont des structures non déterministes stables. Nous avons représenté ces structures cycliques sous la forme de chaînes de Markov dont les nœuds représentent des états équivalents du système (c'est à dire des états dont les marquages ne diffèrent que d'une constante) et les transitions les évolutions possibles du système. Cette représentation a permis de réaliser l'étude formelle de ce phénomène émergent.

Nous avons tout d'abord établi que des cycles séparés (ou chaque nœud de l'environnement n'est patrouillé que par un agent) ne sont stables que s'ils sont de même longueur, ce qui assure une bonne oisiveté au système.

Dans un deuxième temps, nous avons prouvé que si le système est déterministe (dans le cas d'un système mono-agent ou déterminisé), le système converge systématiquement vers un cycle. Si le système n'est pas déterministe, il converge alors vers un sous graphe récurrent (dont les cycles sont un cas particulier). Ces deux preuves nous assurent qu'EVAP adopte, à long terme, un comportement très efficace pour la patrouille.

Détection de comportements cycliques stables dans les systèmes multi-agent discrets La représentation de l'évolution du système par une chaîne de Markov nous a permis d'étendre l'algorithme de détection de cycles dans des séquences déterministes du lièvre et de la tortue de Floyd [Floyd, 1967] à des situations non déterministes. Cet algorithme de détection de comportements cycliques est en outre extensible à l'ensemble des système dynamiques discrets dès lors que leur évolution peut être représentée sous la forme d'une chaîne de Markov.

Amélioration de convergence vers des cycles — L'étude du champ de phéromone nous a permis de mettre évidence des motifs responsables de la convergence vers des comportements cycliques. Nous avons identifié le motif des flip-flops, qui permettent de la formation de cycles non hamiltoniens, et utilisé le motif des queues de cycle pour proposer une amélioration du comportement des agents qui permet une augmentation importante de la vitesse de convergence vers des cycles.

Nous avons ainsi montré à travers des expérimentation sur des environnement de formes et de tailles variées que cette amélioration de comportement permet à EVAP de converger

sur des environnement bien plus grands et avec beaucoup plus d'agents. Ce phénomène reste cependant soumis à un explosion du temps de convergence lorsque la complexité de l'environnement patrouillé augmente. Nous avons en outre, à travers ces expérimentations, mis en évidence l'existence d'un lien fort entre la topologie de l'environnement et la vitesse de convergence vers des cycles.

2 Perspectives

Le travail réalisé dans cette thèse ouvre plusieurs perspectives intéressantes.

2.1 Amélioration du comportement exploratoire d'EVAP

La première perspective que nous présentons ici concerne directement le comportement d'EVAP. Nous avons en effet mis en lumière les difficultés exploratoires rencontrées par ce modèle. Si le déploiement des agents depuis plusieurs points de l'environnement permet d'accélérer l'exploration, il ne s'agit pas d'une solution satisfaisante. Plusieurs solutions sont envisageables. Nous pouvons faire déposer par les agents une seconde phéromone à évaporation rapide avec une faible diffusion qui agirait comme un champ répulsif. Lorsque plusieurs agents se trouvent dans une zone réduite, l'apport en phéromone devient supérieur à sa vitesse d'évaporation, rendant la zone moins attractive. Une autre solution pourrait consister à utiliser des agents hétérogènes. Une partie des agents déployés adopteraient un comportement très exploratoire qui ne serait pas influencé par les phéromones de *visite*. Ce comportement pourrait être activé ou désactivé au besoin à l'aide d'une boucle de rétroaction. Percevoir majoritairement des valeurs importantes de phéromones, sur une fenêtre temporelle donnée, activerait ce comportement alors que la perception répétée de valeurs basses l'inhiberait.

