Auto-organisation dans les algorithmes fourmis pour la patrouille multi-agent

Arnaud Glad, Olivier Buffet, Olivier Simonin, and François Charpillet

MAIA Team / LORIA — INRIA / Nancy Université
Campus Scientifique
BP 239 — 54 506 Vandœuvre-lès-Nancy
firstname.lastname@loria.fr

Résumé : Nous considérons ici la patrouille multi-agent comme la tâche, pour un groupe d'agents, de visiter l'ensemble des cellules d'un environnement de manière répétée et aussi regulièrement que possible. Wagner *et al.* (1999) ont introduit les algorithmes fourmis pour la patrouille, dans lesquels chaque agent peut seulement marquer son environnement et s'y déplacer en fonction de ses perceptions locales. Parmi divers résultats, il a été observé expérimentalement que, pour certains algorithmes, les agents s'auto-organisent souvent en cycles stables, lesquels sont proche de l'optimum en termes de fréquence de visite. Cette propriété garantie la performance à long terme de la patrouille.

Le présent article se focalise sur le comportement de convergence d'un algorithme fourmi typique, EVAW (Wagner *et al.*, 1999; Glad *et al.*, 2008). Notre principale contribution est la preuve théorique de l'auto-organisation du groupe d'agents en cycles sous certaines hypothèses. Ces hypothèses reposent sur certains détails d'implémentation qui permettent de contrôler la prédictibilité du système. En plus de ces résultats qualitatifs sur le comportement de convergence, nous cherchons à évaluer expérimentalement ses caractéristiques. Ceci a amené à une seconde contribution : un algorithme détectant les régimes permanents. Finalement, nous proposons un algorithme amélioré qui accélère fortement le processus d'auto-organisation et nous permet ainsi de conduire des expérimentations sur de plus grands problèmes (en termes de taille et de nombre d'agents).

1 Introduction

L'exploration et la surveillance efficaces de grands environnements est une question centrale en robotique mobile, mais les algorithmes de patrouille multi-agent peuvent aussi être utilisés dans d'autres applications telles que la surveillance d'un réseau ou de sites web (Andrade *et al.*, 2001; Cho & Garcia-Molina, 2000). Diverses approches ont été proposées pour la patrouille multi-agent, telles que décrites et évaluées dans (Almeida *et al.*, 2004), basées par exemple sur des agents i) suivant la solution d'un problème de voyageur de commerce, ii) appliquant des règles heuristiques, iii) négociant des objectifs ou iv) apprenant par renforcement. La performance des algorithmes de patrouille est usuellement mesurée à travers l'oisiveté moyenne sur l'environnement (temps entre deux visites d'une cellule).

Dans ce contexte nous sommes particulièrement intéressés par les algorithmes fourmi de couverture et de patrouille, algorithmes introduits par Wagner *et al.* (1999) et qui garantissent de visiter de manière répétitive toutes les cellules d'un environnement discret et ce, avec des performances compétitives en termes d'oisiveté (Koenig *et al.*, 2001; Chu *et al.*, 2007).[1] Ces algorithmes ne requièrent pas d'information sur l'environnement puisque chaque agent ne marque l'environnement et ne s'y déplace qu'en fonction de ses perceptions locales.

Dans cet article, nous ne focalisons pas sur la performance de la patrouille en termes d'oisiveté, mais étudions son comportement à long-terme. En effet, il a été observé que les agents s'auto-organisent souvent en cycles stables. Ces cycles, qui sont hamiltoniens ou quasi-hamiltoniens, fournissent généralement une solution proche de l'optimal au problème de la patrouille multi-agent. C'est par exemple le cas de divers algorithmes de la famille VAW (Wagner *et al.*, 1999) et en particulier d'EVAW (Glad *et al.*, 2008), l'agorithme fourmi typique que nous considérons ici, lequel est décrit en section 2.

[1] Nous ne connaissons pas de travaux comparant les algorithmes fourmi (aussi appelés algorithmes *real-time search* par Koenig *et al.* (2001)) et les autres types d'algorithmes de patrouille(Almeida *et al.*, 2004) dans la littérature.

La contribution principale de cet article, présentée en section 3, est de prouver théoriquement que le groupe d'agents s'auto-organise en cycles sous certaines hypothèses. Ces hypothèses, liées au déterminisme des agents et de leurs interactions, permettent de contrôler la prédictibilité du système. En plus de ces résultats qualitatifs sur le comportement de convergence, nous évaluons expérimentalement ses caractéristiques. Pour réaliser ces expérimentations, nous introduisons un algorithme qui permet de détecter les régimes permanents (voir section 4). Finalement, nous proposons en section 5 un algorithme amélioré qui accélère fortement le processus d'auto-organisation et nous permet de conduire des expérimentations (section 6) sur de plus grands problèmes (aussi bien en termes de dimension que de nombre d'agents). La section 7 conclue sur ce travail et présente ses perspectives.

2 Contexte

2.1 Problème de la patrouille

Ici, l'environnement discret (et fini) $\mathcal{E}$ est modélisé par un graphe orienté fortement connecté[2] dans lequel les sommets sont des cellules $c_j \in C$ (de voisinage $N(c_j)$) formant une grille. L'*état* d'un agent fourmi $a_i \in A$ est décrit par sa cellule courante : $s(a_i) = c_j$. Dans ce contexte, nous considérons la patrouille multi-agent comme le problème, pour des agents, de visiter chaque cellule de l'environnement de manière répétée.

