Approche multi-agent pour l'exploration coordonnée d'un environnement labyrinthique 2D

Damien Pellier, Humbert Fiorino

Approche multi-agent pour l'exploration coordonnée d'un environnement labyrinthique 2D

Damien Pellier — Humbert Fiorino

Laboratoire Leibniz (CNRS - INPG - IMAG)
Équipe MAGMA - Bât D
46, Avenue Félix Viallet
F-38031 Grenoble Cedex
{Damien.Pellier, Humbert.Fiorino}@imag.fr

RÉSUMÉ. *Cet article présente une approche multi-agent résolvant le problème de «poursuite-évasion» pour un ou plusieurs robots mobiles coopératifs possédant une vision omnidirectionnelle. Cette approche possède l'originalité de mettre en œuvre une coopération réelle entre les agents fondée sur un partage de connaissances. Un algorithme complet pour une patrouille de robots reposant sur une décomposition de l'environnement en points critiques est présenté. Nous abordons son application dans le cadre de l'exploration coordonnée d'un environnement inconnu et nous présentons quelques résultats de simulation.*

ABSTRACT. *This paper introduces a multi-agent approach to solve the «pursuit-evasion» problem for one or more cooperative mobile robots that have omnidirectional vision sensors. This approach has the main characteristic of implementing a real cooperation between agents based on knowledge sharing. A complete algorithm for computing a motion strategy of robots is also presented. This algorithm is based on searching critical points of the environment. We introduce its application to the framework of coordinated exploration within an unknown environment, and a few solution strategies are shown.*

MOTS-CLÉS : *exploration coordonnée — coopération de robots — problème «poursuite-évasion»*

KEYWORDS: *coordinated exploration — robots cooperation — «pursuit-evasion» problem*

1. Introduction

Ces dernières années s'est développé un intérêt croissant pour l'usage d'approches multi-agents appliquées à la robotique, que ce soit dans le cadre de systèmes robotiques réactifs d'exploration (e.g. récolte de minerais [SIM 01, CHA 02]) ou dans le cadre plus général de robots capables d'opérer de façon autonome, adaptative et versatile en environnement humain [DRO 99].

Le domaine abordé dans cet article est à rapprocher des *systèmes robotiques mobiles coopératifs* tels qu'ils ont été présentés par Sellem et al. [SEL 00] dans le cadre de la répartition de la perception dans un système distribué de robots autonomes. Toutefois, le problème traité ici est proche du problème «proie-prédateurs» étudié pour la première fois par Benda et al. [BEN 86]. Rappelons que dans le problème «proie-prédateurs», les agents (i.e, la proie et les prédateurs) évoluent dans un environnement modélisé par une grille. A chaque unité de temps, les agents peuvent se déplacer horizontalement ou verticalement d'une case. Le but des prédateurs est de «capturer la proie» en l'encerclant en un minimum de temps. Les limites des approches développées pour ce problème apparaissent dès lors que l'on envisage un environnement plus réaliste et composé d'obstacles. C'est pourquoi nous nous intéressons au problème de «poursuite-évasion» dont le principe fondamental est de calculer une «ronde» pour un ou plusieurs robots (i.e., les agents poursuivants), *garantissant* qu'un «intrus» ou agent évadé soit nécessairement débusqué quels que soient ses déplacements (inconnus a priori). L'une des principales difficultés de ce problème, comparé avec la «simple» exploration d'un environnement, réside dans le fait qu'un agent évadé peut se déplacer dans une zone de l'environnement précédemment explorée par les poursuivants (cf. évadé figuré par un point blanc, figure 2). Les algorithmes permettant de résoudre ce problème peuvent trouver de larges champs d'application dans de nombreux domaines comme la robotique pour la localisation d'objectifs divers (e.g., recherche de mines ou surveillance d'un chenal par des engins sous-marins autonomes, exploration d'environnements hostiles par des drones ou des robots terrestres dans le cadre d'une Bulle Opérationnelle Aéroterrestre etc.) Au delà de la robotique, ces algorithmes peuvent aussi être mis en œuvre pour contrôler le déplacement des personnages d'un jeu vidéo ou d'avatars dans un monde virtuel.