2.2 Extension d'EVAP à d'autres problèmes

Lorsque qu'un agent EVAP est déployé sur un arbre, nous pouvons remarquer que celui-ci adopte un comportement équivalent à une recherche en profondeur (on notera d'ailleurs, que l'agent répétera ce parcours après la première exploration). EVAP pourrait donc se révéler comme un étant un moyen efficace et décentralisé pour l'exploration de grands espaces d'états. Nous pouvons également nous interroger sur le sens et les implications d'effectuer une recherche en profondeur sur un graphe qui n'est par ailleurs qu'un arbre contenant des cycles. En généralisant, nous pouvons nous interroger sur les capacités d'EVAP et des autres algorithmes fourmi que nous avons présenté ici à être utilisés pour l'exploration des structures de données en général.

2.3 Algorithmes fourmi et programmation dynamique : des phéromones pour apprendre

Dans [Koenig *et al.*, 2001], les auteurs signalent l'existence d'un lien fort entre les algorithmes fourmi (en particulier pour RTLA*) et la programmation dynamique. La loi d'évolution des algorithmes fourmi ressemble en effet à certains algorithmes d'apprentissage par

renforcement, tels que le Q-learning par exemple. Les agents explorent un graphe d'état (l'environnement) et choisissent l'état suivant (leur destination) en fonction d'une récompense (ici, la différence entre les marquages du nœud courant et de la destination choisie). Suite à l'exploration de l'état choisi (c'est-à-dire après avoir effectué leur déplacement), ils mettent à jour la valeur du nœud sur lequel ils se trouvent (Le Q-learning met à jour les Q-valeurs des couples état-action, ce qui reviendrait plutôt, dans le cadre des algorithmes fourmi à marquer, les arcs du graphe). Notons aussi que les algorithmes fourmi que nous avons présentés dans cette thèse vérifient tous la propriété de Markov (le choix du prochain état est uniquement fonction de l'état courant) puisqu'ils ne disposent d'aucune mémoire et ne décident de leur destination qu'en fonction de la perception de leur voisinage.

Au delà de ca, nous avons pu remarquer, lors des expérimentations, que la convergence d'EVAP vers des cycles dépend fortement de la forme du graphe exploré. Nous retrouvons ici un phénomène similaire à celui observable sur les algorithmes d'apprentissage par renforcement pour lesquels certains problèmes sont difficiles à résoudre à cause de la forme du graphe représentant l'espace d'états.

Expliciter formellement les liens pouvant exister entre ces deux domaines pourrait mener à transposer certains résultats de la programmation dynamique vers les systèmes multi-agent réactifs et d'appréhender le problème de l'apprentissage par renforcement avec vision auto-organisée du processus de convergence.

2.4 Résolution de problèmes par des approches auto-organisées

Dans [Koenig *et al.*, 2001], les auteurs font remarquer que les algorithmes fourmi sont tous construits sur un squelette commun, les différences de comportement entre les modèles étant définies par la sémantique du marquage. Cataloguer l'ensemble des marquages possibles (avec/sans évaporation, avec/sans diffusion, marquage additif ou substitutif) et des critères de sélection des agents (tropismes positifs et négatifs, choix stochastiques ou déterministes) pourrait permettre l'identifications de fonctions génériques qui pourront ensuite être combinées pour résoudre des problèmes complexes de manière totalement réactive.

Table des figures

1.1 Boucle perception-action. 5

1.2 Schéma représentant l'architecture d'un agent 6

1.3 Représentation 3D d'un champ de potentiel tiré de [Ferber, 1995]. 11

1.4 Evaporation de la phéromone selon deux taux d'évaporation différents 13

2.1 Une couverture optimale ne signifie pas nécessairement une patrouille optimale 19

2.2 Maximisation de l'utilité pour la capture d'intrus. Sachant qu'un intrus a été récemment perçu à cet endroit, l'agent aura intérêt à se déplacer vers le couloir de gauche. En effet, des deux zones (en vert et en bleu) contenant de l'information (sur la présence ou l'absence d'intrus), l'agent rejoindra la zone vert plus rapidement et minimisera ainsi le temps perdu dans l'éventualité ou l'intrus ne s'y trouve pas. 20

2.3 Représentation d'un environnement (un quartier de Nancy) par un graphe pondéré. Les arcs sont valués relativement à la distance séparant les nœuds (les valeurs ne sont pas représentées sur le schéma). *Fond de carte tiré de google maps.* 21