2.2 Approches à base de fourmis

Dans une approche fourmi typique, les agents fourmis ne connaissent pas leur environnement et ne possèdent aucune mémoire, et chaque cellule c_j est dotée d'un marquage $m(c_j)$ ($\in M$ l'ensemble des marquages possibles). Les agents sont seulement capables de se déplacer vers une cellule voisine, de marquer leur cellule courante et de percevoir le marquage des cellules voisines. Ce marquage de l'environnement $\mathcal{E}$ est inspiré des phéromones de fourmis réelles. Ainsi, nous pouvons définir :

– l'*état* de l'*environnement* $s(\mathcal{E})$ comme le marquage de toutes ses cellules ;
– l'*état* du *système* comme le tuple $s = (s(\mathcal{E}), s(a_1), \cdots, s(a_{|A|}))$.

Si M est un espace vectoriel (par exemple $M = \mathbb{R}^n$), 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 d'une constante :

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

Quand M est uni-dimensionnel ($n = 1$), nous identifions souvent l'état de l'environnement $s(\mathcal{E})$ avec l'état équivalent $s^*(\mathcal{E})$ dont le marquage minimal d'une cellule est 0 ($\min_j m(c_j) = 0$). La même notion d'équivalence s'étend naturellement aux états du système. De plus, si nécessaire, un indice t est ajouté aux états pour spécifier le temps, par exemple en écrivant $s_t(a_i) = c_j$.

Ce modèle s'applique à une variété d'approches fourmi comme certains des algorithmes Vertex-Ant-Walk (VAW) (Wagner *et al.*, 1999, appendice II-A), un algorithme d'exploration par Thrun (1992), LRTA* et Node Counting (Koenig *et al.*, 2001), ainsi que EVAP/EVAW (Glad *et al.*, 2008). Ils diffèrent principalement selon la sémantique de leur marquage.

2.3 EVAW

Cet article se focalise sur l'algorithme EVAW (Glad *et al.*, 2008) qui est typique des algorithmes fourmi dédiés à la patrouille. De plus, son formalisme est pratique pour dériver des résultats théoriques.

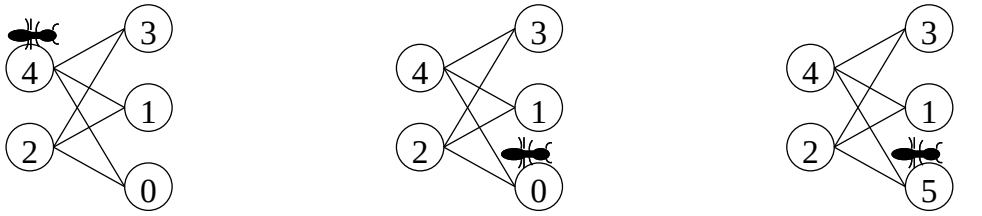
Le mécanisme de marquage de VAW/EVAW est caractérisé par : i) $M = \mathbb{N}$ et ii) un agent a_i marque sa cellule courante en fixant $m(s(a_i)) \leftarrow t$ (l'instant courant). Avec ce mécanisme de marquage, jusqu'à $|A|!$ états du système correspondent au même état de l'environnement (ce majorant n'est pas atteint quand plusieurs agents sont sur la même cellule). Comme détaillé dans l'algorithme 1 et comme illustré par la figure 1, à chaque pas de temps un agent EVAW 1) se déplace vers la cellule voisine de plus petit marquage et 2) marque cette cellule avec une valeur correspondant à l'instant courant t.

Ce premier algorithme est très simple, mais ne spécifie l'implementation lorsque de multiples agents l'éxécutent en parallèle. Nous faisons ici l'hypothèse qu'un agent observe son voisinage et se déplace

[2]Un graphe est fortement connecté si un chemin existe entre toute paire de sommets.

Algorithme 1 : L'algorithme EVAW exécuté par un agent

```
1 c ← cellule au hasard dans C ;                          /* initialisation */
2 pour chaque t = 1, ··· , ∞ faire
3 |   c ← arg min_{d∈N(c)} m(d) ;                          /* descente */
4 |   m(c) ← t ;                                           /* marquage */
```



a) état courant b) descente (line 3) c) marquage (line 4)

FIG. 1 – Une itération de l'algorithme EVAW (Alg. 1)

sans qu'aucun autre agent n'agisse en même temps, ce qui est plutôt réaliste en comparaison avec une implémentation sur des robots (ceci empêche que deux agents se retrouvent sur la même cellule). L'algorithme 2 donne une version séquentielle de l'algorithme multi-agent. On peut observer que deux lignes impliquent des choix non-déterministes :

- Ligne 3 – **Conflits** : Un *conflit* a lieu quand de plusieurs agents "souhaitent" se déplacer vers la même cellule mais que le résultat dépend de celui qui bougera le premier, comme sur la figure 2-a.
- Ligne 4 – **Hésitations** : Un agent a_i *hésite* quand il rencontre un choix entre au moins deux cellules dont les dernières visites étaient simultanées comme sur la figure 2-b.

En pratique, conflits et hésitations ont souvent lieu en même temps.

Algorithme 2 : EVAW séquentiel pour plusieurs agents

```
1 pour chaque i ∈ {1, ··· , |A|} faire  c_i ← cellule au hasard dans C
2 pour chaque t = 1, ··· , ∞ faire
3 |   pour chaque i ∈ {1, ··· , |A|} faire
4 |   |   c_i ← arg min_{d∈N(c_i)} m(d)
5 |   |   m(c_i) ← t
```

Dans une simulation, une autre façon de gérer les interactions entre agents est 1) de déplacer tous les agents d'un pas, puis 2) de les laisser tous marquer leur cellule (voir algorithme 3). Cette version "synchrone" d'EVAW (tous-les-agents-bougent puis tous-les-agents-marquent) est sans conflits, mais ne modélise pas correctement ce qui arriverait avec des robots puisqu'il est vraisemblable que deux agents se retrouvent ainsi sur la même cellule. Dans un tel cas, les deux agents vont continuer à effectuer les mêmes choix pendant un certain temps (jusqu'à la prochaine hésitation au moins), suivant le même chemin, ce qui a pour conséquence un mauvais comportement exploratoire.