Le problème de «poursuite-évasion» a été abordé sous de nombres angles, notamment celui de la théorie des jeux, de la théorie des graphes, de la géométrie computationnelle et de la robotique. Concernant la théorie des jeux, [LEV 92] a considéré le problème de poursuite-évasion dans le cadre d'agents coopératifs évoluant dans un environnement rudimentaire (i.e., une grille). Un certain nombre de résultats ont également été démontrés dans le cadre de la théorie des graphes (l'environnement est représenté par un graphe), dans lesquels les poursuivants et les évadés se déplacent de sommet en sommet jusqu'à ce qu'un poursuivant se trouve sur le même sommet que l'évadé [BIE 91, LAP 93, MAK 83, MEG 88, MON 88, PAR 76]. Dans le domaine de la robotique, le problème de poursuite-évasion a été introduit par [SUZ 92]. Depuis cette publication, un certain nombre de variations du problème ont été étudiées. Dans [LEE 00], des robots munis d'un faisceau de détection surveillent une

grille comportant une sortie qui doit constamment rester surveillée (variante du problème «proie-prédateur»). Dans [ICK 91], deux gardes se déplacent le long des frontières d'un environnement labyrinthique sans îlot (i.e., sans obstacle isolé) et doivent à tout moment rester visibles l'un de l'autre. Ses travaux furent par la suite repris par [HEF 93]. Le problème de poursuite-évasion a également été traité dans un environnement labyrinthique 2D quelconque pour des robots possédant une vision omnidirectionnelle [CRA 95, LAV 99, LAV 97, SUZ 92]. Cependant, on doit s'accommoder des restrictions suivantes : soit l'environnement peut être exploré par un seul poursuivant et on donne le parcours garantissant que l'évadé ne pourra en aucun cas s'échapper ; soit l'environnement implique l'utilisation de plusieurs poursuivants : on donne alors le parcours d'un *seul* poursuivant complété par des postes fixes pour les autres robots. La contribution de cet article réside dans la mise en œuvre d'un algorithme de coopération entre les poursuivants afin d'organiser une véritable exploration coordonnée de l'environnement.

Après avoir défini le problème (paragr. 2), nous présentons successivement un algorithme complet pour un poursuivant lorsque l'environnement le permet (paragr. 3), puis nous introduisons des mécanismes de coopération multi-agent pour un environnement quelconque (paragr. 4). Enfin, nous discutons des avantages de cette approche pour un environnement inconnu a priori (paragr. 5) et nous concluons sur quelques exemples testés (paragr. 6).

2. Définition du problème

Nous faisons l'hypothèse que l'agent «poursuivant» évolue dans un environnement labyrinthique 2D (cf. figure 1). Il possède une vision omnidirectionnelle (360°) et nous supposons, dans un premier temps, qu'il connaît l'ensemble de l'environnement. Par ailleurs, nous ne considérons que le cas d'obstacles polygonaux ce qui n'entraîne aucune perte de généralité par rapport au cas plus réaliste d'un environnement courbe [LAV 99].

Les agents «poursuivants» et «évadés» sont modélisés par des points dans un espace euclidien 2D. Soit F l'ensemble des points de l'espace qui n'appartiennent à aucun obstacle. Tous les poursuivants et les évadés doivent se trouver dans F. Soit $e(t) \in F$, la position de l'évadé au temps $t \geq 0$. Nous supposons que $e : [0, \infty) \rightarrow F$ est une fonction continue, et que l'évadé est capable de se déplacer à une vitesse arbitraire élevée (éventuellement «infinie»). La position initiale de l'évadé $e(0)$ et sa trajectoire e sont supposées inconnus des poursuivants.

Soit $\gamma^i(t)$, la position du i$^{\text{e}}$ poursuivant à l'instant $t \geq 0$. $\gamma^i : [0, \infty) \rightarrow F$ représente la trajectoire continue du i$^{\text{e}}$ poursuivant. Soit γ, l'ensemble des trajectoires continues des N poursuivants : $\gamma = \{\gamma^1, \ldots, \gamma^N\}$. Pour tout point $q \in F$, soit $V(q)$ l'ensemble de tous les points visibles de F en q (i.e., tous les segments reliant q à un point de $V(q)$ sont strictement inclus dans F). La «ronde» γ est une solution si pour chaque fonction continue $e : [0, \infty) \rightarrow F$, il existe un instant $t \in [0, \infty)$

et $i \in \{1, \ldots, N\}$ tel que $e(t) \in V(\gamma^i(t))$. Ceci implique que l'évadé ne peut pas échapper à ses poursuivants : il sera nécessairement débusqué à un instant donné.

Soit $H(F)$, le nombre minimum de poursuivants pour lequel il existe une trajectoire dans F garantissant qu'un évadé, s'il existe, sera trouvé. Les travaux de [GUI 99] donne une borne dans le pire des cas sur le nombre de poursuivants nécessaires ; F est composé de n côtés et h trous : $H(F) = O(h + log\ n)$ dans le cas d'un environnement comportant des «îlots» (i.e., des obstacles non connectés aux bords de l'environnement) et $H(F) = O(log\ n)$ dans le cas contraire.

On doit donc considérer deux problèmes distincts :

1) Déterminer $H(F)$;

2) Pour un F donné, trouver une ronde γ, mettant en œuvre $H(F)$ poursuivants.