2.4 Un cycle (arcs rouge en gras) visitant tous les nœuds du graphe sans réaliser de couverture de l'environnement à proprement dire (les zones couvertes par les arcs noirs restent non visités). 21

2.5 Représentation d'un environnement continu par une grille régulière composée de nœuds de degré 4. 22

2.6 En visitant toujours, à un instant t, le nœud le plus anciennement visité, l'agent se trouvera toujours au temps $t+1$ sur un nœud voisin du nœud le plus anciennement visité. Le système suit alors un comportement cyclique optimal. . 24

2.7 Deux environnements grille carrés, admettant ou non un cycle hamiltonien, l'un de coté de taille paire et l'autre de coté de taille impaire. 25

2.8 Calcul du cycle le plus court par optimisation par colonie de fourmi. A l'initialisation (à gauche), tous les arcs possèdent la même quantité de phéromones. Au cours de l'exécution (au centre), les agents renforcent certains arcs ; les arcs non renforcés disparaissent progressivement (à droite) jusqu'à ne laisser plus qu'un seul chemin. L'expérience est ensuite répétée jusqu'à obtention d'une solution satisfaisante. 27

2.9 Un spanning tree (arcs bleus) d'un graphe grille 27

2.10 Un cycle hamiltonien réalisé à partir d'un spanning tree sur un environnement grille de 8x8 nœuds. 28

3.1 Descente du gradient de phéromone et marquage de l'environnement par un agent 39

3.2 Conflit entre deux agents pour l'accès simultané à un nœud 41

3.3 Une itération du modèle EVAP tel que décrit par l'algorithme 9 42

3.4 Deux représentations possibles de l'environnement 43

3.5 Voisinage de von Neumann (en bleu) de la cellule jaune sur une grille régulière 43

3.6 Captures d'écran d'une simulation de patrouille de l'environnement *densité* par 8 agents EVAP. Plus les cellules sont claires, plus elles ont été visitées récemment. La capture b. représente le système quelques itérations après que la capture a. ait été prise. 44

3.7 Exploration d'un environnement grille de 6*10 cellules par un groupe de 4 agents 45

3.8 Patrouille de l'environnement après une première exploration. Les zones sombres correspondent aux cellules dont le délai depuis la dernière visite est le plus important. Les flèches indiquent les directions que prendront les agents au cours des prochaines itérations. 45

3.9 Le système a convergé vers un cycle unique sur lequel sont répartis 4 agents (cf. représentation du cycle en figure 3.10). 45

3.10 Une représentation plus explicite du cycle présenté sur la figure 3.9. 46

3.11 Oisiveté pour 4 agents sur un environnement grille de 10x6 cellules. Les courbes se stabilisent à l'optimal après 230 itérations, reflétant la convergence vers un cycle stable. 47

3.12 Profil typique de l'oisiveté moyenne au niveau du graphe 48

3.13 Différences de comportement entre EVAW et VAW : b) les agents EVAW se déplacent puis marquent, c) les agents VAW marquent puis se déplacent. . . . 50

3.14 Illustration des conséquences de la perception locale pour un agent EVAP . . 53

4.1 Courbes d'oisiveté pour un environnement 12x12 cellules patrouillé par deux agents. On peut distinguer trois phases différentes : la phase d'exploration, la phase d'organisation et la phase cyclique. 56

4.2 Schéma représentant le fonctionnement de CLInG 58

4.3 liste des environnements utilisés : 50x50 (2500 nœuds), Koenig (883 nœuds), densité 20% (321 nœuds), pièces imbriquées (364 nœuds) et n-pièces (336 nœuds) 60