2.4 Quelques résultats connus

Pour chacun des algorithmes mentionnés en section 2.2, il a été prouvé que le groupe de fourmis visite de manière répétitive chaque cellule de son environnement, c'est-à-dire qu'il effectue la tâche de patrouille (Wagner *et al.*, 1999; Koenig *et al.*, 2001). A notre connaissance, un majorant sur le temps de couverture est connu pour chacun de ces algorithmes sauf Node Counting. Dans le cas d'EVAW, nous notons ce majorant T_{vis}.

Des expérimentations ont montré que ces algorithmes ont de bonnes performances en termes d'oisiveté Koenig *et al.* (2001); Chu *et al.* (2007), même si les performances au pire cas de certains d'entre eux (par exemple Node Counting) peuvent être très mauvaises — typiquement sur des topologies complexes construites manuellement.

0	1	[2]
0	0	0
0	1	[2]

a- Deux agents en conflit

1	[8]	7	[8]
2	5	6	7
3	4	5	6

b- Un agent hésitant

FIG. 2 – Deux états du système correspondant à des situations incertaines (les positions des agents sont représentées par des crochets) : a) les deux agents souhaiteraient aller vers la cellule '0' située entre eux ; b) l'agent droit a le choix entre deux cellules '7' : celle à sa gauche —visitée pour la dernière fois par l'autre agent— et celle en dessous de lui —visitée pour la dernière fois par l'agent lui-même.

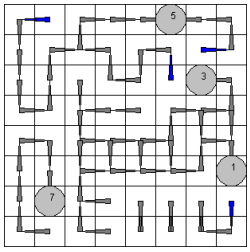
Algorithme 3 : EVAW synchrone centralisé pour plusieurs agents

pour chaque $i \in \{1, \cdots, |A|\}$ **faire** $c_i \leftarrow$ cellule au hasard dans C
pour chaque $t = 1, \cdots, \infty$ **faire**
  pour chaque $i \in \{1, \cdots, |A|\}$ **faire**
    $c_i \leftarrow \arg\min_{d \in N(c_i)} m(d)$
  pour chaque $i \in \{1, \cdots, |A|\}$ **faire** $m(c_i) \leftarrow t$

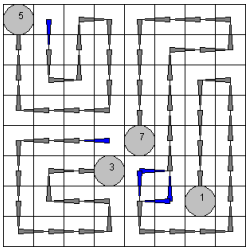
Il est aussi connu que VAW_0, EVAP et EVAW sont quasi-équivalents et peuvent converger vers des cycles stables — après une phase exploratoire — dans lequels chaque agent répète une séquence de cellule indéfiniment (Fig. 3). Seuls des résultats limités sont connus sur ces cycles, tels que : i) dans le cas mono-agent un agent qui suit un cycle hamiltonien[3] une fois le répétera indéfiniment (Wagner *et al.*, 1999) ; ii) dans le cas multi-agent des agents sur des cycles séparés ont nécéssairement des longueurs de cycles égales (Glad *et al.*, 2008).

3 Résultats de convergence

Dans cette section, nous caractérisons le comportement à long terme de l'algorithme EVAW. Cette étude de la convergence de l'algorithme EVAW suit une approche différente des travaux précédents puisque, plutôt que de regarder les cycles du point de vue des agents, nous les considérons du point de vue de l'environnement. En effet, désormais un *cycle* ζ est défini comme une séquence d'états du *système* qui se répète indéfiniment.



a) phase d'exploration



b) agents ayant atteint un cycle

FIG. 3 – Champ de phéromones avant et après convergence vers un cycle

3.1 Cas mono-agent

Théorème 3.1 (Convergence dans le cas mono-agent)

Etant donné un environnement $\mathcal{E}$ et un unique agent a, EVAW converge vers un cycle en temps fini.

[3] Un cycle hamiltonien visite chaque sommet du graphe exactement une fois.

Preuve: La preuve repose sur deux faits : 1) le nombre d'états accessibles du système est majoré et 2) le comportement de l'agent devient déterministe en temps fini.

1) Pour toute cellule c, le temps entre deux visites successives de c est majoré par T_{vis} (tel qu'expliqué en section 2.4). Avec des états "équivalents" $s^*(\mathcal{E})$, cela conduit à : $\forall c \in C,\ 0 \leq m(c) \leq T_{vis}$. L'ensemble des états accessibles de l'environnement est donc fini puisqu'il peut être majoré par $(T_{vis}+1)^{|C|}$. C'est aussi vrai pour les états du systèmes puisque, avec un seul agent, il y a bijection entre les états de l'environnement et les états du systèmes.

2) L'agent a ne peut pas visiter — et donc marquer — deux cellules en même temps. Ainsi, deux cellules ont le même marquage si et seulement si il est nul (c'est-à-dire qu'elles n'ont pas encore été visitées). Ainsi, dès que l'agent a couvert l'environnement — après au plus T_{vis} pas de temps — l'agent n'hésite plus : son comportement devient déterministe.

Ainsi, après l'instant T_{vis}, EVAW génère de manière déterministe une séquence récurrente d'états du système ($s_{t+1} = f(s_t)$). Parce le nombre d'état atteignables du système est fini, cette séquence converge vers un cycle en temps fini.