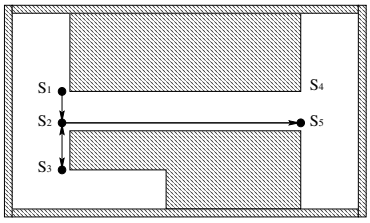

Figure 1. *Le • représente un agent poursuivant et sa trajectoire $\langle S_1, S_2, S_3, S_2, S_5 \rangle$ dans un espace euclidien 2D contenant des obstacles polygonaux.*

3. Algorithme complet pour un poursuivant

L'algorithme proposé ici ne calcule pas une ronde pour un nombre donné de poursuivants ; la ronde est construite pour s'approcher au mieux du nombre minimum de robots nécessaires à la «capture» de l'évadé. La détermination de $H(F)$ en fonction d'un environnement est un problème NP-difficile. Toutefois, il est possible de l'estimer par l'intermédiaire d'heuristiques, par exemple en effectuant un découpage cellulaire de l'environnement [LAT 91] et en plaçant un robot dans chaque cellule.

3.1. *Principe général*

L'algorithme que nous avons développé s'inspire de [LAV 97]. L'une des difficultés du problème de poursuite-évasion tient au fait qu'une zone déjà explorée par le poursuivant peut être à nouveau occupée par un évadé (cf. évadé figuré par un point blanc, figure 2). Par conséquent, la ronde du poursuivant doit, à chaque instant, tenir compte des déplacements possibles de l'évadé. Ainsi, toutes les régions de F qui

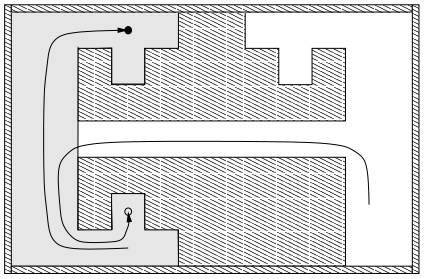

Figure 2. *Un évadé peut recontaminer une zone déjà explorée.*

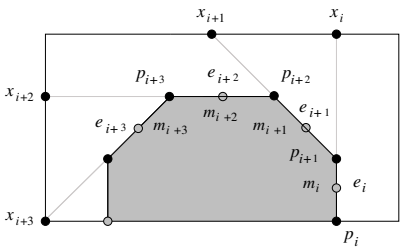

Figure 3. *Identification des points critiques d'un environnement.*

peuvent contenir potentiellement un évadé sont dites «contaminées», et «nettoyées» dans le cas contraire. Le but du poursuivant peut s'exprimer comme étant la décontamination de toutes les régions de l'environnement. Il faut donc découper l'espace F à décontaminer en régions convexes dans lesquelles un évadé peut se cacher.

Soit P un obstacle de l'environnement et ∂P l'ensemble des côtés de P. Considérons un sommet $p_i \in \partial P$. On dira qu'il s'agit d'un *point critique* si l'angle interne à P formé par les côtés e_{i-1} et e_i est inférieur à π (cf. figure 3).

Soit le point m_i défini comme le milieu de e_i. Si l'on trace le segment reliant m_i en passant par p_i jusqu'à x_i, intersection de ce segment avec un bord de l'environnement, on crée une région convexe passant par x_i et dans laquelle l'évadé ne peut pas se dissimuler. En revanche, l'évadé peut se dissimuler à l'extérieur de la «frontière» $[m_i, x_i]$. Par conséquent, on obtient par cette procédure un découpage de F en cellules convexes devant être explorées par le poursuivant. L'exploration de F peut donc se réduire à l'exploration de l'ensemble des points critiques CP qui déterminent sa décomposition. Ceci constitue l'une des différences entre notre approche et celle de [LAV 97].

Soit y^i la position du i^e poursuivant, $0 \leq i \leq N$. Un point critique $p_j \in CP$ est nettoyé si le point p_j est visible par au moins l'un des poursuivants situé en y^i et que les côtés adjacents de p_j sont entièrement visibles depuis y^i.

L'état de l'environnement en un point critique peut être modélisé par l'ensemble des points critiques visibles en ce point auxquels on associe une étiquette, «nettoyé» ou «contaminé». L'union de tous les états aux points y^i, où se situent les poursuivants, forme l'état global de l'environnement à l'instant $t \geq 0$ de la ronde. Le problème que doivent résoudre les agents poursuivants peut donc se résumer à trouver un chemin γ, qui passe par CP et conduit à un état final tel que tous les points de CP sont nettoyés.

3.2. *Construction de la ronde pour un seul poursuivant*

Notre algorithme pour un poursuivant construit une ronde optimale, en termes de distance parcourue. Le point de départ du poursuivant est choisi arbitrairement dans CP. L'algorithme se décompose en quatre étapes.

