REPUBLIQUE ALGERIENNE DIMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SURERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE

Département d'Automatique

Projet de fin d'études

En vue de l'obtention du diplôme d'ingénieur d'état et du diplôme de Master en Automatique

Thème :

Planification et optimisation de trajectoire d'un robot mobile par les algorithmes évolutionnaires

Réalisé par:

Houcine TAZEBINTE

Khaled LAMECHE

Proposé et dirigé par :

Pr . H. CHEKIREB

Juin 2012

Ecole Nationale Polytechnique, 10, AV. Hassen Badi, El-Harrach, Algérie

Remerciements

En premier lieu, nous remercions dieu tout puissant de nous avoir donné le courage, la volonté et la patience pour réaliser ce travail.

Nous remercions nos parents, qui nous ont apporté l'aide et le soutien tout au long de nos études.

Nous tenons à exprimer tous nos vifs remerciements et nos profondes Gratitudes à notre promoteur : Pr H.CHEKIREB d'avoir proposé et dirigé ce travail et pour ses conseils scientifiques, sa disponibilités, aides et bonnes humeurs.

Nos remerciements vont également aux enseignants du département d'Automatique et des sciences fondamentales, qui ont contribué à notre formation d'ingénieur.

Enfin, nous remercions les membres du jury qui ont accepté d'évaluer notre travail.

Dédicace

Je dédie ce travail à mes très chers parents qui m'ont toujours apporté leur amour et leur soutien pour affronter les difficultés de la vie.Qu'ils trouvent ici le témoignage de ma reconnaissance la plus dévouée.

A mes chers frères, et à ma chère sœur.

A tous les membres de ma famille : oncles, tantes, cousins et cousines.

A tous les automaticiens.

A tous mes amis en particulier mon binome Hocine, Ayoub, Mohamed, Sohaib, Tahar, Ossama, Djalel, Mokrane, Hamza et les autres.

Lameche Khaled

Dédicace

Je dédie ce travail :

À mes très chers parents pour les sacrifices et les encouragements qu'ils n'ont pas cessé de me Conférer. Que dieu les gardera éternellement heureux.

À mes sœurs.

À mon frère et son épouse.

À mes chers oncles et tantes.

À mes cousins et cousines.

À toute ma grande famille.

À tous mes amis.

À tous qui m'ont connu et aidé de près ou de loin pour réaliser ce modeste travail.

Tazebinte Houcine

Table des matières

Table des figures

Liste des tableaux

Introduction générale

La robotique est un domaine de recherche qui se situe au carrefour de l'intelligence artificielle, de l'automatique, de l'informatique ; cette interdisciplinarité est à l'origine d'une certaine complexité. Des applications dans des domaines aussi variés que l'industrie manufacturière, le spatial, l'automobile ou plus récemment les loisirs et le secteur médical, démontrent aujourd'hui l'intérêt économique et social de ces recherches.

L'objet de la robotique est l'automatisation de systèmes mécaniques. En dotant le système de capacités de perception, d'action et de décision, l'objectif est de lui permettre d'interagir rationnellement avec son environnement, et de façon autonome.

La robotique mobile autonome vise plus spécifiquement à concevoir des systèmes capables de se déplacer de façon autonome. Les applications directes se situent notamment dans les domaines de l'automobile, de l'exploration planétaire ou de la robotique de service par exemple. De nombreuses applications restent à découvrir.

Notre mémoire a pour cadre général la planification de trajectoire des robots mobiles, notre but est de rendre le robot capable d'explorer de plus grandes aires et de parcourir de plus longues distances sur des terrains partiellement connus, Cet objectif sera plus facilement accompli, si le robot est équipé d'un planificateur de chemin fiable et rapide, et qui minimise les risques engagés pour le robot.

Ce projet cherche donc à fournir une solution au problème de la planification de chemin tout en considérant la topographie du terrain. L'objectif est de fournir au robot une trajectoire reliant le point de départ et le point de destination tout en évitant la collision avec les obstacles dans un environnement statique, de minimiser la longueur de chemin et le temps de planification, de garantir la faisabilité du chemin tracé par le planificateur sur longues distances.

La planification étant un domaine vaste, on n'intéresse pas ici à la perception de l'environnement et l'obtention de l'information : les robots dans nos applications disposent d'une connaissance parfaite et absolue du monde dans lequel ils évoluent.

Nous présentons dans cette thèse l'adaptation de deux algorithmes évolutionnaires ; l'algorithme de colonie de fourmis (ACO) et l'algorithme des essaims particulaires (PSO) au problème de planification de chemin. Et pour évaluer ces algorithmes on les compare avec l'algorithme de planification A* le plus implémentés dans les robots mobile.

Notre mémoire comporte huit chapitres :

Le premier chapitre est consacré à des généralités sur la planification de trajectoire. Aussi, nous donnons un bref aperçu sur les différentes approches qu'existent jusqu'à l'état actuel.

Dans le second chapitre, nous décrivons tout d'abord le cadre de l'optimisation difficile et les algorithmes métaheuristiques, puis nous présentons le concept des algorithmes évolutionnaires et nous décrivons leur principes : l'inspiration biologique et le processus d'auto-organisation.

Le troisième chapitre est dédié au concept de l'optimisation par colonies de fourmis. Au début, nous énumérons les similarités et les différences entre les fourmis réelles et artificielles, Nous citons par la suite, quelques expériences qui ont été faites sur des fourmis réelles pour comprendre leur comportement lors de la recherche de la nourriture. Puis nous présentons l'algorithme (ACO) avec ces extensions.

Dans le quatrième chapitre, nous présentons la méthode d'optimisation par essaims particulaires (Particle Swarm Optimization), qui est une méthode d'optimisation récente. Nous décrivons son principe, sa formulation, les paramètres de l'algorithme d'optimisation.

Le cinquième chapitre concerne la description de l'espace de travail dans lequel on a adapté les algorithmes de planification ACO, A*, PSO.

Dans le sixième chapitre, nous décrivons l'algorithme de planification par ACO et nous présentons les résultats de simulation en mettant le robot mobile dans des différentes situations, puis nous proposons un algorithme de lissage du chemin, et on vérifie son efficacité par simulation.

Le septième chapitre est réservé à l'algorithme de recherche de graphe A* adapté au problème de planification chemin dans une grille et la façon de le programmer.

Dans le huitième chapitre, nous allons proposer un algorithme d'optimisation de trajectoire par PSO qui sert à améliorer les performances des chemins crées par d'autre algorithmes, dans notre cas ACO et A*, il a pour but d'améliorer la longueur et la rugosité de la trajectoire.

Enfin, une conclusion générale résume les principaux résultats auxquels nous avons abouti et les perspectives qui seraient susceptibles d'aboutir à de nouvelles améliorations.

Chapitre 1

Généralités sur la planification de trajectoire des robots mobiles

1.1 Introduction

A cause de l'intérêt croissant pour 1'exploration planétaire, les robots mobiles sont appelés à avoir une très grande autonomie. Le besoin d'une intervention humaine dans les décisions de planification de chemin cause une énorme perte de temps et de ressources du robot.

La navigation d'un robot consiste à déterminer une suite de coordonnées que ce dernier doit suivre pour atteindre une destination donnée. On distingue en principe deux types de taches complémentaires qui permettent au robot de naviguer :

- la génération de trajectoire ;
- le suivi de trajectoire.

En effet, ces deux types de tache doivent être distingues, le suivi de trajectoire permet de commander le robot à passer par des points déterminés, indépendamment de la manière dont ont été générés ces points. La génération de trajectoire consiste en la génération des points de passage, indépendamment de la manière dont le robot s'y rendra.

Lorsque l'environnement du robot est entièrement connu, des méthodes permettent de générer une trajectoire et de contrôler le robot de façon à suivre cette trajectoire. Lorsque l'environnement n'est pas connu à l'avance, il n'est pas possible de séparer la génération de trajectoire du suivi de trajectoire : la trajectoire est générée au fur et à mesure que le robot évolue dans l'environnement.[1]

1.2 Génération de trajectoire en environnement connu

La génération de trajectoire consiste à interpréter l'ensemble des informations disponibles afin de spécifier une série d'actions à imposer au robot. Ces actions peuvent être exprimées sous plusieurs formes : elles peuvent être un ensemble de points de passage ou une série de vitesses que le robot doit prendre à chaque instant. Les informations dont le robot a besoin pour planifier une trajectoire dépendent directement de l'environnement. La connaissance de l'environnement signifie la connaissance de l'emplacement et de la forme des obstacles.

1.2.1 Planification de trajectoire

Le rôle d'un planificateur de chemins consiste donc à augmenter l'autonomie du robot en lui permettant de se déplacer d'une position initiale vers une position finale arbitraire tout en évitant les collisions avec les obstacles présents dans son environnement comme le montre sa représentation de fonctionnement (figure 1.1).Cette problématique a été dans l'intérêt de chercheurs dans différents domaines durant les dernières décennies.

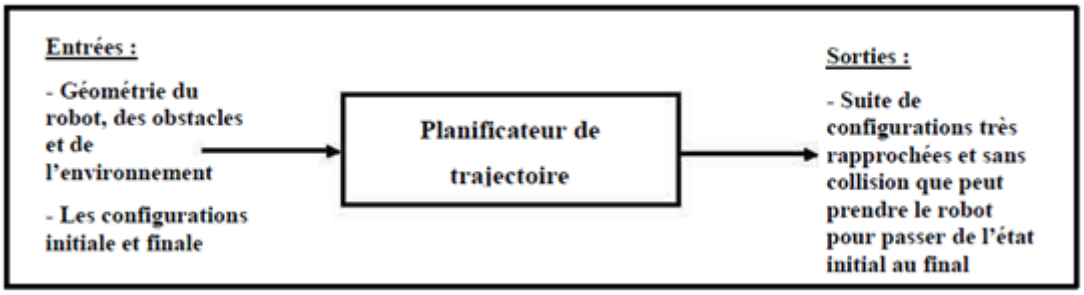


FIGURE 1.1: Représentation du fonctionnement d'un planificateur de trajectoire.