3.2 Cas multi-agent

Dans le cas multi-agent, il est toujours vrai que toutes les cellules sont visitées infiniment souvent. Un majorant sur le nombre d'états du système accessibles est ici $|A|! \times (T_{vis}+1)^{|C|}$.

Toutefois, avoir visité chaque cellule une fois ne signifie pas que la dynamique du système est désormais déterministe. Des conflits comme des hésitations peuvent encore être rencontrés. Il est possible de s'attaquer à ces sources de non-déterminisme de diverses manières :

- **Conflits :** On peut spécifier un protocole déterministe décidant de l'ordre dans lequel les agents agissent. Ce protocole peut dépendre par exemple des identifiants des agents ou de leur position relative dans l'environnement.
- **Hésitations :** On peut sélectionner une cellule en fonction de la direction suivie précedemment (un agent peut par exemple préférer ne pas tourner) ou en fonction de l'agent qui a laissé le dernier marquage (si cette information est disponible, un agent peut préférer sa propre trace).

La façon de rendre les décisions déterministes aura évidemment une influence sur le comportement de patrouille.

Théorème 3.2 (Convergence dans le cas multi-agent)

Le théorème est légèrement différent selon le comportement — déterministe ou non — des agents.

- *Si le système est rendu déterministe, il converge vers un cycle en temps fini (preuve similaire au théorème 3.1).*
- *Si le système n'est pas rendu déterministe (et que les tirages aléatoires effectués sont indépendants), son comportement peut être modélisé par une chaîne de Markov sur un espace d'état fini. Il aboutit nécessairement dans un sous-graphe absorbant, les cycles étant des cas particuliers de tels sous-graphes.*

Ces deux théorèmes confirment les observations de cycles et clarifient le comportement des agents EVAW. Toutefois, nous ne savons pas par exemple à quelle vitesse ces cycles ou graphes absorbants sont atteints, ou combien d'états ils incluents. En l'absence de résultats théoriques sur de tels points, nous examinons dans cet article des résultats expérimentaux.

4 Détection de cycles

Un point clef pour l'étude expérimentale de la convergence de comportements de patrouille stable est la capacité à détecter automatiquement des cycles ou des graphes absorbants.

Il existe de multiples algorithmes de détection de cycles pour systèmes déterministes, tels que l'algorithme du lièvre et de la tortue de Floyd (1967) ou l'algorithme de Brent (Brent, 1980). Cette section présente un nouvel algorithme — étendant l'algorithme du lièvre et de la tortue — que nous introduisons pour la détection de constructions stables dans des cas non-déterministes.

4.1 Le lièvre et la tortue (Floyd, 1967)

Considérons un ensemble X, une fonction $f : X \mapsto X$ et une séquence u définie par $u_0 \in X$ et, pour tout $n > 1$, $u_{n+1} = f(u_n)$.

L'algorithme de détection de cycles proposé par Floyd (1967) (Knuth, 1969, exercices 6-7, page 7) détecte si une séquence u aboutit à un cycle en comparant l'évolution de u (appelée "la tortue") et de v défini par $v_n = u_{2n}$ (appelée "le lièvre"). Comme montré sur l'algorithme 4, un cycle est détecté quand $u_n = v_n$, $n > 0$ (c'est-à-dire quand le lièvre et la tortue se rencontrent au même point).

Algorithme 4 : L'algorithme du lièvre et de la tortue

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

4.2 Tenir compte de dynamiques incertaines

4.2.1 Difficultés

Nous considérons l'utilisation de l'algorithme de détection de cycle du lièvre et de la tortue pour détecter si un groupe d'agents EVAW a atteint un point de convergence. Un premier point important est que le lièvre et la tortue devraient tous deux évoluer de manière identique, effectuant les mêmes choix aléatoires. Ceci requière qu'ils emploient tous deux la même séquence de nombres pseudo-aléatoires.

Considérant des versions "déterminisées" d'EVAW, un cycle peut survenir seulement après la phase de couverture (seule période pendant laquelle l'agent peut prendre des décisions au hasard), de sorte que l'algorithme 4 peut être employé sans problème.

Sinon, s'il reste des transitions non-déterministes, l'algorithme peut être trompé et s'arrêtera en cas :
- de vrai cycle,
- de sous-graphe absorbant, voir figure 4,
- de sous-graphe transitoire c'est-à-dire de sous-graphe dans lequel le système peut boucler pendant quelques temps mais dont il sortira nécessairement de manière définitive.

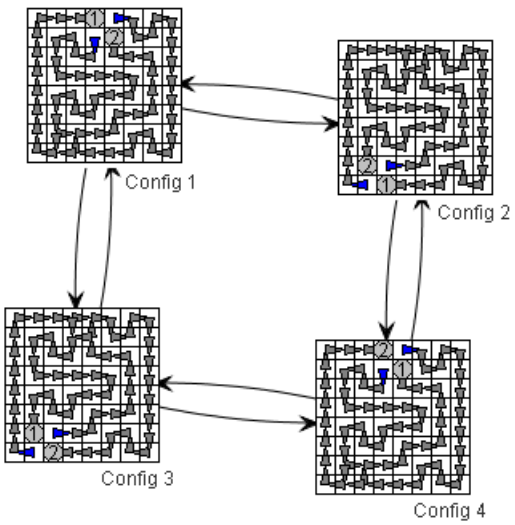


FIG. 4 – Un exemple de sous-graphe absorbant avec deux agents et quatre nœuds non-déterministes

Vérifier si ce qui a été détecté est effectivement un vrai cycle requière de simuler le système jusqu'à ce que le même état soit atteint sans avoir à effectuer des choix au hasard. Si ce n'est pas le cas, nous avons un

faux cycle qui est soit un sous-graphe transitoire soit un sous-graphe absorbant. La section suivante explique comment notre algorithme distingue les deux types de “faux” cycles.