La première permet l'identification des points critiques de l'environnement (cf. figure 1 : les points critiques sont numérotés de 1 à 5). La procédure d'identification s'effectue en un coût linéaire du nombre de sommets de l'environnement (cf. figure 3).

Dans un deuxième temps, l'algorithme construit un *graphe de visibilité* fini G_v dont les nœuds sont les points critiques identifiés lors de la première étape. A partir d'un sommet donné, on crée une arête le reliant à un autre sommet chaque fois que ce dernier est visible depuis le point considéré. Ces arêtes sont étiquetées par la distance séparant les deux sommets (cf.figure 4 a)). Le graphe $G_v = (X, E)$ est un graphe orienté connexe tel que $X = \{x_1, \ldots, x_n\}$, avec n points critiques et $E = \{e_1, \ldots, e_m\}$, avec m arêtes. G_v donne pour tout point critique de l'environnement les déplacements possibles du poursuivant associé à la distance correspondante.

La troisième étape de l'algorithme est la construction du *graphe de surveillance* G_s de F à partir de G_v. L'état initial $E_{initial}$ de G_s est calculé en prenant comme point de départ le point donné en paramètre d'entrée de l'algorithme, p_{input}. $E_{initial}$ est composé de l'ensemble des points critiques auxquels on a adjoint une étiquette renseignant l'état de surveillance du sommet («contaminé» ou «nettoyé»). Pour chaque point critique p_i de l'état considéré, l'étiquette «nettoyé» est attribuée si et seulement si le sommet p_i est visible depuis p_{input} et les segments adjacents à p_i sont entièrement visibles depuis p_{input}. Pour chaque transition possible $T = \{t_1, \ldots, t_n\}$ avec n le nombre de points critiques visibles en p_{input}, un nouvel état E_{i+1} est construit de manière identique. Toutefois, lors du calcul des états suivants, il faut s'assurer que les points critiques nettoyés dans l'état courant le reste dans l'état suivant. Il faut par conséquent vérifier pour tous points critiques nettoyés dans l'état courant qu'il n'existe pas de chemin partant d'un point contaminé et arrivant à un point déjà nettoyé ; et que s'il en existe un, le segment qui les relie est visible depuis la transition t_i. Si la condition est remplie alors le point critique restera nettoyé. Dans le cas contraire, il deviendra contaminé. La transition est étiquetée par la distance précédemment calculée lors de la construction de G_s. L'algorithme s'applique récursivement pour tout nouvel état créé. Ainsi toutes les transitions possibles sont explorées, ce qui garantit que si une ronde γ pour un seul poursuivant existe, alors elle sera nécessairement trouvée.

La dernière étape de l'algorithme détermine la ronde optimale permettant la décontamination de F. Après avoir identifier tous les états finaux du graphe de surveillance G_s (i.e., dont tous les points critiques sont marqués nettoyés), pour chaque état final,

l'algorithme de Dijsktra est appliqué afin de déterminer la ronde optimale reliant les sommets finaux à $E_{initial}$. Il est à noter que l'identification des états finaux peut se faire lors de la construction de G_s.

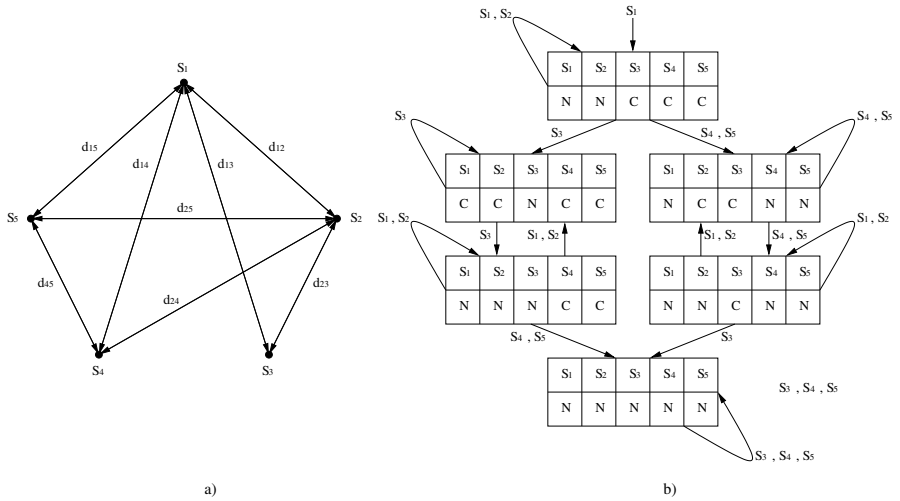

Figure 4. *a) montre le graphe de visibilité et b) le graphe de surveillance associés à la figure 1 : «N» indique que le sommet est nettoyé; «C» qu'il est contaminé.*

4. Mise en œuvre de la coopération

Nous avons vu précédemment l'algorithme complet pour un poursuivant. Toutefois, la construction du graphe G_s peut être faite sans pour autant qu'un état final soit identifié. Dans ce cas précis, on est donc sûr que l'environnement doit être exploré par au moins un poursuivant supplémentaire.