4.4 Temps de couverture moyens comparés pour EVAP, LRTA* et CLInG avec 20 agents sur les 5 environnements proposés. 61

4.5 Proportion de l'environnement "densité 20%" couvert par EVAP, LRTA* et CLInG en fonction du temps. 62
4.6 Proportion de l'environnement "pièces imbriquées" couvert par EVAP, LRTA* et CLInG en fonction du temps. 63
4.7 Présentation de la situation initiale permettant d'expliquer le comportement exploratoire des trois approches. 64
4.8 Comportements des trois algorithmes à partir de la situation initiale de la figure 4.7. 65
4.9 Schéma représentant l'entrée d'une *pièce*. 65
4.10 Exploration des pièces imbriquées par EVAP. 66
4.11 Comparaison des oisivetés d'EVAP et de CLInG utilisant un paramétrage favorisant l'exploration. 67
4.12 Comparaison EVAP/LRTA*/CLInG sur l'environnement 50x50 cellules avec 20 agents 68
4.13 Comparaison EVAP/LRTA*/CLInG sur l'environnement Densité avec 20 agents 68
4.14 Comparaison EVAP/LRTA*/CLInG sur l'environnement pièces imbriquées avec 20 agents 69
4.15 Comparaison EVAP/LRTA*/CLInG sur l'environnement Koenig avec 20 agents 69
4.16 Comparaison EVAP/LRTA*/CLInG sur l'environnement n-pièces avec 20 agents 70
4.17 Représentation par niveaux de couleur du champ scalaire de LRTA* et de l'oisiveté pendant la patrouille. 70
4.18 Comparaison des moyennes des IGI sur la durée des simulations des modèles EVAP et CLInG et des valeurs optimales théoriques sur l'environnement 50x50 cellules avec un nombre croissant d'agents. 72
4.19 Robustesse à la perte d'agents : oisiveté sur l'environnement 25 pièces avec un passage de 20 à 10 agents puis réintroduction de 10 agents supplémentaires. . 74
4.20 Oisiveté (IGI et WGI) sur les environnements 400×200 avec 600 agents et 40×20 avec 6 agents (même densité d'agents) 75
4.21 Oisiveté (IGI et WGI) sur les environnements 400×200 avec 600 agents et 40×20 avec 6 agents. Zoom sur l'intervalle [15.000 ;70.000]. 76
4.22 Environnement de 400×200 cellules (80.000 nœuds) et 600 agents. Chaque petit point vert correspond à un agent. 77

5.1 Un cycle d'états du système. Le système converge à l'état s_3 vers un cycle de 4 états (s_3, s_4, s_5, s_6) qui se répétera indéfiniment. 82
5.2 Chaîne de Markov homogène représentées par un graphe orienté (dans le cas d'EVAP, les transitions à la sortie d'un nœud sont équiprobables, cf. section 5.1.2). 83
5.3 Un exemple de sous-graphe transient. L'agent 1 agit toujours en premier ; tant que celui ci va vers la droite, le sous-graphe persiste ; s'il choisit de se déplacer vers le haut, le système sort du sous-graphe. 84

5.4 Un exemple de sous-graphe récurrent. Les nœuds du graphe représentent les états non déterministes du sous-graphe. récurrent 84