4.2.2 Notre algorithme de détection

Faire la différence entre un sous-graphe absorbant et un phénomène transitoire est difficile. Développer le sous-graphe complet (c’est-à-dire tous les futurs possibles) est un processus long et gourmand en mémoire. Puisque l’objectif de la détection est d’analyser de multiples exécutions de l’algorithme, nous avons juste besoin de continuer à simuler le système jusqu’à ce qu’un cycle ou un sous-graphe absorbant soit atteint.

Ceci est accompli en construisant de manière incrémentale le sous-graphe visité après qu’un faux cycle ait été rencontré. Nous définissons un nœud du graphe comme un *état du système* dans lequel au moins un agent doit prendre une décision. Chaque arc du graphe représente tous les pas de simulation (états suivis de transitions déterministes) entre deux nœuds.

Quand un nœud est détecté (du fait d’un conflit ou d’une hésitation), l’algorithme calcule et mémorise tous ses fils. Ensuite, le prochain nœud à visiter est choisi, parmi ces fils, en suivant le comportement des agents. A intervalles réguliers, l’algorithme analyse le sous-graphe courant pour tester si le système est entré dans un cycle ou un sous-graphe absorbant.

Pour éviter les dépassements de mémoire, une profondeur maximale est fixée qui arrête la construction du graphe (et l’efface) quand un nombre maximum de nœuds ont été visités. La simulation continue alors — et la détection de cycles *via* l’algorithme du lièvre et de la tortue est redémarré — à partir de ce point.

5 EVAW amélioré

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.

5.1 Visualisation de chemins

Les marques numériques laissées par les fourmis forment un champ potentiel, de sorte qu’une idée intuitive est de visualiser le champ vectoriel du gradient associé, c’est-à-dire de dessiner, pour chaque cellule c, des flèches dans les directions de plus grande pente descendante (donc vers chaque cellule dans $\arg\min_{d\in N(c)} m(d)$). Toutefois, comme illustré par la figure 5-a, les chemins ne sont pas visibles avec cette représentation. Nous proposons pour cette raison une nouvelle représentation qui résoud ce problème : à chaque instant nous dessinons des flèches entre une cellule c et les cellules voisines atteintes en suivant les directions de plus *faible* pente ascendante (donc les cellules dans $\arg\min_{d\in N(c) s.t. m(d)\geq m(c)} m(d)$). Comme illustré par la figure 5-b, le résultat exhibe clairement les chemins et les cycles hamiltoniens quand ceux-ci se forment.

a) gradient de plus grande pente descendante b) gradient de plus faible pente ascendante

FIG. 5 – Deux représentations à base de champs de vecteurs

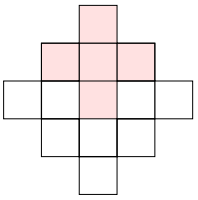
5.2 Observation

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 cellules qui correspondent à des minima locaux ($m(queue) \leq \min_{c \in N(queue)} m(c)$) 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 5-b), il peut suivre le chemin jusqu'au bout sans réorganiser tout 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 toujours 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 d'identifier des chemins, un agent peut facilement détecter une queue parmi ses voisins en utilisant une zone de perception étendue (13 cellules au lieu de 4 dans nos exemples). En effet, de manière à détecter ce motif un agent a seulement besoin de percevoir, en plus de sa perception courante, le voisinage de ses cellules adjacentes (c'est-à-dire toutes les cellules à une distance de Manhattan de 2) comme illustré par la figure 6.



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

FIG. 6 – Zone de perception d'un agent EVAW+ sur la grille

5.3 Accélérer la formation de cycles

Sur la base de ces observations et de la capacité de détecter des queues de cycle, nous étendons EVAW en forçant les agents à se déplacer vers une queue quand on en détecte une. Cette variante — non limitée à des environnements grilles — est appelée EVAW+ et montrée dans l'algorithme 5, où $queues(N(c))$ est l'ensemble de cellules voisines de c dans lesquelles on reconnait le motif caractéristique d'une queue.

Algorithme 5 : Une version améliorée d'EVAW

```
c ← cellule choisie au hasard dans C
pour chaque t = 1, ··· , ∞ faire
    si queues(N(c)) ≠ ∅ alors
        c ← arg min_{d∈queues(N(c))} m(d)
    sinon
        c ← arg min_{d∈N(c)} m(d)
    m(c) ← t
```

5.4 Comment maintenir la propriété de patrouille

Si cette amélioration de l'algorithme EVAW accélère la convergence (voir section 6), 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 7 a), c) et e) que l'agent n'a pas d'autre choix — en appliquant l'algorithme 5 — que de suivre des queues de chemins (flêches foncées). En conséquence, la cellule du coin en haut à gauche et ses deux voisines ne peuvent plus être visitées.

Ce “bug” est corrigé facilement 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 agée est plus grande qu'un seuil donné T_{thr} (par exemple $10 \times |C|$ pour éviter d'inhiber l'heuristique EVAW+), alors l'agent doit se déplacer vers cette cellule la plus agée. Avec cette modification, on peut à nouveau prouver que les agents EVAW+ couvrent leur environnement en temps borné (Wagner *et al.*, 1999, Th. 3, App. II-A) en utilisant $\Delta = T_{thr}$. De là, on a immédiatement que les résultats théoriques présentés pour EVAW dans ce papier sont aussi valides pour EVAW+.