Dans cette section, nous énoncerons d'abord le problème général de planification, par la suite nous présentons les grandes approches et méthodes de résolution proposées dans la littérature et enfin nous parlerons de certains travaux de recherche qui seront plus pertinents pour notre problème.

Avant d'aborder le problème de la planification du chemin il faut définir ce qui est un chemin. Un chemin pour le robot entre deux configurations q_{init} et q_{goal} est une fonction continue τ telle que $\tau : [0,1] \rightarrow C$ avec $\tau(0) = q_{init}$ et $\tau(l) = q_{goal}$ ou C est l'espace de configurations du robot.

De plus, si toute configuration définie par le chemin τ entre l'intervalle $[0, 1]$ se trouve dans le même composant de C_{free}, il s'agit d'un chemin libre d'obstacles ou C_{free} est l'espace de configuration qui n'est pas occupé par les obstacles.

Comme nous l'avons déjà dit, le problème de la planification de trajectoire consiste à trouver cette fonction continue qui éventuellement permettra au robot de passer d'une position ou configuration initiale q_{init} à une autre finale q_{goal}. Le problème se présente lorsque dans cet espace de configuration il y a certaines configurations qui sont interdites à cause des obstacles.Alors, l'espace de configuration (C) est l'union du sous-espace C_{obs} et du sous-espace C_{free}. Puisque nous voulons que le chemin soit libre d'obstacles, nous devrions effectuer notre recherche de la fonction τ sur le sous-espace C_{free}.

Dans cet ordre d'idées, le problème de planification de chemin consiste alors à définir le sous-espace C_{free}, pour continuer par la suite avec la recherche des configurations continues qui forment le chemin.

Comme solution à ce problème, il y a un nombre important d'approches qui se sont faites remarquer à travers le temps et qui ont inspiré plusieurs autres travaux de recherche.

On peut résumer l'objectif de planification comme suit :

L'objectif principal de planification de chemin est de : développer et de valider un «algorithme efficace» de génération de «chemins optimaux» entre une position initiale et une destination dans un maillage.

La section 1.3 présentera les critères de performance à atteindre qui permettent de définir les termes «algorithme efficace» et «chemins optimaux». La figure 1.2 indique les intrants et extrants de l'algorithme à développer.[1]

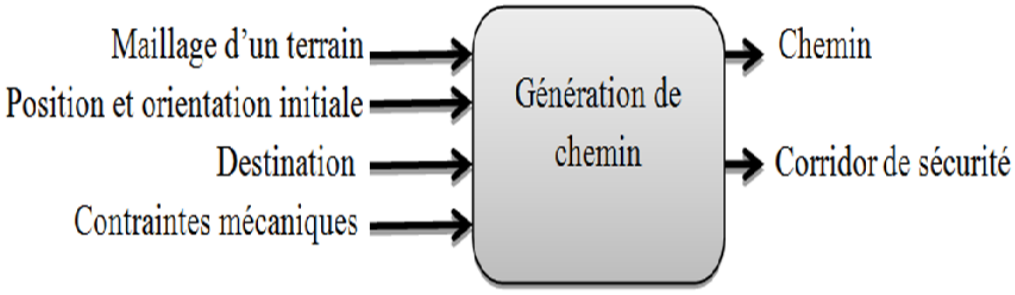


FIGURE 1.2: Schéma des entrées et sorties de l'algorithme de planification.

- **Corridor de sécurité**

Aucun robot n'est apte à suivre parfaitement une ligne vu la possibilité de glissement des roues. II est donc utile de fournir en plus d'un chemin un corridor de sécurité. Le corridor permet de quantifier dans quelle mesure le robot peut sortir de son chemin sans compromettre sa sécurité. Celui-ci permet d'associer au chemin une certaine incertitude quant à la précision du positionnement du robot requise. Sans ce corridor, le chemin possède une épaisseur nulle et aucun robot n'est capable de suivre une ligne si mince en pratique, Pour mieux comprendre le concept du corridor de sécurité, la figure 1.3 est présentée.

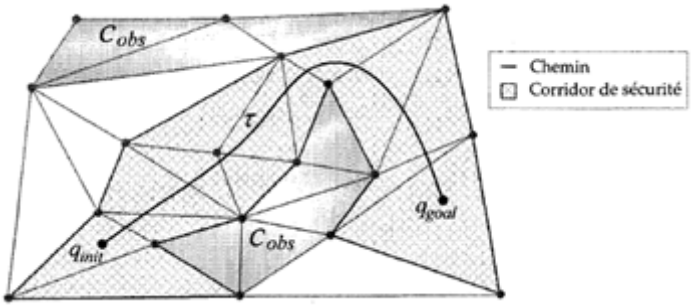


FIGURE 1.3: Schéma du maillage d'un terrain accompagne d'un chemin et d'un corridor de sécurité.

Dans la littérature, trois principales approches pour la résolution du problème de planification de trajectoires apparaissent :

- les méthodes Roadmap ;

- Méthode de champ de potentiel artificiel ;
- la décomposition en cellules.

Dans ce qui suit, nous donnons un bref aperçu du fonctionnement interne de chacune de ces méthodes. Nous nous attarderons un peu plus sur la méthode de la décomposition en cellule puisque c'est sur la base de cette dernière que nous avons appliqués les algorithmes évolutionnaires.

1.2.1.1 Les méthodes Roadmap

Ces approches représentent la connexité de l'espace des configurations libres C_{free} par un graphe. Le fonctionnement global de l'algorithme se déroule toujours sur deux étapes successives. Dans la première phase, la construction de la Roadmap est effectuée. Cette Roadmap se présente sous la forme d'un graphe non orienté $G = (N, V)$. L'ensemble des nœuds N est un ensemble de configurations du robot dans l'espace libre C_{free}. V est l'ensemble des liens entre les nœuds du graphe deux à deux. Ces liens représentent des trajectoires simples dans C_{free}, donc sans collision, entre deux configurations du graphe. Dans la deuxième phase, les configurations initiale et finale q_{init} et q_{goal} sont ajoutées au réseau avant d'essayer de trouver un chemin reliant les deux.

On distingue cependant deux variantes différentes dans les méthodes Roadmap : les méthodes déterministes et les méthodes probabilistes. Ces deux modèles diffèrent dans l'implémentation de la première phase : la construction de la Roadmap.

Dans les approches déterministes, le graphe construit est toujours le même pour un même espace des configurations. Elles sont complètes mais compliquées et lentes. Elles ne sont pas applicables à tout type d'espace mais plutôt aux espaces de faibles dimensions (2 ou 3). On pourrait en citer différents exemples comme l'approche par graphe de visibilité, l'approche par rétraction, la méthode Freeway et la méthode Silhouette.

Pour expliquer un peu le fonctionnement de ces méthodes, nous nous attardons un peu pour faire une brève description de la méthode par graphe de visibilité. Cette méthode a été introduite par Nilsson en 1969 (figure 1.4) et donne de bons résultats pour le cas de chemins courts dans un espace 2-D.

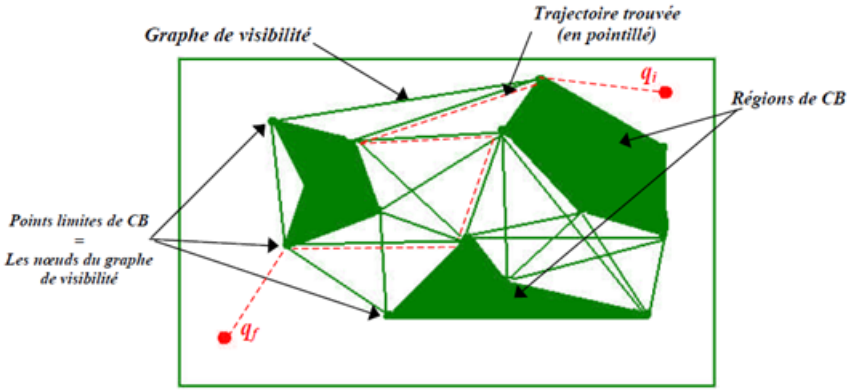


FIGURE 1.4: Approche par graphe de visibilité.

L'ensemble des configurations avec collision donc l'ensemble à omettre de l'espace de recherche est dénoté CB (figure 1.4). La Roadmap construite lors de la phase de discrétisation sur l'espace de configurations C consiste en un graphe de visibilité noté G. C'est un graphe non orienté où les nœuds correspondent aux points limites de CB. La construction de G doit cependant répondre à certains critères qui sont :

1. Deux nœuds du graphe G sont connectés par une liaison s'ils peuvent être reliés par une frontière de CB ou si la ligne qui les relie se trouve entièrement dans C_{free} ;
2. Les configurations initiales et finales peuvent être ajoutées facilement dans G.

Pour améliorer cette approche, on peut construire des graphes de visibilité réduits (seulement ceux dont on a besoin) ce qui permettrait de diminuer la complexité énormément.

– **Diagramme de Voronoi**

Contrairement à l'approche du graphique de visibilité, l'usage du diagramme de Voronoi est une méthode roadmap qui maximise la distance entre le robot et les obstacles. L'espace des configurations est décomposé en réseau de courbes forme par le diagramme de Voronoi. La figure 1.5 montre un exemple de cette méthode.

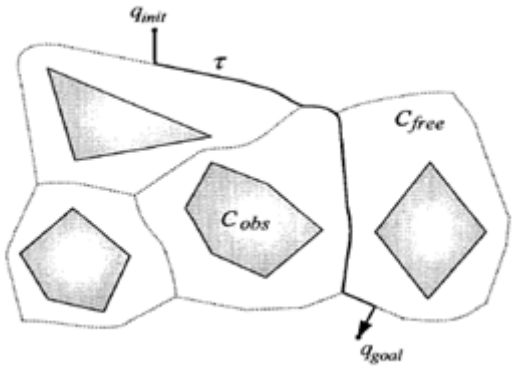


FIGURE 1.5: Exemple d'un chemin génèré au moyen de la méthode du diagramme de Voronoi.