4.1. *L'identification des «points de délégation»*

Lorsqu'un poursuivant ne suffit pas à la décontamination de l'environnement, il faut reconstruire un graphe de surveillance en interdisant les mouvements des poursuivants pouvant mener à une recontamination complète ou partielle de l'environnement. Cette propriété du graphe G_s est obtenue en vérifiant que les états successeurs construits à partir d'un état donné respectent la condition suivante : si au moins l'un des points critiques nettoyés constituant l'état considéré reste nettoyé dans l'état successeur alors cet état est considéré comme valide. On appelle *point de délégation* un point critique qu'un poursuivant ne peut quitter sans risquer une recontamination ; par conséquent, il faut *déléguer* à cet endroit la tâche de surveillance à autre poursuivant. Un tel point P_d est trouvé si et seulement si on ne peut pas créer un nouvel état en

partant de l'état considéré. P_d est un puits du graphe G_s. De part la construction du graphe G_s en largeur d'abord, il est possible que plusieurs points de délégation soient identifiés. Pour chaque point de délégation, l'algorithme va construire la trajectoire permettant de l'atteindre comme s'il s'agissait d'un état final à atteindre. Le choix d'un point de délégation lorsque plusieurs candidats sont possibles, est effectué par rapport au critère de la distance parcourue : le point P_d le plus proche en termes de trajectoire de décontamination est retenu. Les poursuivants peuvent donc progresser jusqu'au point de délégation P_d tout en décontaminant partiellement l'environnement.

4.2. *La délégation de tâches de surveillance*

Au point de délégation, les robots doivent être capables de se concerter dans le but de décomposer la tâche collective de décontamination en un ensemble de sous-tâches. On peut envisager plusieurs cas : les robots peuvent prendre la décision de décontaminer chacun une zone particulière de l'environnement ; au contraire, un robot (le guetteur) peut être assigné à la surveillance de points critiques, garantissant qu'aucun évadé ne pourra se déplacer d'une partie contaminée à une partie décontaminée de l'environnement. Ainsi les poursuivants restants peuvent décontaminer les zones susceptibles de contenir encore un évadé, de manière indépendante. A partir du graphe de visibilité G_v, le poursuivant bloqué au point de délégation tente de construire une coupe du graphe. Le poursuivant responsable de la décomposition en sous-tâches alloue aux autres poursuivants une composante connexe du graphe de visibilité sur laquelle chaque poursuivant va appliquer de manière récursive l'algorithme complet pour un poursuivant. En fait, le poursuivant en charge de l'allocation des tâches tente d'effectuer plusieurs découpages de l'environnement.

La première coupe dite «simple», est réalisée en supprimant du graphe de visibilité G_v le point de délégation P_d. Si la suppression de P_d rend G_v non connexe, les poursuivants se scindent en deux groupes pour nettoyer la sous partie de l'environnement représentée par chaque composante connexe du graphe de visibilité G_v (cf. figure 6). Si la suppression du point P_d de G_v ne transforme pas G_v en graphe non connexe, l'algorithme tente d'effectuer une coupe dite «complexe» en supprimant de G_v, P_d et tous les points critiques visibles depuis P_d (cf. figure 5). Dans le cas d'une coupe complexe du graphe, un robot guetteur se positionne au point de délégation. Si toutes les sous parties de l'environnement générées ne peuvent pas être décontaminées par un seul poursuivant, le robot restant décontamine chaque sous parties de l'environnement jusqu'à ce qu'il n'en reste plus que deux. Dans ce cas, le comportement des poursuivants est le même que dans le cas d'une coupe simple : le robot guetteur et l'autre poursuivant décontamine en parallèle les deux dernières composantes de l'environnement. Si l'ensemble des sous parties de l'environnement ne peut pas être décontaminé par un seul poursuivant, le poursuivant décontamine en priorité les sous parties qui le peuvent. Si une composante nécessite plus d'un poursuivant alors le poursuivant se rend au prochain point de délégation et applique à nouveau l'algorithme de déléga-

tion. La décontamination prioritaire des composantes connexes pouvant être nettoyées par un poursuivant permet d'envisager que le guetteur pourra à nouveau reprendre sa place en tant que poursuivant et ainsi venir en aide le cas échéant aux autres robots. Ainsi, dans cette approche fondée sur une exploration coordonnée, les guetteurs ne le sont que ponctuellement et si nécessaire.