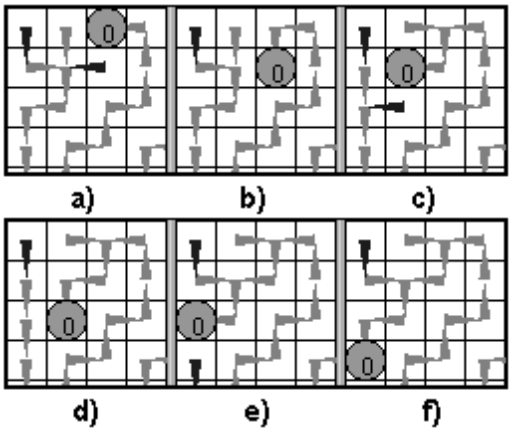


FIG. 7 – Perte de la propriété de patrouille

5.5 Un autre motif intéressant

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 certaines cellules reste instable. Nous appelons ce phénomène un *flip-flop*. La figure 8 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 flêche sortante alternera entre les directions haut et gauche.

FIG. 8 – 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.

Dans un environnement non-hamiltonien, les flip-flops sont nécessaires à la formation d'un regime stable. Ils permettent d'aller sur certaines cellules plus souvent que sur les autres. Une conséquence est qu'il n'est pas une bonne idée d'essayer de supprimer un flip-flop si l'on est dans un environnement non-hamiltonien : un autre flip-flop apparaîtra autre part.

Nous envisageons actuellement de mener des expérimentations avec une version d'EVAW intégrant un effaceur de flip-flops. L'objectif serait de réduire le nombre de flip-flops à son minimum (et d'arrêter de chasser ces motifs quand le nombre de flip-flops détectés se stabilise).

6 Expérimentations

Cette section présente les expérimentations conduites pour illustrer les résultats théoriques vus en section 3 et pour étudier quantitativement le comportement de convergence du système, en mesurant par

exemple le temps avant convergence et la probabilité de converger vers des cycles. Nous regardons 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. A notre connaissance de telles expérimentations n'ont pas encore été effectuées sur des algorithmes fourmi dédiés à la patrouille.

6.1 Cadre expérimental

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 Madkit/turtlekit (Michel *et al.*, 2005) avec l'ordonnancement par défaut de turtlekit, lequel contraint les agents à toujours agir dans le même ordre. Nous avons conservé cet ordonnancement, de sorte que les conflits soient résolus avec des agents dont la priorité dépend de leur ordre.

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 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 employée 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. Des vidéos d'exemple sont accessibles sur `http://patrolling-ants.domire.net/`.

6.2 Résultats

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 1 à 3 fournissent une partie des statistiques collectées.

6.2.1 Comportement de convergence

Notre premier objectif était de vérifier les résultats de la section 3 avec quatre paramétrages : $(1, 8, -)$, $(1, 8, d)$, $(3, 8, -)$ et $(3, 8, d)$. Comme escompté, les statistiques recueillies dans le tableau 1 montrent que le seul cas où la convergence vers des cycles n'est pas garanti (i.e. quand des sous-graphes absorbants sont trouvés) est celui de multiples agents non-déterministes ($(3, 8, +)$ et $(3, 8, -)$).

	# runs	cycles ham.	$longueur$ (ham.)	cycles non-h.	$\overline{longueur}$ (non-h.)	graphes absorbants	non convergence	$\overline{temps}$ $\overline{de\ conv.}$	$\sigma(temps$ $de\ conv.)$
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

TAB. 1 – Résumé des expérimentations sur la *déterminisation* d'EVAW.

6.2.2 EVAW vs EVAW+

Ensuite, considérons les résultats qualitatifs recueillis dans divers cadres expérimentaux — montrés dans le tableau 2 — ainsi que les figures 9, 10 et 11, lesquelles représentent les courbes de convergence (la probabilité d'avoir convergé vers un cycle après N itérations) en ne tenant compte que des vrais 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.	$longueur$ (ham.)	cycles non-h.	$\overline{longueur}$ (non-h.)	graphes absorbants	non convergence	$\overline{temps}$ $\overline{de\ conv.}$	$\sigma(temps$ $de\ conv.)$
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

TAB. 2 – Résumé des expérimentations conduites avec EVAW et EVAW+.

	# runs	cycles ham.	$longueur$ (ham.)	cycles non-h.	$\overline{longueur}$ (non-h.)	graphes absorbants	non convergence	$\overline{temps}$ $\overline{de\ conv.}$	$\sigma(temps$ $de\ conv.)$
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

TAB. 3 – Résumé des expérimentations conduites avec EVAW+ sur une grille 8x8 et avec un nombre croissant d'agents.

Forme des courbes de convergence

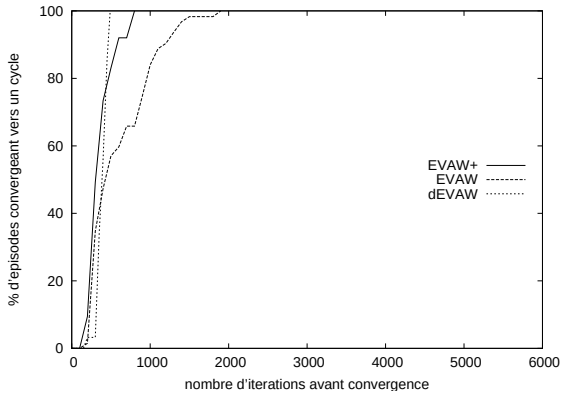
Comme on pouvait s'y attendre les diverses courbes de convergence ressemblent à des fonctions de répartition de distributions exponentielles. Cela signifie que, en moyenne, la probabilité de converger (de se retrouver sur un cycle) est la même à chaque instant. Elles diffèrent de ces fonctions de répartition 1) au début — pendant une phase d'exploration initiale — et 2) du fait que des "marches" apparaissent (en particulier dans le cas mono-agent) — c'est-à-dire parce qu'il y a des phases périodiques sans convergence. Ce phénomène périodique reste inexpliqué pour l'instant.