Cette approche est de complexité $O(nlogn)$. Elle requiert donc moins de ressource informatique que la méthode du graphe de visibilité. Aussi, elle génère des chemins plus sécuritaires, mais moins optimaux que dans le cas de cette dernière méthode.[2]

1.2.1.2 Méthode de champ de potentiel artificiel

Cette méthode consiste à représenter le robot comme une particule qui se déplace dans un champ de potentiel fictif. Les variations locales de ce champ de potentiel fourniront une idée de la structure de l'espace de configuration libre, dans la plus part des cas, l'espace libre sera obtenu par la composition d'un premier champ de potentiel attractif, qui a pour objectif d'attirer le robot vers la solution et d'un champ répulsif qui a pour but d'éloigner le robot des obstacles. Le chemin est alors calculé par un algorithme de descente du gradient du potentiel obtenu.

Dans cette partie on introduit la théorie des champs de potentiels artificiels (APF) telle qu'elle a été développée par Khatib (1986).

Le concept des champs de potentiels artificiels a été introduit initialement pour le contrôle des robots manipulateurs. Le but est de déplacer l'effecteur dans l'espace du robot, tout en s'assurant que les différents segments du bras n'entrent pas en collision avec les obstacles. La méthode a naturellement été étendue aux robots mobiles.

II s'agit d'attribuer à l'espace du robot un champ scalaire appelé champ de potentiel artificiel. Ce champ est la résultante de deux composantes :

1. Un champ calculé en fonction de la position du but relativement à la position instantanée du robot, ce champ est concave et minimal au but.

2. Un champ calculé en fonction de la forme et de la position relative des obstacles par rapport à celle du robot, ce champ augmente lorsque le robot s'approche d'un obstacle.

$$\phi = \phi_b + \phi_o \tag{1.1}$$

Ou ϕ_b représente le champ de potentiel du but,ϕ_o représente celui des obstacles et ϕ représente le champ de potentiel total.

Notons que le champ de potentiel total ϕ est calculé par le contrôleur du robot et n'existe pas physiquement dans l'environnement. Ce n'est qu'un artifice de calcul pour bâtir les fondements théoriques de la méthode. II permet de calculer une référence de vitesse à donner au robot.

Une fois le champ de potentiel total calculé, un champ de forces est calculé par le gradient du potentiel selon 1'equation (1.1) :

$$\overrightarrow{F} = -grad\,(\phi) \tag{1.2}$$

Qui est équivalent à deux équations algébriques :

$$\begin{cases} F_x = & -\frac{d\phi}{dx} \\ F_y = & -\frac{d\phi}{dy} \end{cases} \tag{1.3}$$

Le fait de choisir un champ de forces artificielles qui dérive d'un potentiel revient à dire que ce champ de forces est irrotationnel. Cette supposition est logique puisque les forces virtuelles sont calculées par analogie aux forces de gravitation mécanique. De plus, le fait que le champ de forces soit irrotationnel garantit que le robot ne sera pas pris dans un « vortex », ce qui serait contraire au but de la méthode qui est de le conduire vers le but suivant un chemin relativement direct.

L'expression de la force dans 1'equation (1.3) n'est autre que la ligne de plus grande pente, qui rappelle la méthode de descente de gradient classique. En d'autres termes le robot se déplace vers le but suivant les potentiels décroissants, dans la direction ou la chute de potentiel est la plus grande.

La force calculée en chaque point permet de déduire une vitesse de référence à imposer au robot. Dans le cas classique, la vitesse de référence est confondue avec la force :

$$\begin{cases} F_x = V_x = & -\frac{d\phi}{dx} \\ F_y = V_y = & -\frac{d\phi}{dy} \end{cases} \tag{1.4}$$

Cette méthode a connu du succès grâce à sa vitesse et à son applicabilité dans la planification de mouvements en temps réel. Mais elle présente l'inconvenient de rester bloquée dans des minimaux locaux. Ces minimums sont généralement lies à la géométrie et à la répartition des obstacles dans l'espace de travail du robot et surtout aux coefficients de penalties qui sont associés à cet espace pendant la construction du champ de potentiel. Pour éviter ce problème, plusieurs travaux ont visé deux solutions. La première est de définir une fonction de potentiel avec peu ou sans minimaux locaux, et la deuxième s'adresser à l'implantation d'algorithmes de recherche que puissent éviter ces minimaux locaux.[2]

1.2.1.3 Méthode de décomposition en cellule

Elle consiste à décomposer l'espace libre du robot en régions simples appelées cellules. Dans ce graphe, il est possible d'associer un coût pour passer d'une cellule à l'autre. Ce cout peut être basé sur plusieurs critères relies a la topologie du terrain, aux capacités du robot, a la distance de la destination, etc.

Typiquement, il existe deux méthodes pour bâtir une décomposition de 1'environnement du robot :

1. La décomposition cellulaire exacte décompose l'espace libre en cellules qui une fois unies forment exactement l'espace libre du robot ;
2. La décomposition cellulaire approximative utilise des cellules de forme prédéfinie (p.ex., triangles ou rectangles) qui une fois unies forment un sous-ensemble de l'espace libre.

La figure 1.6 montre un exemple de chemin obtenu par la méthode de décomposition cellulaire exacte.

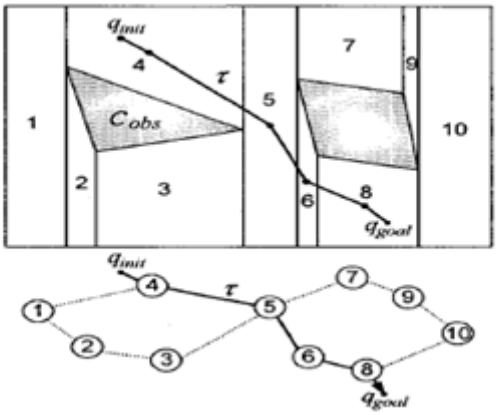


FIGURE 1.6: Exemple d'un chemin généré au moyen de la méthode de décomposition cellulaire exacte et graphe de connectivité entre les cellules.

Dans un premier temps, l'espace libre est décomposé en un assemblage de cellules (numérotés de 1 à 10 a la figure 1.6). Ensuite, le graphe de connectivité est généré et enfin, le chemin τ est obtenu au moyen d'une recherche de graphe. La méthode approximative, quant à elle, est une des techniques de planification de chemin les plus utilisées vue la popularité d'une représentation approximative en «grille» de l'espace des configurations des robots .La figure 1.7 montre un exemple d'une décomposition en grille et d'un chemin planifie.

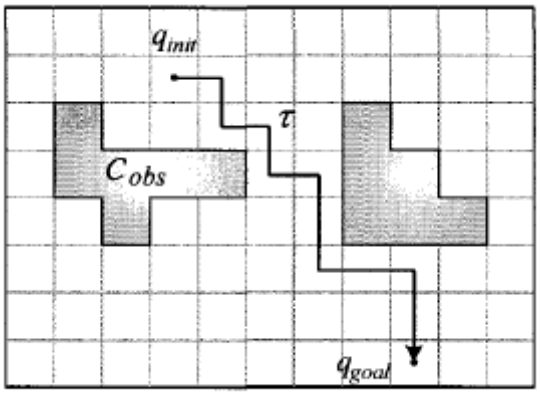


FIGURE 1.7: Exemple d'un chemin généré au moyen de la méthode de décomposition cellulaire approximative.

Recherche de graphe tel qu'expliqué précédemment, dans une représentation cellulaire de l'espace, il est possible de relier les cellules entre elles par des liens pondérés, c'est-à-

dire qu'il est possible d'associer un cout pour se déplacer d'une cellule à l'autre.

S'il y existe une cellule jugée dangereuse, un coût infini peut lui être assigne afin de s'assurer que le robot ne visite pas cette cellule. Un chemin peut être obtenu au moyen d'une recherche dans le graphe de connectivité. Les mathématiques combinatoires proposent quelques algorithmes de recherche dans des graphes.

L'algorithme de Dijkstra a jeté les bases de la recherche optimale dans des graphes. Cette méthode permet d'obtenir le chemin le moins «couteux» dans des graphes à pondération positive (dont les poids sont positifs). Cette optimalité a un prix, la complexité de calcul de la recherche peut atteindre $O(n^2)$.

L'algorithme de recherche A* est un cas particulier de l'algorithme de Dijkstra qui cherche à réduire l'ampleur de la recherche en incluant une heuristique a la fonction coût .Par exemple, il est possible d'ajouter à chaque cellule un cout qui est fonction de la distance par rapport à la destination. Cela a pour effet de pénaliser les cellules qui s'éloignent de la position de la cible. II existe aussi d'autres variantes de la méthode de Dijkstra telle que l'algorithme B* qui guide lui aussi la recherche au moyen d'une heuristique. La méthode A* est bien détaillé dans le chapitre 7.[4]

Des recherches récentes développent l'adaptation des algorithmes évolutionnaires telles que : optimisation par colonie de fourmis (ACO), essaim particulaire (PSO), colonie de bactérie, Artificial fish swarm algorithm (AFSA),... au problème de planification de trajectoire dans l'espace des configurations des robots représenté en «grille».

1.2.2 Suivi de trajectoire

Une fois la trajectoire générée, il s'agit de faire déplacer le robot de façon à ce qu'il suive cette trajectoire. C'est donc un cas de contrôle à bas niveau qui consiste en une boucle d'asservissement interne à celle de la navigation.

Vue à la non-linéarité d'un robot mobile, due à la contrainte non-holonome, il est nécessaire de concevoir des méthodes de contrôle spécifique.

Des approches basées sur les principes de stabilité et d'énergie consistent à faire converger vers zéro l'erreur entre l'état actuel et l'état désiré du robot en utilisant les principes de stabilité de Lyapunov - Xu et Yang (2001). Koh et Cho (1995) introduisent des méthodes de commande de robot mobile par mode de glissement.