4.3. *La synchronisation des robots*

Dans le paragraphe précédent, nous avons vu comment les poursuivants réalisaient la délégation de tâches de surveillance au niveau fonctionnel. Toutefois, un problème persiste. Imaginons un environnement tel que celui de la figure 7. Cet environnement ne peut pas être décontaminé par un seul poursuivant. L'algorithme de délégation de tâche s'applique, l'un des poursuivants devient guetteur. L'autre poursuivant tente de décontaminer la sous partie restante (cf. figure 7 b)) de l'environnement. Mais cette composante ne peut être décontaminée par un poursuivant. Le poursuivant au point de délégation va donc demander de l'aide au guetteur en lui donnant les points critiques de l'environnement qu'il souhaiterait voir surveiller afin de pouvoir continuer sa décontamination (cf. figure 7 c)). Le guetteur doit être capable de savoir s'il peut venir en aide et dans quelle mesure son déplacement pour réaliser la tâche qui lui est demandée peut entraîner une recontamination partielle ou totale de l'environnement. Dans un cadre plus général, un poursuivant lorsqu'il détecte un point de délégation peut demander de l'aide aux autres poursuivants dont la tâche est terminée ou aux guetteurs. Le poursuivant demandant de l'aide effectue l'agrégation des connaissances partielles des différents poursuivants concernant l'environnement exploré puis le transmet à chaque poursuivant pouvant potentiellement l'assister. En fonction de l'état global calculé, les poursuivants cherchent la plus courte des trajectoires qui mène vers l'un des points critiques à partir desquels ils peuvent percevoir les points demandés par le poursuivant bloqué. Les poursuivants dont la tâche est finie n'ont plus de contrainte et peuvent être réalloués à une autre tâche. Concernant les guetteurs, ils doivent trouver une trajectoire qui passe uniquement par des points déjà nettoyés de l'environnement afin de ne pas le recontaminer. Une fois cette trajectoire calculée, elle est envoyée au poursuivant bloqué qui choisit la meilleure trajectoire en fonction d'un critère de distance. Le poursuivant choisi se positionne au point de surveillance. Le poursuivant bloqué peut alors continuer sa décontamination. L'algorithme est terminé lorsque tous les poursuivants ont terminé de réaliser leurs tâches.

5. Discussion sur l'algorithme

Cet algorithme présente un certain nombre d'avantages par rapport aux travaux présentés en introduction (notamment [LAV 97]). En particulier, il est possible de l'appliquer dans le cas d'un environnement inconnu a priori.

En effet, la construction du graphe de surveillance nécessite dans l'algorithme présenté la connaissance totale du graphe de visibilité qui modélise l'environnement. Mais dans le cas où l'environnement est inconnu, la construction du graphe de surveillance peut se faire de manière itérative en fonction des nouveaux points critiques découverts au fur et à mesure de l'exploration : il est ainsi possible d'explorer l'environnent et de construire le graphe de surveillance conjointement. Lorsqu'un point de délégation est rencontré, les poursuivants vont agir comme précédemment et coordonner leur exploration. Ainsi, des gains importants en termes de distance parcourue peuvent être obtenus. Ceci est un avantage important par rapport aux travaux de [RAJ 01] dans lesquels les poursuivants sont obligés de passer par une phase d'exploration *puis* une phase de surveillance.

Enfin, la complexité de l'algorithme pour un poursuivant est semblable à celle de [LAV 97] : $O(2^n)$ avec n le nombre de points critiques de l'environnent. Toutefois, l'explosion combinatoire est contenue car, en pratique, le nombre de points critiques reste faible ; notre approche présente l'avantage de distribuer la tâche d'exploration sur plusieurs agents. Ainsi, chaque agent traite une sous partie du problème en répartissant la charge de calcul. Il est à noter qu'il est possible d'optimiser le temps de calcul par plusieurs méthodes au détriment de la ronde optimale. En effet, on peut arrêter la construction du graphe de surveillance dès lors qu'un état final a été trouvé. Dans ce cas rien ne garantit que cet état soit celui qui permette une décontamination optimale par rapport à un critère de distance. Il est également envisageable de définir des heuristiques permettant de diminuer de manière significative le nombre d'états explorés. Lors de la construction du graphe de surveillance, l'algorithme présenté explore toutes les transitions possibles depuis un état donné. L'idée est ici de choisir une seule transition déterminée à partir de trois critères : la fréquence de passage dans par cette transition ; le type de transition (transition menant à un «cul-de-sac») ou la distance associée avec cette transition.