6.2.3 Densité d'agents croissante

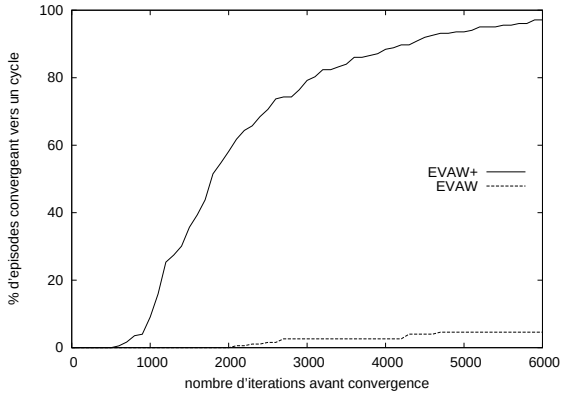
Finalement, nous avons mené des expérimentations avec un nombre croissant d'agents. Elles concernent seulement EVAW+ — parce que les autres algorithmes ne passent pas à l'échelle — utilisés sur une grille 8x8.

Les résultats sont présentés sur le tableau 3. L'observation principale est que aussi bien le pourcentage de convergence vers des sous-graphes absorbants que le temps de convergence augmentent avec le nombre d'agents, mais de manière irrégulière. Ces irrégularités coincident 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 (voir section 2.4).

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 absorbants. Pour ces expérimentations, nous avons limité la profondeur de recherche à 3 000 nœuds, restreignant donc la taille des sous-graphes absorbants détectables à 3 000 sommets. Ainsi, EVAW+ a effectivement convergé vers un sous-graphe absorbant mais l'algorithme de détection n'a pas été en mesure de le trouver. Comme EVAW, dans des cas identiques, n'a pas produit de tels résultats (pour $(3, 8, -)$, voir tableau 2), nous pouvons supposer que



a- sur une grille 8x8



b- sur une grille 14x14

FIG. 9 – Courbes de convergence pour un agent

l'heuristique a de fortes chances de produire des sous-graphes absorbants de très grande taille dans certaines situations.

6.2.4 Décroissance du nombre de cycles hamiltoniens

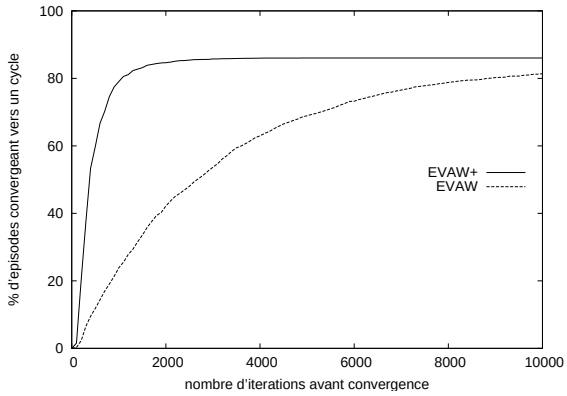
Finalement, on peut aussi observer que la probabilité de converger vers des cycles hamiltoniens décroît quand la taille de la grille croît. Il n'est pas clair que cela reflète une propriété de l'environnement — par exemple qu'il existe moins de cycles hamiltonien parmi les sous-graphes absorbants possibles — ou une propriété d'EVAW — que son comportement d'exploration est moins susceptible de trouver des cycles hamiltoniens.

7 Discussion / Conclusion

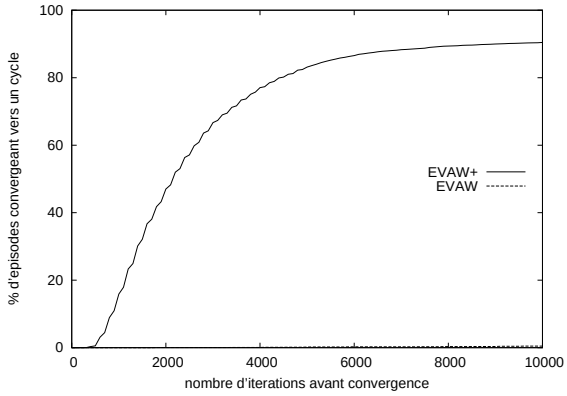
Dans cet article nous avons étudié le comportement de convergence d'un algorithme de patrouille à base de fourmis typique. Les théorèmes 3.1 et 3.2 montrent que le système s'auto-organise : les agents suivent des motifs répétitifs dont la stabilité est garantie dans le cas mono-agent ou si les agents sont déterministes. Ces résultats peuvent être généralisés aux autres algorithmes fourmis que nous avons cités, à part Node Counting (parce qu'aucun majorant de son temps de couverture n'est connu).

De plus, rappelons que les résultats théoriques ou les algorithmes dans cet article ne sont pas limités à des topologies de graphe particulières, même si la plupart des exemples et expérimentations utilisent des environnements grilles. D'autres pavages réguliers — triangulaires ou hexagonaux — pourraient être considérés, mais aussi irréguliers ou même de topologies changeantes — comme au sein d'un site web dont les pages et liens relatifs évoluent avec le temps.