Des méthodes de commande intelligente sont aussi abondantes dans la littérature. L'utilisation de la logique floue permet de commander le robot de façon à passer par les

points de passage donnes en utilisant des régies qualitatives. L'utilisation des réseaux de neurones - Yang et Meng (2001) permet de doter le robot de capacités d'apprentissage de façon à pouvoir s'adapter à une grande variété de conditions.[1]

1.3 Critères de performances

1.3.1 Critères de performances d'un «chemin optimal»

Critère	**Description et explication**
Atteinte de la cible	Si la destination appartient à l'espace libre, le robot doit atteindre cette cible.
Longueur	Le chemin doit minimiser la distance à parcourir.
Energie	Le chemin doit considérer la minimisation de l'énergie consommée par le robot lors du déplacement.
Sécurité	Le robot doit suivre un chemin dépourvu d'obstacle (roche, cratère, etc.).
Rugosité	Le chemin doit être lisse et dépourvu de changement vif de direction.
Contrainte holonome	Le chemin doit respecter les limites mécaniques du robot.

TABLE 1.1: Critères de performances d'un «chemin optimal».

Les critères juges de plus hautes importances pour un «chemin optimal» sont l'atteinte de la cible ainsi que la sécurité du robot. L'optimalité du chemin au sens mathématique vient surtout des critères de longueur de chemin et d'énergie consommée. Enfin, l'aspect rugosité permet de minimiser l'accumulation d'erreur de l'odométrie, car chaque fois que le robot change brusquement d'orientation, il glisse et accumule des erreurs de positionnement. Le tableau suivant expose les critères qui permettent de considérer un algorithme comme «efficace».

1.3.2 Critères de performances d'un «algorithme efficace»

Critère	Description et explication
Complexité	La complexité d'un algorithme influence directement le temps de calcul. Ce critère doit être minimisé.
Robustesse	Un algorithme doit être capable de gérer la presque totalité des situations sans retourner d'erreur à 1'execution.
Contrainte d'optimisation	Tout problème d'optimisation impose des contraintes qui limitent la dimension de la recherche de solution. L'algorithme doit imposer le minimum de contrainte dans sa recherche.
Complétude	Un algorithme complet retourne une solution lorsqu'elle existe et retourne un message d'erreur dans le cas contraire.
Capable d'utiliser un maillage 3D	L'algorithme doit être en mesure d'utiliser un maillage 3D, autrement un traitement est requis pour abaisser la dimension à 2.5D [1].
Insensibilité au maillage	Le conditionnement et la qualité du maillage doivent influencer minimalement la solution.

TABLE 1.2: Critères de performances d'un «algorithme efficace».

La robustesse et la complétude sont les critères les plus importants d'un algorithme. Néanmoins, un robot immobile dans un environnement statique peut prendre son temps à planifier un «chemin optimal». Dans cette situation précise, la qualité du chemin est de plus haute importance que le temps de calcul. Mieux vaut que l'algorithme requiert quelques secondes de plus que d'aventurer le robot sur un chemin incertain ou trop énergivore. Aussi, il est préférable que l'algorithme n'impose pas de contrainte non essentielle à la solution telle que de forcer le chemin à passer par le centre des cellules d'un maillage .[4]

1.4 Sommaire de méthodes de planification

Depuis les quatre dernières décennies, l'homme a imaginé toutes sortes de méthodes ayant pour but de tracer les chemins des robots mobiles. Une des plus anciennes est la méthode roadmap qui permet, au moyen d'un réseau de courbes, de réduire l'ampleur

de la recherche de chemin. La difficulté de cette méthode réside en la construction d'un réseau de courbes qui permet au robot d'accéder à l'ensemble de l'espace libre. II existe beaucoup d'approches, certaines déterministes et d'autres probabilistes, qui permettent la construction du roadmap. Une fois le roadmap obtenu, l'algorithme sélectionne un chemin considèré acceptable parmi l'ensemble du réseau de courbes.

La méthode de décomposition cellulaire a été, elle aussi, étudiée de façon intensive. Celle-ci décompose l'espace libre du robot en un nombre fini de cellules reliées par un lien de connectivité auquel il est possible d'associer un coût. Puis, un algorithme recherche dans l'assemblage des cellules, le maillage, un chemin qui rencontre certains critères d'optimalité. La faiblesse de cette méthode vient des contraintes qu'elle impose à la solution. En effet, généralement ce type de méthode force le chemin à passer par le centre ou bien les sommets des cellules. Ces contraintes limitent la quantité de solutions envisageables. II peut arriver qu'un robot, «cloué» à son maillage, doive effectuer de nombreux changements de direction pour atteindre sa destination alors que la ligne droite est libre d'obstacle.

Aussi, cette surcontrainte rend la solution sensible aux maillages placés en entrée. Deux maillages presque identiques peuvent engendrer deux chemins complètement différents. Un avantage de l'approche réside dans sa faible complexité qui, sous certaines conditions, peut atteindre $O(nlogn)$ ou même $O(n)$.

D'autres chercheurs ont vu dans la physique gouvernant le mouvement des corps, une façon à la fois simple et élégante de guider un robot vers sa cible. Pour ce faire, il est possible d'élever en potentiel la position initiale du robot et d'associer un potentiel minimal à la destination à atteindre. Les obstacles peuvent être potentiellement élevés ou retirés du domaine. Le chemin s'obtient par la recherche d'une courbe qui décroit en potentiel et qui atteint le minimum global, la cible. Cette façon d'aborder le problème apporte de nombreux avantages. D'abord, la solution est généralement indépendante de la qualité et de la régularité du maillage. Aussi, les chemins obtenus sont souvent lisses, continus et intuitifs. Cependant, l'inconvénient le plus souvent cité est la présence de minimums locaux dans la fonction potentiel ce qui peut amener le robot a une solution qui n'atteint pas la cible. II existe des méthodes réactives qui, une fois le robot coincé, permettent de sortir des minimums locaux. Une autre solution au problème est qualifiée de préventive, car c'est dans la construction de la fonction potentiel que les minimums locaux sont évités. Pour ce faire, il suffit d'utiliser une fonction harmonique et on obtient la certitude mathématique qu'une telle fonction ne présente pas de minimum local. Néanmoins, cette prévention amène une complexité de calcul élevée.

Chapitre 2

Algorithmes évolutionnaires

2.1 Introduction

L'optimisation est un sujet central en recherche opérationnelle, un grand nombre de problèmes d'aide à la décision pouvant en effet être décrits sous la forme de problèmes d'optimisation. Les problèmes d'identification, l'apprentissage supervisé de réseaux de neurones ou encore la recherche du plus court chemin sont, par exemple, des problèmes d'optimisation. Ce chapitre décrit tout d'abord le cadre de l'optimisation difficile et des algorithmes métaheuristique dans lequel nous nous plaçons dans ce travail.

2.2 Problème d'optimisation

Un problème d'optimisation au sens général est défini par un ensemble de solutions possibles s, dont la qualité peut être décrite par une fonction objectif f. (par la suite, on peut chercher à minimiser ou à maximiser $f(s)$) On cherche alors à trouver la solution $s*$ possédant la meilleure qualité $f(s*)$. Un problème d'optimisation peut présenter des contraintes d'égalité (ou d'inégalité) sur$s$, être dynamique si $f(s)$ change avec le temps ou encore multi-objectif si plusieurs fonctions objectifs doivent être optimisées.

Il existe des méthodes déterministes ("dites exactes") permettant de résoudre certains problèmes en un temps fini. Ces méthodes nécessitent généralement un certain nombre de caractéristiques de la fonction objectif, comme la stricte convexité, la continuité ou encore la dérivabilité. On peut citer comme exemple de méthode la programmation linéaire, quadratique ou dynamique, la méthode du gradient, la méthode de Newton, etc.

2.3 Optimisation difficile

Certains problèmes d'optimisation demeurent cependant hors de portée des méthodes exactes. Un certain nombre de caractéristiques peuvent en effet être problématiques, comme l'absence de convexité stricte (multi modalité), l'existence de discontinuités, une fonction non dérivable, présence de bruit, etc.

Dans tels cas, le problème d'optimisation est dit "difficile", car aucune méthode exacte n'est capable de le résoudre exactement en un temps "raisonnable", on devra alors faire appel à des heuristiques permettant une optimisation approchée.

L'optimisation difficile peut se diviser en deux types de problèmes : les problèmes discrets et les problèmes continus. Le premier cas rassemble les problèmes de type NP-complets, tels que le problème du voyageur de commerce.

Un problème "NP" est dit complet s'il est possible de le décrire à l'aide d'un algorithme polynomial sous la forme d'un sous-ensemble d'instances. Concrètement, il est "facile" de décrire une solution à un tel problème, mais le nombre de solutions nécessaires à la résolution croit de manière exponentielle avec la taille de l'instance. Jusqu' à présent, la conjecture postulant que les problèmes NP-complets ne sont pas solubles en un temps polynomial n'a été ni démontrée, ni révoquée. Aucun algorithme polynomial de résolution n'a cependant été trouvé pour de tels problèmes. L'utilisation d'algorithmes d'optimisation permettant de trouver une solution approchée en un temps raisonnable est donc courante.

Dans la seconde catégorie, les variables du problème d'optimisation sont continues. C'est le cas par exemple des problèmes d'identifications, où l'on cherche à minimiser l'erreur entre le modèle d'un système et des observations expérimentales. Ce type de problème est moins formalisé que le précédent, mais un certain nombre de difficultés sont bien connues, comme l'existence de nombreuses variables présentant des corrélations non identifiées, la présence de bruit ou plus généralement une fonction objectif accessible par simulation uniquement. En pratique, certains problèmes sont mixtes et présente à la fois des variables discrètes et des variables continues.

2.4 Algorithmes d'optimisation approchée

2.4.1 Heuristiques

Une heuristique d'optimisation est une méthode approchée se voulant simple, rapide et adaptée à un problème donné. Sa capacité à optimiser un problème avec un minimum

d'informations est contrebalancée par le fait qu'elle n'offre aucune garantie quant à l'optimalité de la meilleure solution trouvée.

Du point de vue de la recherche opérationnelle, ce défaut n'est pas toujours un problème, tout spécialement quand seule une approximation de la solution optimale est recherchée.