5.5 Deux agents dans un cycle d'états du système. Toutes les transitions sont déterministes. 85

5.6 Schématisation de l'idée de la preuve de convergence vers des cycles dans le cadre mono-agent 87

5.7 Illustration de l'algorithme du lièvre et de la tortue 88

5.8 Détection de faux positifs par la lièvre et la tortue. 90

5.9 Flowchart représentant l'algorithme de détection de cycles dans des séquences non déterministes 91

5.10 Etat d'avancement des deux séquences lorsque le lièvre a convergé vers un cycle 94

6.1 Deux cycles optimaux pour les mêmes paramètres de simulation : a - cycle de 32 états, b - cycle de 64 états 99

6.2 Deux cycles distincts de longueur différentes connectés par les nœuds (v_1, v_2) 100

6.3 Un cycle d'états du système composé de deux cycles de longueurs égales. . . . 101

6.4 Un cycle instable (transitoire) d'états du système composé de deux cycles de longueur égale 102

6.5 Agents se déplaçant dans des sens opposés le long de la frontière entre leurs cycles 103

6.6 Agents se déplaçant dans le même sens le long de la frontière entre leurs cycles 103

6.7 Cycles d'états du système avec a) une frontière non continue et b) plus de deux agents 103

6.8 Cycles d'états du système avec a) un cycle commun aux deux agents and b) une cellule partagées par deux agents 104

6.9 Deux représentations à base de champs de vecteurs d'un même état du système 106

6.10 Mise en évidence d'un chemin hamiltonien partiel et la zone de perception nécessaire pour détecter son entrée. 106

6.11 Une fourmi (notée avec des crochets) parcourant un cycle non-hamiltonien avec deux flip-flops (notés par des parenthèses). Les figures montrent les pas de temps 12 et 15. 107

6.12 Perte de la propriété de patrouille 108

6.13 Courbes de convergence pour un agent ("vrais" cycles seulement) 111

6.14 Courbes de convergence pour deux agents ("vrais" cycles seulement) 112

6.15 Courbes de convergence pour trois agents ("vrais" cycles seulement) 112

6.16 Deux environnements irréguliers 113

6.17 Influence de l'heuristique EVAW+ sur la qualité de la patrouille 114

7.1 Conflit entre agents pour la cellule centrale. 120

7.2 Schéma de l'émulateur iTiles 123

7.3 Implémentation robotique d'EVAP sur l'émulateur de dalles 123

7.4 Comparaison de l'oisiveté d'EVAP entre un ordonnancement séquentiel cyclique et ordonnancement séquentiel acyclique 125

7.5 Exemple de graphe absorbant avec deux agents. Les flèches rouges représentent les destinations possibles des agents. 126

7.6 Simulateur du projet SUSIE : EVAP intégrant des contraintes de navigation de drones autonomes (connexion au moteur physique 2D LightSim). 129

Bibliographie

[Almeida *et al.*, 2004] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre. Recent advances on multi-agent patrolling. In *Advances in Artificial Intelligence — Seventeenth Brazilian Symposium on Artificial Intelligence (SBIA'04)*, pages 474–483. Springer-Verlag, 2004.

[Andrade *et al.*, 2001] R.d.C. Andrade, H.T. Macedo, G.L. Ramalho, and C.A.G. Ferraz. Distributed mobile autonomous agents in network management. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'01)*, 2001.

[Arkin, 1987] R. Arkin. Motor schema based navigation for a mobile robot : An approach to programming by behavior. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, pages 264 – 271, mar 1987.

[Axtell, 2001] R. Axtell. Effects of interaction topology and activation regime in several multi-agent systems. In *MABS 2000 : Proceedings of the second international workshop on Multi-agent based simulation*, pages 33–48, Secaucus, NJ, USA, 2001. Springer-Verlag New York, Inc.

[Basilico *et al.*, 2009] Nicola Basilico, Nicola Gatti, and Francesco Amigoni. A formal framework for mobile robot patrolling in arbitrary environments with adversaries. *CoRR*, abs/0912.3275, 2009.

[Bonabeau *et al.*, 2001] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. Swarm intelligence : From natural to artificial systems. *J. Artificial Societies and Social Simulation*, 4(1), 2001.

[Bourjot *et al.*, 2003] C. Bourjot, V. Chevrier, and V. Thomas. A new swarm mechanism based on social spiders colonies : from web weaving to region detection. *Web Intelligence and Agent Systems*, 1 :47–64, 2003.

[Briot and Demazeau, 2001] J-P. Briot and Y. Demazeau. *Principes et Architecture des Systèmes Multi-Agents*. Hermes, 2001.

[Brooks, 1986] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1) :14 – 23, mar 1986.

[Cho and Garcia-Molina, 2000] J. Cho and M. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the International Conference on Management of Data (SIGMOD'00)*, 2000.

[Chu *et al.*, 2007] H. Chu, A. Glad, O. Simonin, F. Sempé, A. Drogoul, and F. Charpillet. Swarm approaches for the patrolling problem, information propagation vs. pheromone eva-

poration. In *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'07)*, pages 442–449, 2007.