6. Simulation et exemples

La mise en œuvre de l'algorithme a été réalisée en JAVA. Les figures 5, 6 et 7 présentent trois exemples testés. La figure 5 met en évidence un exemple pour deux poursuivants plus complexe dans lequel il est possible d'observer le comportement des poursuivants dans le cas d'une délégation avec une coupe complexe du graphe de visibilité fournissant quatre composantes connexes. La figure 6 donne un exemple pour deux poursuivants dans lequel la délégation de tâche est effectuée par une coupe simple du graphe de visibilité. Pour terminer la figure 7 montre la coopération qui est effectuée entre un guetteur et un poursuivant.

NOTE. — Dans tous les exemples, les • représentent les poursuivants, les zones hachurées leur perception sur l'environnement, les zones blanches les parties décontaminées et les zones noires les parties contaminées.

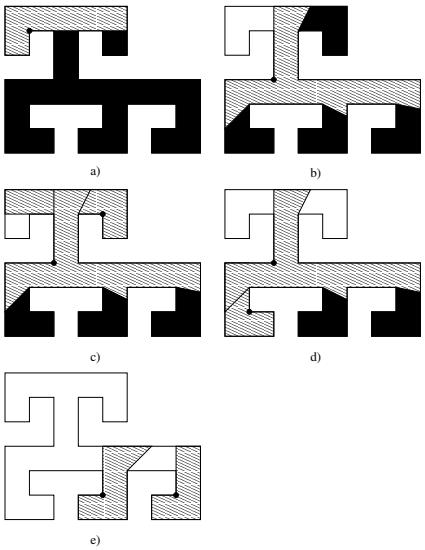

Figure 5. *Cet exemple montre la trajectoire emprunté par deux poursuivants dans une environnement simplement connecté.*

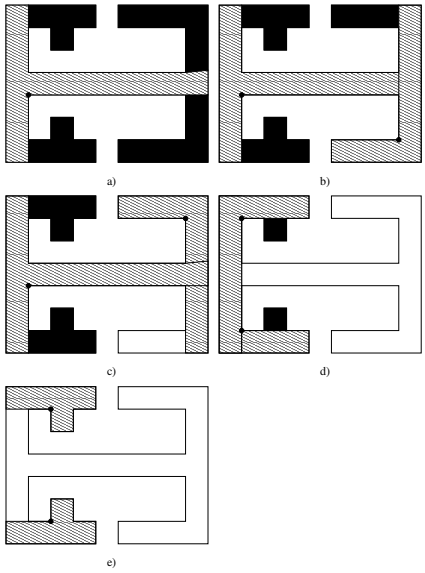

Figure 6. *Cet exemple met en évidence le comportement de deux poursuivants lorsque le point de délégation autorise une coupe simple de l'environnement.*

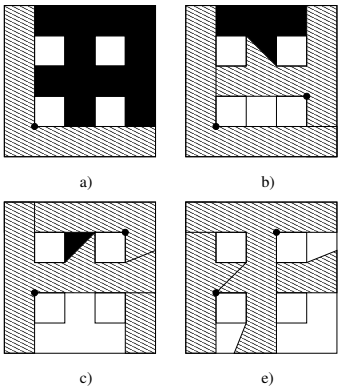

Figure 7. *Cet exemple la coopération en un poursuivant bloqué et un guetteur*

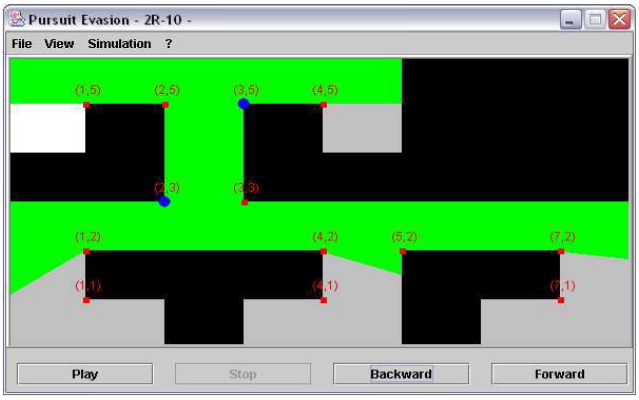

Figure 8. *Capture de l'écran de simulation de poursuite évasion*

7. Conclusion

Dans cet article, nous avons présenté un algorithme résolvant le problème de poursuite évasion dans un environnement labyrinthique comportant des obstacles et plusieurs contributions apportées par les systèmes multi-agents ont été utilisées : la délégation de tâche, des mécanismes de coopération et la construction d'un état global du système. Cela nous a permis de mettre en œuvre une véritable exploration coordonnée par une patrouille de robots et d'envisager sa généralisation à un environnement inconnu a priori. D'autre part, nous avons montré qu'il était possible d'utiliser en pratique cette approche malgré sa complexité dans la mesure où la charge de calcul est répartie sur plusieurs agents. Toutefois, l'optimisation de la ronde globale des poursuivants reste un problème ouvert. Il serait également intéressant d'étudier la mise en œuvre de cette approche dans le cas de poursuivants possédant une vision restreinte de leur environnement.