2.4.2 Métaheuristiques

Parmi les heuristiques, certaines sont adaptables à un grand nombre de problèmes différents sans changements majeurs dans l'algorithme, on parle alors de métaheuristiques.La plupart des heuristiques et des métaheuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire. En plus de cette base stochastique, les métaheuristiques sont généralement itératives, c'est-à-dire qu'un même schéma de recherche est appliqué plusieurs fois au cours de l'optimisation, et directes, c'est-à-dire qu'elles n'utilisent pas l'information du gradient de la fonction objectif. Elles tirent en particulier leur intérêt de leur capacité à éviter les optima locaux, soit en acceptant une dégradation de la fonction objectif au cours de leur progression, soit en utilisant une population de points comme méthode de recherche.

Souvent inspirées d'analogies avec la réalité (physique, biologie, éthologie, . .), elles sont généralement conçues au départ pour des problèmes discrets, mais peuvent faire l'objet d'adaptations pour des problèmes continus.

Les métaheuristiques, du fait de leur capacité à être utilisées sur un grand nombre de problèmes différents, se présentent facilement à des extensions. Pour illustrer cette caractéristique, citons notamment :

- l'optimisation multi objectif (dites aussi multicritères) où il faut optimiser plusieurs objectifs contradictoires. La recherche vise alors non pas à trouver un optimum global, mais un ensemble d'optima "au sens de Pareto" formant la "surface de compromis" du problème.
- l'optimisation multimodale, où l'on cherche un ensemble des meilleurs optima globaux et/ou locaux.
- l'optimisation de problèmes bruités, où il existe une incertitude sur le calcul de la fonction objectif. Incertitude dont il faut alors tenir comptes dans la recherche de l'optimum.
- l'optimisation dynamique, où la fonction objectif varie dans le temps. Il faut alors approcher au mieux l'optimum à chaque pas de temps.

– Le parallélisme, où l'on cherche à accélérer la vitesse de l'optimisation en répartissant la charge de calcul sur des unités fonctionnant ensemble. Le problème revient alors à adapter les métaheuristiques pour qu'elles soient distribuées.
– l'hybridation, qui vise à tirer parti des avantages respectifs de métaheuristiques différentes en les combinats.

Enfin, la grande vitalité de ce domaine de recherche ne doit pas faire oublier qu'un des intérêts majeurs des métaheuristiques est leur facilité d'utilisation dans des problèmes concrets. L'utilisateur est généralement demandeur de méthodes efficaces permettant d'atteindre un optimum avec une précision acceptable dans un temps raisonnable. Un des enjeux de la conception des métaheuristiques est donc de faciliter le choix d'une méthode et de simplifier son réglage pour l'adapter à un problème donné.[9]

2.5 Algorithmes évolutionnaires

Les algorithmes évolutionnaires (AE) sont une classe d'algorithmes d'optimisation par recherche probabiliste basés sur des modèles inspirés des comportements des êtres vivants.

Les AE sont basés sur des principes simples. En effet, peu de connaissances sur la manière de résoudre ces problèmes sont nécessaires, même si certaines peuvent être exploitées afin de rendre plus efficace l'évolution (en effet, il n'est pas réaliste d'espérer obtenir une méthode d'optimisation raisonnablement efficace si aucune connaissance sur le domaine à traiter). C'est pourquoi, dans de nombreux domaines, les chercheurs ont été amenés à s'y intéresser.

Un certain nombre de travaux ont porté sur l'optimisation de fonctions mathématiques complexes, Les AE ont été appliqués à différents problèmes issus de la recherche opérationnelle : la coloration de graphes, le problème du voyageur de commerce, la gestion d'emploi du temps, etc.

On trouve aussi des applications en robotique pour la recherche de trajectoire optimale ou l'évitement d'obstacles, en vision pour la détection ou la reconnaissance d'images, en recalage d'images médicales, en reconnaissance de formes, mais aussi en mécanique pour l'optimisation de formes, en chimie ,économie et gestion de production, gestion de trafic aérien ,etc.

2.5.1 Inspiration biologique

À la fin des années 80, une nouvelle voie d'exploration est apparue en intelligence artificielle. Il s'agit de l'étude et de l'utilisation des phénomènes observés dans la nature (voir figure 2.1). En effet, ce sont des systèmes composés d'agents très simples, qui peuvent produire des constructions complexes et des solutions à des problèmes non triviaux (tri, parcours optimaux, répartition de tâches, . . .).

Les informaticiens ont repris les principes d'auto-organisation et d'"émergence" présents dans ces sociétés, pour définir ce que l'on nomme l'intelligence collective.

Nous pouvons dire que l'intelligence collective caractérise un système où le travail collectif des entités (non complexes) interagissant entre elles fait "émerger" un comportement complexe global.

La méthode de recherche par colonie de fourmis basée sur le dépôt et l'évaporation de pistes de phéromone, et la méthode d'essaims particulaires ne sont que deux méthodes parmi d'autres qui s'inspirent de la biologie. Nous trouvons ainsi, dans la nature, plusieurs exemples qui donnent des idées pour la conception algorithmique. À titre d'exemple, nous pouvons citer les vols groupés d'oiseaux. Ces comportements se trouvent aussi chez des organismes plus simples unicellulaires, comme les bactéries. De nouvelles recherches ont établi que les bactéries utilisent des molécules pour communiquer entre elles. Elles utilisent un réseau composé de liens de cellule à cellule. Elles sont capables de détecter des changements dans leur environnement, s'associer avec des bactéries d'une même espèce, faire des alliances bénéfiques avec d'autres espèces, et prendre l'avantage sur des espèces concurrentes. Ce type de stratégie collective est également attribué aux fourmis, aux abeilles .....

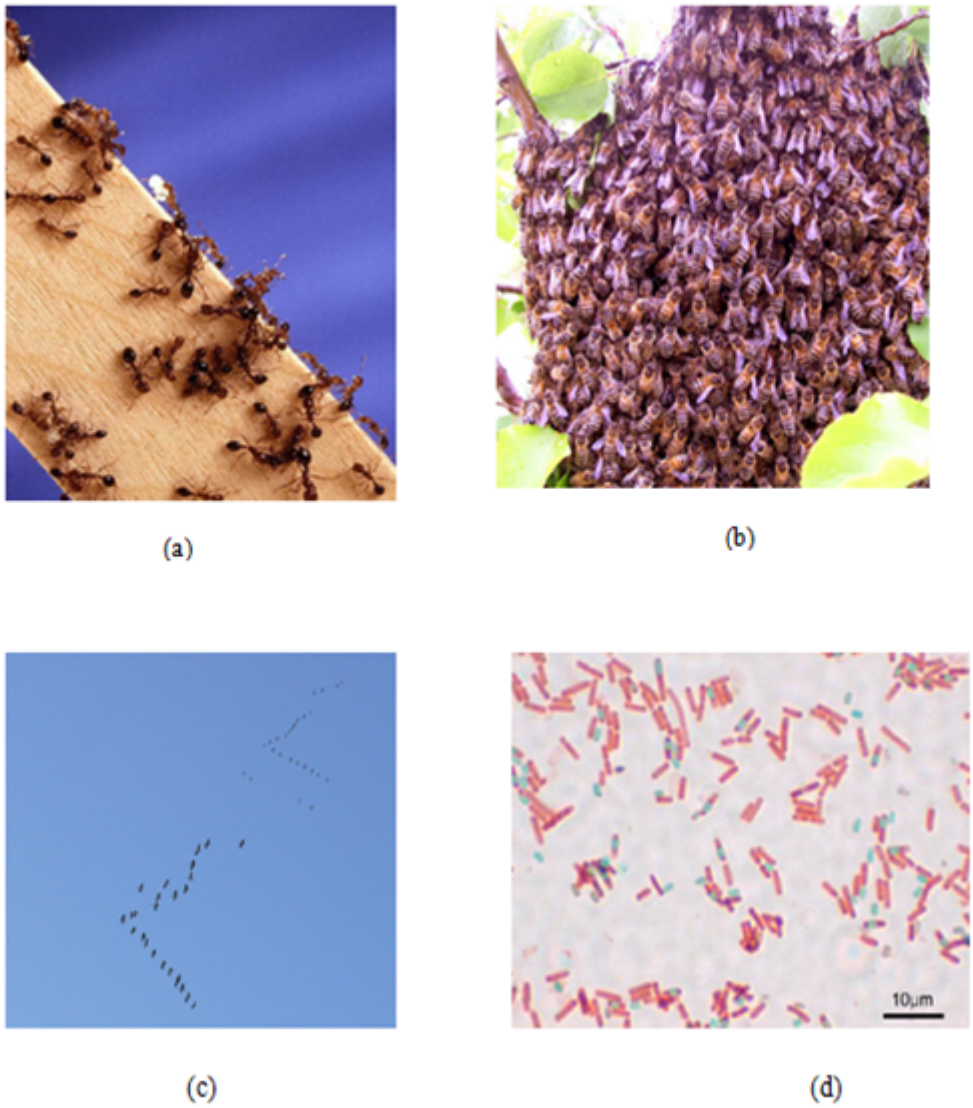


FIGURE 2.1: Auto-organisation dans les systèmes biologiques : (a) Une colonie de fourmis qui ramène de la nourriture vers le nid (b) un essaim d'abeilles (c) un vol groupé d'oiseaux (d) une formation de bactéries.

Les avantages liés à l'utilisation d'une telle approche sont :

1. la production d'une performance collective supérieure à celle des individus ;
2. une plus grande adaptation et flexibilité aux environnements réels (en général dynamiques) ;
3. la fiabilité du système dans son ensemble (la perte d'un agent ne met pas en cause

le processus général) ;
4. le faible coût des “unités”.

Par contre, un certain nombre de problèmes surgissent :

1. difficulté pour anticiper la résolution d'un problème par une intelligence “émergente” ;
2. problèmes de formulation, de compréhension et de preuve de la résolution ;
3. nécessité d'un grand nombre d'agents (notion de masse critique),
4. risques de comportements oscillatoires ou bloquants ;
5. pas de coopérations locales intentionnelles, c'est-à-dire de comportements volontairement coopératifs dédiés ou exigés par les individus.