[Concepcion and Zeigler, 1988] Arturo I. Concepcion and Bernard P. Zeigler. Devs formalism : A framework for hierarchical model development. *IEEE Trans. Software Eng.*, 14(2) :228–241, 1988.

[Deneubourg *et al.*, 1990] Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2) :159–168, March 1990.

[Dorigo and Gambardella, 1997] Marco Dorigo and Luca Maria Gambardella. Ant colony system : a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evolutionary Computation*, 1(1) :53–66, 1997.

[Drogoul, 1993] A. Drogoul. *De la simulation multi-agent à la résolution collective de problèmes : une étude de l'émergence de structures d'organisation dans les systèmes multi-agents.* PhD thesis, Université Paris VI, 1993.

[Edmonds and Hales, 2003] Bruce Edmonds and David Hales. Replication, replication and replication : Some hard lessons from model alignment. *J. Artificial Societies and Social Simulation*, 6(4), 2003.

[Elmaliach *et al.*, 2007] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-07)*, 2007.

[Faratin *et al.*, 2002] P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make issue trade-offs in automated negotiations. *Artificial Intelligence*, 142 :205–237, 2002.

[Fatès, 2004] Nazim Fatès. *Robustesse de la dynamique des systèmes discrets : le cas de l'asynchronisme dans les automates cellulaires.* PhD thesis, Ecole normale supérieure de Lyon, 2004.

[Ferber and Muller, 1996] J. Ferber and J.P. Muller. Influences and reaction : a model of situated multiagent systems. In *Proceedings of the 2nd International Conference on Multi-agent Systems*, pages 72–79, 1996.

[Ferber, 1995] Jacques Ferber. *Les Systemes Multi-Agents : Vers une intelligence collective.* InterEditions, 1995.

[Floyd, 1967] R.W. Floyd. Non-deterministic algorithms. *Journal of the ACM*, 14(4) :636–644, October 1967.

[Gabriely and Rimon, 2001] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Ann. Math. Artif. Intell.*, 31(1-4) :77–98, 2001.

[Gardner, 1977] M. Gardner. Mathematical games : The fantastic combinations of john conway's new solitaire game 'life' (the original description of conway's game of life). *Scientific American*, 236 :120–123, 1977.

[Glad *et al.*, 2008] A. Glad, O. Simonin, O. Buffet, and F. Charpillet. Theoretical study of ant-based algorithms for multi-agent patrolling. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08), pp. 626-630*, pages 626–630, 2008.

[Glad *et al.*, 2009] A. Glad, O. Buffet, O. Simonin, and F. Charpillet. Self-organization of patrolling-ant algorithms. In *CD-ROM Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems, SASO09*, 2009.

[Glad *et al.*, 2010] A. Glad, O. Buffet, O. Simonin, and F. Charpillet. Influence of different execution models on patrollin ant behaviors : from agents to robots. In *CD-ROM Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems, AAMAS'10*, 2010.

[Grassé, 1959] P.-P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie : Essai d'interprétation du comportement des termites constructeurs. In *Insectes Sociaux*, volume 6, pages 41–84, 1959.

[Karp, 1972] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Khatib, 1985] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 500–505, St. Louis, 1985.

[Koenig *et al.*, 2001] S. Koenig, B. Szymanski, and Y. Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31(1–4), 2001.

[Korf, 1990] Richard E. Korf. Real-time heuristic search. *Artif. Intell.*, 42(2-3) :189–211, 1990.

[Kubera *et al.*, 2010] Yoann Kubera, Philippe Mathieu, and Sébastien Picault. Everything can be agent ! In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen, editors, *AAMAS*, pages 1547–1548. IFAAMAS, 2010.

[Legras *et al.*, 2008] F. Legras, A. Glad, O. Simonin, and F. Charpillet. Authority sharing in a swarm of UAVs : simulation and experiments with operators. In *in SIMPAR'08 International Conference on Simulation, Modeling and Programming for Autonomous Robots LNAI 5325 Springer-Verlag*, pages 293–304, 2008.