8. Bibliographie

[BEN 86] BENDA M., JAGANNATHAN V., DODHIAWALA R., « On optimal cooperation of knowledge sources - an empirical investigation », rapport n° BCS-G2010-28, July 1986, Boeing Advanced Technology Center, Boeing Computing Services.

[BIE 91] BIENSTOCK D., SEYMOUR P., « MonotocityCity in graph searching », *J. Algorithms*, vol. 12, 1991, p. 239-245.

[CHA 02] CHADÈS I., SCHERRER B., CHARPILLET F., « A Heuristic Approach for Solving Decentralized-POMDP : Assessment on the Pursuit Problem », *ACM Symposium on Applied Computing*, Madrid, Spain, 2002.

[CRA 95] CRASS D., SUZUKI I., YAMASHITA M., « Searching for a mobile intruder in a corridor - the open edge variant of the polygon search problem », *International Journal of Computational Geometry and Applications*, vol. 5(4), 1995, p. 397-412.

[DRO 99] DROGOUL A., PICAULT S., « MICROBES : Vers des collectivités de robots socialement situés », *Actes des JFIADSMA'99*, 1999, p. 265-278.

[GUI 99] GUIBAS L., LATOMBE J. C., LAVALLE S. M., LIN D., MOTWANI R., « A visibility-based pursuit-evasion problem », *International Journal of Computational Geometry and Applications*, vol. 9, 1999, p. 471-493.

[HEF 93] HEFFERNAN P. C., « An optimal algorithm for the two guards problem », *Proc. ACM Symp. Comp. Geom.*, 1993, p. 348-358.

[ICK 91] ICKING C., KLEIN R., « The two guards problem », *Proc. ACM Symp. Comp. Geom.*, 1991, p. 166-175.

[LAP 93] LAPAUGH A. S., « Recontamination does not help to seach a graph », *J. ACM*, vol. 40(2), 1993, p. 224-245.

[LAT 91] LATOMBE J. C., « Robot Motion Planning », *Kluwer Academic Publishers*, , 1991.

[LAV 97] LAVALLE S. M., LIN D., GUIBAS L. J., LATOMBE J. C., MOTWANI R., « Finding an unpredictable target in workspace with obstacles », *Proc. IEEE International Conference on Robotics and Automation*, 1997, p. 737-742.

[LAV 99] LAVALLE S. M., HINRICHNSEN J., « Visibility-Based Pursuit-Evasion: The Case of Curved Environments », *Proc. IEEE International Conference on Robotics and Automation*, 1999.

[LEE 00] LEE J. H., PARK S. M., CHWA K. Y., « Searching a polygonal room with a door by 1-searcher », *International Journal of Computational Geometry and Applications*, vol. 10, 2000, p. 201-220.

[LEV 92] LEVY R., ROSENSCHEIN J. S., « A game theoretic approach to distributed artificial intelligence and the pursuit problem », 1992.

[MAK 83] MAKEDON F., SUDBOROUGH I. H., « Minimizing width in linear layout », *Proc. 10th ICALP, LNCS 154*, 1983, p. 478-490.

[MEG 88] MEGIDDO N., HAKINI S. L., GAREY M. R., JOHNSON D., PAPADIMITRIOU C. H., « The complexity of searching a graph », *J. ACM*, vol. 35(1), 1988, p. 18-44.

[MON 88] MONIEN B., SUDBOROUGH I. H., « Min cut is NP-complete for edge weighted graphs », *Theoretical Computer Science*, vol. 58, 1988, p. 209-229.

[PAR 76] PARSONS T. D., « Poursuit-evasion in a graphe », *Therory and Application of Graphs*, Springer-Verlag, Berlin, 1976, p. 426-441.

[RAJ 01] RAJKO S., LAVALLE M., « A Pursuit-Evasion BUG Algorithm », *Proc. IEEE International Conference on Robotics and Automation*, 2001.

[SEL 00] SELLEM P., LUZEAUX D., « Répartition de la perception dans un système distribué de robots autonomes », *Actes des JFIADSMA'99*, , 2000, p. 181-190.

[SIM 01] SIMONIN O., « Le modèle satisfaction-altruisme : coopération et résolution de conflits entre agents situés réactifs, application à la robotique », PhD thesis, Université Monpellier II - Sciences et Techniques du Languedoc, 2001.

[SUZ 92] SUZUKI I., YAMASHITA M., « Searching for mobile intruder in a polygonal region », *SIAM Journal on computing 21*, vol. 21(5), 1992, p. 863-888.