2.5.2 Auto-organisation

C'est un mécanisme connu par les biologistes, qui stabilise les systèmes biologiques et évite les fluctuations indésirables. Un exemple célèbre de l'auto-organisation est l'évaporation de traces de phéromone chez certains insectes sociaux, cette évaporation agit d'une façon globale et continue dans le temps. Sans ce processus, l'utilisation de phéromone par la colonie serait sans utilité, puisque tout le système se mettrait à renforcer le même chemin de plus en plus, ce qui n'est pas adapté pour des environnements variables. Un autre exemple est observé dans les traces de pas dans les sentiers, où les conditions physiques ou climatiques comme l'érosion jouent le rôle d'asservissement négatif.

En résultat, nous obtenons une mémoire dynamique et robuste (une sorte de plan global) partagée par tous les individus, permettant de trouver la bonne solution ou le bon chemin, par exemple le chemin entre deux villages. Ce qui est frappant dans cet exemple, c'est que l'ensemble de la population arrive au bout du compte à trouver un chemin commun sans communication directe. Le signal de renforcement positif se fait d'une façon discontinue dans le temps, et, à chaque nouveau changement, la population est redistribuée localement pour reconnecter le chemin global. Le plan du système cognitif en entier est variable au fil du temps, s'adapte intelligemment, et s'auto-organise en réponse à des environnements non connus ou variables, sans plan ou stratégie globale. C'est comme si l'environnement coopérait directement à l'aide d'un plan global et fictif, pour former un plan complètement nouveau.

Le chemin ne sera jamais réorganisé pour explorer de nouvelles régions dans l'espace de recherche, sans l'apparition d'obstacles ou sans que des changements brusques arrivent dans l'environnement, et sans que des individus qui perdent leur chemin arrivent

à trouver d'autres solutions. Mais il faut dire aussi que le chemin n'aura jamais été créé, sans la présence locale de certains individus pour renforcer les traces. Donc, sans le processus de rétroaction négative, le système se stabilise sur une configuration donnée, et un chemin marqué ne changera jamais, les traces seront renforcées de plus en plus, sans se déplacer au fil du temps. Mais le système a besoin de composantes (des individus) pour assurer le renforcement des solutions précédentes, c'est l'exploitation de l'espace de recherche ; et de composantes qui évoluent, volontairement ou pas, d'une façon aléatoire, ces dernières serviront au processus d'exploration de l'espace de recherche. En général, les composantes d'un système social peuvent assurer les deux processus (exploration / exploitation).

Certaines espèces d'oiseaux vivent en colonies. Le fait de vivre en colonie a des bénéfices, comme une meilleure détection des prédateurs, ou encore la facilité de recherche des sources de nourriture. Dans ce cas, le mécanisme est l'imitation : des oiseaux d'une même espèce, qui cherchent à nicher, sont attirés par des sites où d'autres oiseaux de cette espèce nichent déjà, le comportement est le suivant "nicher à côté du lieu où l'autre a niché". Chez les insectes sociaux, la rétroaction positive est illustrée par le renforcement des traces de phéromone, ce qui permet à la colonie toute entière d'exploiter des solutions anciennes ou récentes. En général, comme dans les exemples précédents, la rétroaction positive est imposée implicitement au système, et localement à travers les unités qui la constituent. Les bancs de poissons suivent le comportement "je te suis où tu vas". Chez les humains, les bâillements et les rires contagieux sont des exemples de rétroaction positive, concernant la forme, "je fais ce que tu fais". Le fait de voir quelqu'un bailler ou même vouloir bailler peut déclencher chez nous un bâillement.

Cependant, il y a un risque si la rétroaction positive agit toute seule, sans la présence d'une rétroaction négative. Cette dernière a un rôle important, puisqu'elle maintient sous contrôle l'"effet boule de neige", ce qui permet d'avoir des paramètres adaptés. Le fait que la rétroaction positive par nature s'amplifie peut engendrer un potentiel destructif ou explosif dans le système où elle opère. Le comportement est donc plus compliqué qu'il n'y paraît. Par exemple, pour un banc de poissons, le vrai comportement est le suivant "je vais où les autres vont, sauf s'il n'y a plus de place". Dans ce dernier cas, les rétroactions positive et négative peuvent être implémentées dans des règles comportementales.

Il est intéressant de savoir comment les individus ont accès à l'information, ou la communiquent, puisque l'organisation émerge de plusieurs interactions. Il y a deux formes importantes de communication : soit les informations sont recueillies des voisins directement (communication directe), soit à partir du travail en cours d'exécution, c'est la stigmergie.[6]

Chapitre 3

Optimisation par colonie de fourmis (ACO)

3.1 Historique

Dans les sections qui viendront plus tard, nous présenterons la métaheuristique ACO "Ant Colony Optimization". Toutes ces idées abstraites sont inspirées des travaux de Deneubourg sur les fourmis. Cette méta-heuristique est relativement récente. Elle a été introduite en 1991 par Colorni, Dorigo et Maniezzo pour résoudre le problème du Voyageur de commerce. Elle s'est popularisée, puis a été l'objet d'améliorations dès 1995 et a été appliquée avec succès à d'autres problèmes d'optimisation combinatoire dès 1994. Nous allons tout d'abord exposer les différences et les points communs entre les fourmis virtuelles et les fourmis réelles, avant d'exposer en termes plus abstraits la métaheuristique proprement dite. Ceci expliquera comment les fourmis virtuelles peuvent être exploitées pour résoudre un problème d'optimisation combinatoire.

3.2 Similarités et différences avec les fourmis réelles

Les fourmis virtuelles ont une double nature. D'une part, elles modélisent les comportements abstraits de fourmis réelles, et d'autre part, elles peuvent être enrichies par des capacités que ne possèdent pas les fourmis réelles, afin de les rendre plus efficaces que ces dernières. Nous allons maintenant synthétiser ces ressemblances et différences.

3.2.1 Points communs

– **Colonie d'individus coopérants** : Comme pour les fourmis réelles, une colonie virtuelle est un ensemble d'entités non-synchronisés, qui se rassemblent ensemble

pour trouver une "bonne" solution au problème considéré. Chaque groupe d'individus doit pouvoir trouver une solution même si elle est mauvaise.
- **Pistes de phéromones** : Ces entités communiquent par le mécanisme des pistes de phéromone. Cette forme de communication joue un grand rôle dans le comportement des fourmis : son rôle principal est de changer la manière dont l'environnement est perçu par les fourmis, en fonction de l'historique laissé par ces phéromones.
- **Évaporation des phéromones** : La méta-heuristique ACO comprend aussi la possibilité d'évaporation des phéromones. Ce mécanisme permet d'oublier lentement ce qui s'est passé avant. C'est ainsi qu'elle peut diriger sa recherche vers de nouvelles directions, sans être trop contrainte par ses anciennes décisions.
- **Recherche du plus petit chemin** : Les fourmis réelles et virtuelles partagent un but commun : recherche du plus court chemin reliant un point de départ (le nid) à des sites de destination (la nourriture).
- **Déplacement locaux** : Les vraies fourmis ne sautent pas des cases, tout comme les fourmis virtuelles. Elles se contentent de se déplacer entre sites adjacents du terrain.
- **Choix aléatoire lors des transitions** : Lorsqu'elles sont sur un site, les fourmis réelles et virtuelles doivent décider sur quel site adjacent se déplacer. Cette prise de décision se fait au hasard et dépend de l'information locale déposée sur le site courant. Elle doit tenir compte des pistes de phéromones, mais aussi du contexte de départ (ce qui revient à prendre en considération les données du problème d'optimisation combinatoire pour une fourmi virtuelle).

3.2.2 Différences

Les fourmis virtuelles possèdent certaines caractéristiques que ne possèdent pas les fourmis réelles :

- **Elles vivent dans un monde non-continu** : Leurs déplacements consistent en des transitions d'état.
- **Mémoire (état interne) de la fourmi** : Les fourmis réelles ont une mémoire très limitée. Tandis que nos fourmis virtuelles mémorisent l'historique de leurs actions. Elles peuvent aussi retenir des données supplémentaires sur leurs performances.
- **Nature des phéromones déposées** : Les fourmis réelles déposent une information physique sur la piste qu'elles parcourent, là où les fourmis virtuelles modifient des informations dans les variables d'états associées au problème. Ainsi, l'évaporation des phéromones est une simple décrémentation de la valeur des variables d'états à chaque itération.
- **Qualité de la solution** : Les fourmis virtuelles déposent une quantité de phéromone proportionnelle à la qualité de la solution qu'elles ont découvert.
- **Retard dans le dépôt de phéromone** : Les fourmis virtuelles peuvent mettre

à jour les pistes de phéromones de façon non immédiate : souvent elles attendent d'avoir terminé la construction de leur solution. Ce choix dépend du problème considéré bien évidemment.

– **Capacités supplémentaires** : Les fourmis virtuelles peuvent être pourvues de capacités artificielles afin d'améliorer les performances du système. Ces possibilités sont liées au problème et peuvent être :

1. l'anticipation : la fourmi étudie les états suivants pour faire son choix et non seulement l'état local.
2. le retour en arrière : une fourmi peut revenir à un état déjà parcouru car la décision qu'elle avait prise à cet état a été mauvaise.

3.3 Expériences

3.3.1 Pont binaire de Deneubourg

L'expérience montre un nid d'une colonie de fourmis, qui est séparé d'une source de nourriture par un pont à deux voies de même longueur. On laisse évoluer les fourmis sur le pont, on trace ainsi en fonction du temps, le graphe du nombre de fourmis empruntant chaque branche. Le résultat de l'expérience est exposé à la figure 3.1.

L'illustration (a) représente la configuration physique de l'expérience. Le graphique (b) indique l'évolution de ce système en fonction du temps : on constate que les fourmis ont tendance à emprunter le même chemin (par exemple celui du haut) après une dizaine de minutes.