[Machado *et al.*, 2002] A. Machado, G. Ramalho, J-D. Zucker, and A. Drogoul. Multi-agent patrolling : an empirical analysis of alternative architectures. In *Third International Workshop on Multi-Agent Based Simulation*, pages 155–170, 2002.

[Mamei and Zambonelli, 2005] M. Mamei and F. Zambonelli. Spreading pheromones in everyday environments through RFID technology. In *2nd IEEE Symposium on Swarm Intelligence*, 2005.

[Mandelbrot and Hudson, 2006] Benoit Mandelbrot and Richard L. Hudson. *The Misbehavior of Markets : A Fractal View of Financial Turbulence.* Basic Books, annotated edition edition, March 2006.

[Marino *et al.*, 2009] Alessandro Marino, Lynne E. Parker, Gianluca Antonelli, and Fabrizio Caccavale. Behavioral control for multi-robot perimeter patrol : A finite state automata approach. In *ICRA*, pages 831–836. IEEE, 2009.

[Michel *et al.*, 2001] F. Michel, J. Ferber, and O. Gutknecht. Generic simulation tools based on MAS organization. In *Proceedings of the 10th European Workshop on Modelling Autonomous Agents in a Multi Agent World MAMAAW*, 2001.

[Michel *et al.*, 2005] F. Michel, G. Beurier, and J. Ferber. The TurtleKit simulation platform : Application to complex systems. In *Proceedings of the International Conference on Signal-Image Technology and Internet-Based Systems (SITIS'05)*, 2005.

[Michel, 2007] Fabien Michel. The irm4s model : the influence/reaction principle for multiagent based simulation. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *AAMAS*, page 133. IFAAMAS, 2007.

[Minsky, 1986] Marvin Minsky. *The society of mind.* Simon & Schuster, Inc., New York, NY, USA, 1986.

[Moujahed, 2007] Sana Moujahed. *Approche multi-agents auto-organisée pour la résolution de contraintes spatiales dans les problèmes de positionnement mono et multi-niveaux.* PhD thesis, Université de technologie de Belfort Montbéliard, 2007.

[M.R.-Jean, 1997] M.R.-Jean. Emergence et SMA. In *JFIADSMA'97, Hermès*, Nice, 1997.

[Muller, 1996] J.P. Muller. The design of intelligent agents. *Lecture Notes in Artificial intelligence*, 1177, 1996.

[Parunak *et al.*, 2002] H. Van Dyke Parunak, M. Purcell, and R. O'Connell. Digital pheromones for autonomous coordination of swarming UAV's. In *Proc. of AIAA First Technical Conference and Workshop on Unmanned Aerospace Vehicles, Systems, and Operations*, 2002.

[Pellier, 2005] D. Pellier. *Modèle dialectique pour la synthèse de plans.* PhD thesis, Université Joseph Fourier, Grenoble, 2005.

[Pépin *et al.*, 2009] N. Pépin, O. Simonin, and F. Charpillet. Intelligent tiles : Putting situated multi-agents models in real world. In *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART'09)*, pages 513–519, 2009.

[Raafat *et al.*, 2009] Ramsey M. Raafat, Nick Chater, and Chris Frith. Herding in humans. *Trends in Cognitive Sciences*, 13(10) :420–428, October 2009.

[Rao and Georgeff, 1995] Anand S. Rao and Michael P. Georgeff. Bdi agents : From theory to practice. In *In proceedings of the first international conference on multi-agent systems (ICMAS-95)*, pages 312–319, 1995.

[Resnick, 1994] M. Resnick. *Turtles, Termites, and Traffic Jams : Explorations in Massively Parallel Microworlds.* Cambridge, MA : MIT Press, 1994.

[Reynolds, 1987] Craig W. Reynolds. Flocks, herds and schools : A distributed behavioral model. In *SIGGRAPH '87 : Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, July 1987. ACM.

[Seeley and Buhrman, 2001] Thomas D. Seeley and Susannah C. Buhrman. Nest-site selection in honey bees : how well do swarms implement the "best-of-N" decision rule ? *Behavioral Ecology and Sociobiology*, 49(5) :416–427, April 2001.