Nous avons aussi étendu l'algorithme du lièvre et de la tortue pour détecter des cycles et des sous-graphes absorbants dans des systèmes dont la dynamique n'est pas déterministe. L'algorithme de détection de cycles



a- sur une grille 8x8



b- sur une grille 14x14

FIG. 10 – Courbes de convergence pour deux agents

de Brent (Brent, 1980) pourrait être étendu exactement de la même manière.

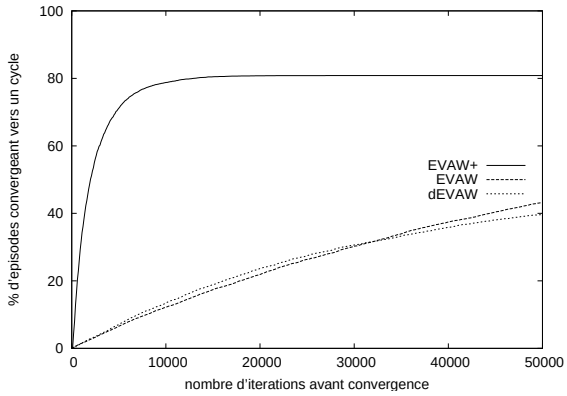
S'il est facile de distinguer des sous-graphes absorbants de sous-graphes transitoires — en construisant tous les futurs possibles depuis une certain point de la simulation — cela peut être coûteux en temps et en mémoire. Notre approche consiste à construire progressivement un graphe partiel représentant la séquence d'*états du système* parcourus et à vérifier régulièrement si le système a convergé vers un cycle ou un sous-graphe absorbant.

Une troisième contribution de ce travailest la proposition de l'algorithme EVAW+, lequel s'avère converger vers des cycles plus vite qu'EVAW, et plus souvent vers des cycles hamiltoniens quand ils existent. Notons que quand un cycle hamiltonien est atteint, il s'agit une solution optimale en termes de fréquence de visite (oisiveté). En outre, nous observons que les autres cycles, non hamiltoniens, sont souvent proches de l'optimum (la longueur est habituellement proche d'un multiple du nombre de cellules). Cela signifie que la performance en termes d'oisiveté est très bonne une fois la convergence atteinte.

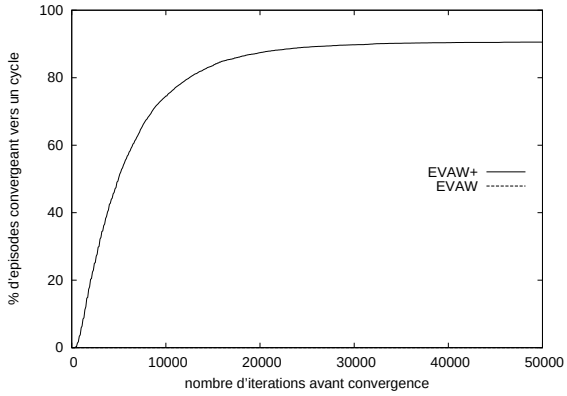
Il faudrait conduire davantage d'expérimentations pour évaluer EVAW et EVAW+ de manière plus approfondie, y compris pendant la phase d'exploration. En fait, trouver ou concevoir le "meilleur" algorithme de patrouille à base de fourmis requiert d'étudier l'influence d'une variété de paramètres : la règle de mise-à-jour des valeurs (/marquages), l'ordre des actions, la synchronisation entre agents, la déterminisation des conflits et hésitations... Il serait aussi intéressant de voir s'ils sont compétitifs avec d'autres approches, aussi bien en utilisation en-ligne, qu'hors-ligne.

Références

ALMEIDA A., RAMALHO G., SANTANA H., TEDESCO P., MENEZES T., CORRUBLE V. & CHEVALEYRE Y. (2004). Recent advances on multi-agent patrolling. In *Advances in Artificial Intelligence — Seventeenth Brazilian Symposium on Artificial Intelligence (SBIA'04)*, p. 474–483 : Springer-Verlag.



a- sur une grille 8x8



b- sur une grille 14x14

FIG. 11 – Courbes de convergence pour trois agents

ANDRADE R., MACEDO H., RAMALHO G. & FERRAZ C. (2001). Distributed mobile autonomous agents in network management. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'01)*.

BRENT R. (1980). An improved Monte Carlo factorization algorithm. *BIT*, **20**, 176–184.

CHO J. & GARCIA-MOLINA M. (2000). Synchronizing a database to improve freshness. In *Proceedings of the International Conference on Management of Data (SIGMOD'00)*.

CHU H., GLAD A., SIMONIN O., SEMPÉ F., DROGOUL A. & CHARPILLET F. (2007). Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation. In *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'07)*, p. 442–449.

FLOYD R. (1967). Non-deterministic algorithms. *Journal of the ACM*, **14**(4), 636–644.

GLAD A., SIMONIN O., BUFFET O. & CHARPILLET F. (2008). Theoretical study of ant-based algorithms for multi-agent patrolling. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*.

KNUTH D. E. (1969). *The Art of Computer Programming, vol. II : Seminumerical Algorithms*. Addison-Wesley.

KOENIG S., SZYMANSKI B. & LIU Y. (2001). Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, **31**(1–4).

MICHEL F., BEURIER G. & FERBER J. (2005). The TurtleKit simulation platform : Application to complex systems. In *Proceedings of the International Conference on Signal-Image Technology and Internet-Based Systems (SITIS'05)*.

THRUN S. (1992). *Efficient exploration in reinforcement learning*. Rapport interne CMU-CS-92-102, Carnegie Mellon University.

WAGNER I., LINDENBAUM M. & BRUCKSTEIN A. (1999). Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, **15**, 918–933.