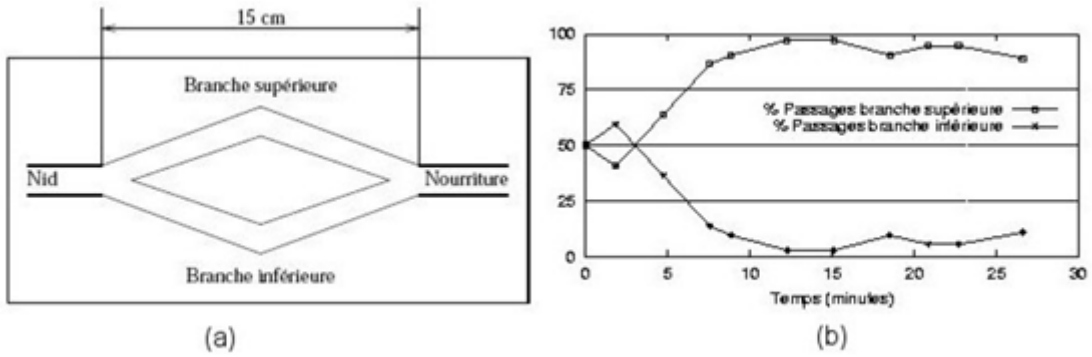


FIGURE 3.1: Pont binaire de Deneubourg.

Explication

Au départ, il n'y a pas de phéromone sur le pont. Donc, chaque branche peut être choisie par une fourmi avec la même probabilité. Néanmoins, dans notre exemple, après une certaine période, des variations aléatoires ont fait qu'un peu plus de fourmis ont choisi le chemin du haut plutôt que celui du bas.

Puisque les fourmis déposent des phéromones en avançant et puisqu'elles sont plus nombreuses en haut qu'en bas, le chemin du haut comportera plus de phéromones. Cette quantité supérieure de phéromone incite plus de fourmis à choisir la branche du haut, donc la quantité de phéromone déposée augmentera encore plus.

On en déduit que plus les fourmis suivent un chemin, plus ce chemin devient intéressant à suivre. Ainsi la probabilité avec laquelle une fourmi choisit un chemin, augmente avec le nombre de fourmis qui ont pris ce chemin précédemment.

3.3.2 Expérience du double pont binaire

On peut se demander à présent quel serait l'effet de l'augmentation de la longueur d'une des deux branches du pont. L'effet produit sera que la branche la plus courte sera sélectionnée.

En effet, les premières fourmis qui reviennent au nid avec de la nourriture sont celles qui ont emprunté le chemin le plus court dans les deux sens. Ce chemin, marqué deux fois par les phéromones, attire plus les autres fourmis que le long chemin, qui lui est marqué une seule fois dans le sens de l'aller. Cet effet se renforce au fur du temps, jusqu'à ce que toutes les fourmis choisissent le chemin le plus court. C'est ainsi que dans cette expérience, on voit que les variations aléatoires sont réduites, puisque les deux chemins n'ont plus la même longueur. Contrairement à la première expérience, le comportement des fourmis qui consistait à suivre les pistes de phéromones n'est plus le seul mécanisme présent : maintenant on associe ce mécanisme à une notion de distance.

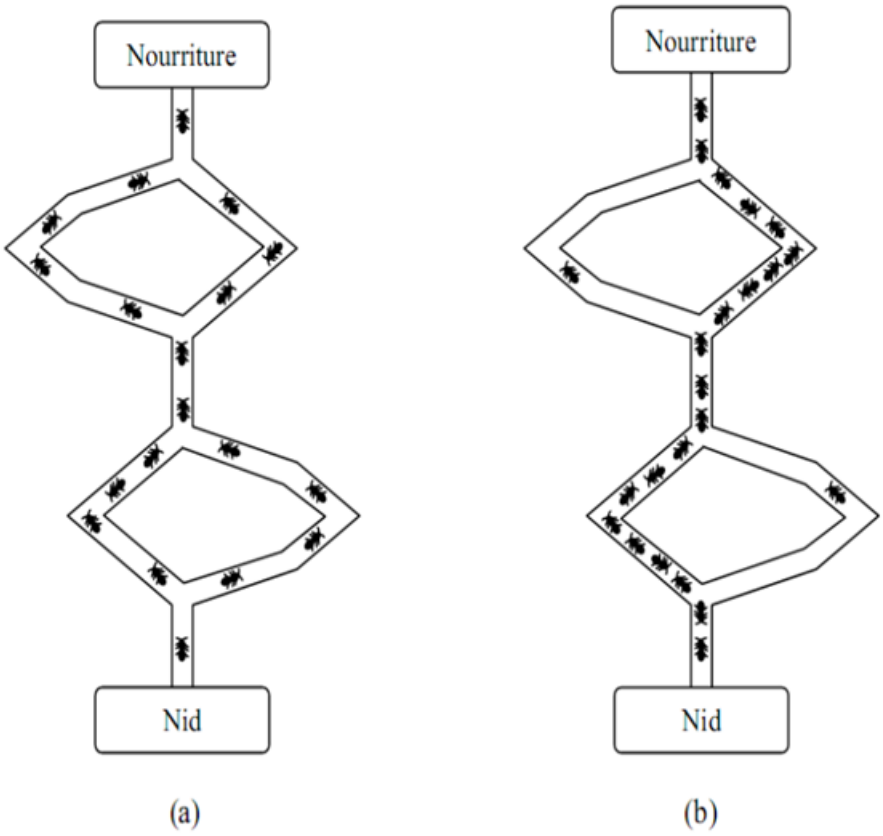


FIGURE 3.2: Expérience du double pont binaire.

Toutefois, quand le chemin le plus court n'est ouvert qu'après le chemin plus long (par exemple, quand le chemin était au départ bloqué par un obstacle), les fourmis continuent de parcourir le chemin le plus long car c'est le seul à être recouvert de phéromone. C'est ainsi que, l'évaporation des phéromones joue un rôle capital : les pistes de phéromones sont moins présentes sur les chemins les plus longs, car le réapprovisionnement en phéromone demande plus de temps que sur les chemins plus courts. Donc il suffit alors qu'une poignée de fourmis choisissent le chemin auparavant bloqué pour favoriser celui-ci. Ainsi, même quand des voies plus courtes ont été découvertes après, les fourmis finissent par les emprunter.

On peut généraliser cela à plus de deux chemins possibles : dans la figure 3.2 (a), on a utilisé un double pont avec quatre chemins possibles de différentes longueurs. On voit dans le dessin (b) que la plupart des fourmis finissent par choisir le chemin le plus court. Les expériences montrent que quand environ cent fourmis ont déjà emprunté le pont, plus de 90 pourcents d'entre elles sélectionnent le chemin le plus court : les fourmis convergent donc assez rapidement.

3.3.3 Effet de la coupure d'une piste de phéromone

Cette fois, les fourmis sont en train de suivre une piste de phéromones, comme présenté à la figure 3.3 (a). À un moment donné, on a un obstacle qui barre la route des fourmis. Les fourmis qui arrivent à côté de l'obstacle doivent choisir soit d'aller à gauche soit d'aller à droite (b).Puisqu'aucune phéromone n'est déposée le long de l'obstacle, il y a autant de fourmis qui partent à gauche qu'à droite.

Néanmoins, puisque le chemin de droite est plus court que celui de gauche, les fourmis qui l'empruntent, vont retrouver plus vite la piste de phéromone de départ. Pour chaque fourmi allant du nid à la nourriture, on associe également une fourmi qui va de la nourriture au nid (en fait elles ont été séparées par l'apparition brutale de l'obstacle). Les phéromones de ces fourmis vont se superposer à droite. Donc quand elles vont rejoindre le chemin initial, le chemin de droite sera deux fois plus imprégnée de phéromone que la piste de gauche, où les deux fourmis n'ont pas encore pu rejoindre la piste initiale (ce chemin étant plus long).

Les fourmis qui arrivent à l'obstacle à partir de ce moment, préféreront suivre la piste de droite. Le nombre de fourmis qui passent par la droite va augmenter, ce qui augmentera encore la concentration de phéromones. De plus, l'évaporation des phéromones sera plus forte sur la piste de gauche du fait que sa longueur est supérieure. La piste de gauche sera donc rapidement abandonnée, parce qu'elle en est beaucoup moins imprégnée : les fourmis passeront toutes très rapidement par la piste la plus courte.

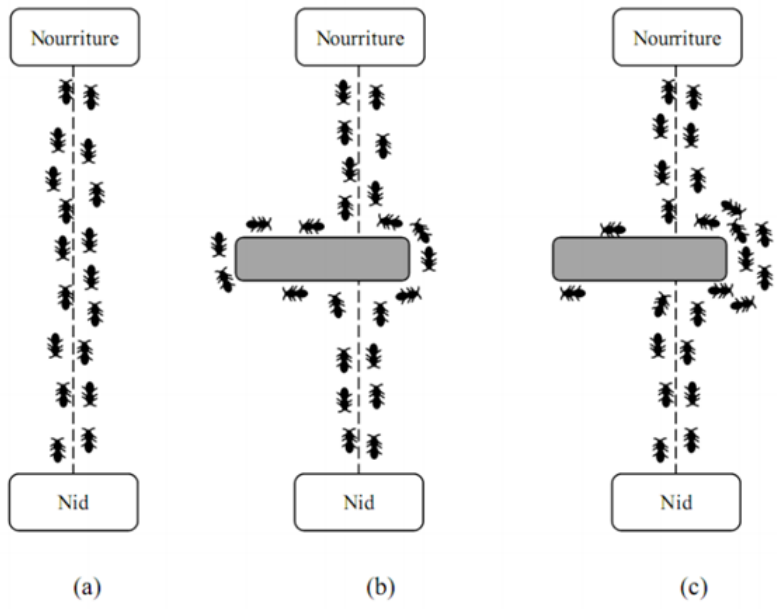


FIGURE 3.3: Effet de la coupure d'une piste de phéromone.