[Sempé, 2004] François Sempé. *Auto-organisation d'une Collectivité de Robots : Application à l'Activité de patrouille en présence de perturbation.* PhD thesis, Université Pierre et Marie Curis (paris, France), 2004.

[Simonin, 2001] Olivier Simonin. *Le modèle Satisfaction-Altruisme : coopération et résolution de conflits entre agents situés réactifs, application à la robotique.* PhD thesis, Université Montpellier II, 2001.

[Simonin, 2010] Olivier Simonin. *Contribution à la résolution collective de problème - Modèles d'auto-organisation par interactions directes et indirectes dans les SMA réactifs et robotiques.* PhD thesis, Université Henri Poincaré (Nancy I), 2010.

[Thrun, 1992] S. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, 1992.

[Wagner and Bruckstein, 1999] Israel A. Wagner and Alfred M. Bruckstein. Hamiltonian(t) - an ant-inspired heuristic for recognizing hamiltonian graphs, 1999.

[Wagner *et al.*, 1999] I. Wagner, M. Lindenbaum, and A. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15 :918–933, 1999.

[Weyns and Holvoet, 2003] D. Weyns and Tom Holvoet. Model for simultaneous actions in situated multi-agent systems. In *First German Conference on Multi-Agent System Technologies*, pages 105–119. Springer–Verlag, 2003.

[Wooldridge *et al.*, 1999] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design, 1999.

[Yanovski *et al.*, 2001] Vladimir Yanovski, Israel A. Wagner, and Alfred M. Bruckstein. Vertex-ant-walk - a robust method for efficient exploration of faulty graphs. *Ann. Math. Artif. Intell.*, 31(1-4) :99–112, 2001.

Résumé

Nous abordons, dans cette thèse, la problématique de la résolution de problèmes à l'aide des systèmes multi-agent réactifs, une approche décentralisée et auto-organisée. Nous étudions comment des agents réactifs, dont les décisions ne dépendent que de leurs perceptions locales, peuvent interagir pour produire des solutions robustes et performantes. Assurer formellement les propriétés d'un tel système devient alors un enjeu particulièrement important du domaine. Nous cherchons en particulier à évaluer l'intérêt de ce type d'approches pour pour le problème de la patrouille multi-agent qui consiste à visiter l'ensemble des nœuds d'un environnement discret le plus régulièrement possible. Nous proposons le modèle EVAP qui repose sur l'utilisation d'agents fourmi se coordonnant par marquage de l'environnement à l'aide de phéromones digitales. Nous nous intéressons à ce modèle à travers les études théorique et expérimentale de son comportement. En particulier, nous prouvons que les agents s'auto-organisent vers des attracteurs cycliques stables. Ceux-ci garantissent une fréquence de visite de l'environnement quasi optimale. Nous étudions enfin la robustesse d'EVAP aux variations des hypothèses d'exécution.

Mots-clés: Systèmes Multi-Agent Réactifs, Approches bio-inspirées, Patrouille multi-agent, Auto-organisation, Emergence, Simulation individu centrée

Abstract

We use here reactive multi-agent systems — a self-organized and decentralized approach — for problem solving. We study how reactive agents, acting only according to their local perceptions, can produce robust and competitive performance. So, getting formal proofs of the system's properties is an important issue to be taken care of. In particular, we evaluate the interest of these kind of approaches for the multi-agent patrolling problem which aims for a group of agents at repeatedly visiting the vertices of a discrete environment. We tackle these problems by proposing the EVAP model. It relies on a group of agents patrolling the environment only using pheromone markings and their local perceptions. We take on this model through both theoretical and experimental studies of its behavior. In particular, we prove the self-organization of the system in stable cycles which are near optimal in terms of visit frequency. This property is particularly interesting as it guarantees the long-term performance of the patrol. We also study EVAP's robustness against the modification of the execution model.

Keywords: Reactive Multi-Agent Systems, Bio-inpired approaches, Multi-agent patrolling, Self-organization, Emergence, Agent based simulation