Il est intéressant de remarquer que bien qu'une seule fourmi soit capable de construire une solution (i.e. de trouver un chemin du nid à la nourriture), c'est seulement le comportement de l'ensemble de la colonie qui crée le chemin le plus court.[22]

3.4 Optimisation par colonie de fourmis et problème de voyageur de commerce

Le problème de voyageur de commerce (PVC) a fait l'objet de la première implémentation d'un algorithme de colonie de fourmis : le 'Ant System'

3.4.1 Ant System

Dans l'algorithme Ant System (AS), chaque fourmi est initialement placée sur une ville choisie aléatoirement, chacune possède une mémoire qui stocke la solution partielle qu'elle a construite auparavant. Initialement, la mémoire contient la ville de départ. Commençant à partir de cette ville, une fourmi se déplace itérativement d'une ville à une autre. Quand elle est à une ville i, une fourmi k choisit d'aller à une ville non encore visitée j avec une probabilité donnée par :

$$P_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{l \epsilon N_j^k} \left([\tau_{il}(t)]^{\alpha} \times [\eta_{il}]^{\beta} \right)} & si \quad j \epsilon N_i^k \\ 0 & sinon \end{cases} \tag{3.1}$$

- $\tau_{ij}(t)$ est l'intensité de la trace de phéromone dans l'arête (i, j) à l'instant t ;
- $\eta_{ij} = 1/d_{ij}$ est une information heuristique à priori valable, où d_{ij} est la distance entre la ville i et la ville j , l'idée étant d'attirer les fourmis vers les villes les plus proches ;
- α, β sont deux paramètres qui déterminent l'influence relative de la trace de phéromone et de l'information heuristique ;
- N_i^kest défini comme étant l'ensemble des villes que la fourmi k, placée sur la ville i, n'a pas encore visité à l'instant t dans le cycle courant.

La construction de solution se termine après que chaque fourmi ait complété un tour. Ensuite, les traces de phéromone sont mises à jour. Dans AS, la mise à jour se fait, d'abord, en réduisant les traces de phéromone avec un facteur constant ρ (c'est l'évaporation de phéromone) et, ensuite, en permettant à chaque fourmi de déposer de la phéromone sur les arêtes qui appartiennent à son tour. Ainsi la formule de mise à jour de phéromone est comme suit :

$$\tau_{ij}(t+1) = (1-\rho) \times \tau_{ij}(t) + \sum_{k=1}^{M} \Delta \tau_{ij}^{k} \tag{3.2}$$

Avec $0 \leq \rho \leq 1$ et M est le nombre de fourmis.

Le paramètre ρ est ainsi utilisé pour éviter l'accumulation illimitée de phéromone et permet à l'algorithme d'oublier les mauvaises décisions précédemment prises. Sur les arêtes qui n'ont pas été choisis par les fourmis, la force associée va décroître rapidement avec le nombre d'itérations.

$\Delta \tau_{ij}^{k}$est la quantité de phéromone que la fourmi k dépose sur l'arête (i, j), il est définie par :

$$\Delta \tau_{ij}^{k} = \begin{cases} \frac{Q}{L^k} & si \quad (i,j) \epsilon tabou^k \\ 0 & sinon \end{cases} \tag{3.3}$$

Ou L^k est la longueur du tour généré par la fourmi k, Q une constante de l'algorithme et $(Tabou)^k$est la liste des villes déjà visitées.

Avec cette formule, les arêtes du tour le plus court recevront la plus grande quantité de phéromone. En général, les arêtes qui sont utilisées par plusieurs fourmis et qui appartiennent aux tours les plus courts recevront plus de phéromone et en conséquence

seront plus favorisés dans les itérations futures de l'algorithme.

3.4.2 Extensions de "Ant System"

Malgré ses résultats encourageants, AS n'était pas compétitif avec les algorithmes de l'état de l'art du PVC. Des recherches sont alors entreprises pour l'étendre et essayer d'améliorer ses performances en contrôlant mieux l'intensification et la diversification de la recherche.

3.4.2.1 Les fourmis élitistes

Nous présentons ici une amélioration qui donne de bons résultats en pratique. Elle consiste à renforcer de manière artificielle à chaque étape les quantités de phéromone présentes sur les arcs appartenant au meilleur tour trouvé jusqu'à présent.

Intuitivement, cette méthode consiste à faire reparcourir le meilleur tour par certaines fourmis artificielles, dites fourmis élitistes. Ces fourmis sont conceptuellement identiques aux autres fourmis, si ce n'est qu'elles choisissent leur chemin de manière déterministe et qu'elles ne se mettent en route que quand toutes les autres fourmis ont terminé un tour.

Si l'on désigne par T^* le meilleur tour et par L^* sa longueur, à chaque fin de cycle on réalise l'opération suivante :

$$\tau_{ij} \leftarrow \tau_{ij} + \begin{cases} e\frac{Q}{L^k} & si \quad (i,j)\epsilon T^* \\ 0 & sinon \end{cases} \tag{3.4}$$

Où e est le nombre de fourmis élitistes.

L'effet des fourmis élitistes est de grandement accroître la convergence de la méthode (typiquement d'un facteur 10), au risque que celle-ci converge vers une solution non-optimale. Cette amélioration de la convergence s'explique très bien intuitivement, puisque cela revient à favoriser un chemin parmi ceux déjà explorés. Elle est précieuse dans l'AS qui reste un algorithme très gourmand en ressources de calcul.

Il y a donc une valeur optimale à e. Quand nous sommes sous cette valeur et que nous augmentons e, le meilleur tour peut être découvert plus tôt. Quand nous sommes au-delà de cette valeur, les fourmis élitistes forcent l'exploration autour de circuits non-optimaux dès les premières étapes de la recherche, ce qui amoindrit les performances.

Notons aussi que la présence de fourmis élitistes peut même permettre l'émergence de meilleurs tours. En effet, les fourmis élitistes sont un mécanisme qui permet de focaliser

la recherche sur des zones prometteuses de l'espace de recherche en augmentant l'attrait des fourmis pour des pistes que l'on sait faire partie d'un bon tour. Or, quand un bon tour a été découvert, il y a de forte chance que le tour optimal n'en est pas très éloigné.

3.4.2.2 Ant-Q

Dans cette variante de "AS", la règle de mise à jour locale est inspirée du "Q learning" inventée par Gambardella et Dorigo en 1995. Cependant, aucune amélioration par rapport à l'algorithme AS n'a pu être démontrée. Cet algorithme n'est d'ailleurs, de l'aveu même des auteurs, qu'une préversion du "Ant Colony System".

3.4.2.3 Ant Colony System

L'algorithme "Ant Colony System" (ACS) a été introduit par Dorigo et Gambardella pour améliorer les performances du premier algorithme sur des problèmes de grandes tailles. ACS est fondé sur des modifications du "Ant system" :

1. Sur des grands parcours on peut reprocher à AS de ne pas assez se focaliser sur les meilleures solutions trouvées précédemment pour influer sur les décisions des fourmis. Pour remédier à cela ACS fait rentrer en jeu une variable constante paramétrable q_0 de valeur distribuée normalement entre 0 et 1. À chaque fois que l'on doit décider de la prochaine ville où placer la fourmi, on attribue une valeur aléatoire uniformément distribué entre 0 et 1 à la variable q. Lorsque $q \leq q_0$ on effectue un choix uniquement basé sur les villes déjà visitées (de manière efficace). Dans le cas où $q \geq q_0$ on effectue un choix de la même manière que pour AS. En ajustant q_0 on peut donc pousser les fourmis à se focaliser sur les meilleures solutions trouvées ou bien à les pousser à chercher d'autres chemins.
2. ACS met en place une politique de mise à jour des phéromones visant à intensifier la solution optimale. Lors de la mise à jour locale (la mise à jour pas à pas effectué sur un arc lorsqu' une fourmi l'emprunte), ACS a tendance de diminuer la teneur en phéromones. A l'inverse de AS, ACS augmente lors de la mise à jour globale (après chaque cycle) uniquement le chemin appartenant au meilleur tour (celui de longueur minimale).

3.4.2.4 MAX - MIN Ant System (MMAS)

Différences avec " AS"

1. Uniquement le parcours le plus court est mis-à-jour en phéromones.

2. Les valeurs des phéromones sur chaque arc sont bornées par τ_{min} et τ_{max}.
3. Les valeurs des phéromones sur chaque arc sont initialisées à la valeur maximum τ_{max}.
4. La quantité de phéromones que l'on fait évaporer est proportionnelle à sa valeur au moment de la modification, plus les pistes sont fortes plus ses phéromones seront diminués.

Intérêt

- On empêche ainsi la monopolisation de certains arcs qui ont été tellement imprégnés au début du processus de recherche qu'ils sont systématiquement parcourus par les fourmis.
- On permet grâce à cette façon de gérer l'évaporation de tirer vers le bas, les arcs chargés fortement en phéromones afin de vérifier si leur importance est pertinente. De ce fait si ce n'est pas le cas, les pistes plus faiblement chargés pourront leur prendre le pas.[7]

3.5 La métaheuristique d'optimisation par colonie de fourmis

En utilisant la métaheuristique ACO, l'une des tâches principales est d'encoder le problème à résoudre sous la forme d'un problème de recherche d'un chemin optimal dans un graphe, appelé graphe de construction, où la faisabilité est définie en respectant un ensemble de contraintes. Les fourmis artificielles peuvent être caractérisées comme des procédures de construction gloutonnes stochastiques qui construisent des solutions en se déplaçant dans le graphe de construction.

Dans cette section, nous définissons d'abord une représentation générique du problème que les fourmis exploitent pour construire des solutions. Ensuite, nous détaillons le comportement des fourmis durant la construction de solution et finalement nous définissons la métaheuristique d'optimisation par colonie de fourmis.

3.5.1 Représentation du problème

Considérons le problème de minimisation (S, f, Ω) où S est l'ensemble des solutions candidates, f la fonction objectif qui associe à chaque solution candidate $s \epsilon S$ une valeur de la fonction objectif $f(s)$ et Ω est l'ensemble des contraintes. L'objectif est de trouver une solution optimale globales$_{opt} \epsilon S$ qui est une solution minimisant f et qui satisfait les contraintes Ω. Les différents états du problème sont caractérisés comme une séquence

de composants.

Les fourmis artificielles construisent les solutions en se déplaçant sur le graphe construit $G(C, L)$ où les composants C sont les sommets et l'ensemble L sont les connexions des composants de C (un élément de$L$ est une connexion). Les contraintes du problème sont implémentées directement dans les règles de déplacement des fourmis (soit en empêchant les mouvements qui violent les contraintes, soit en pénalisant de telles solutions).

3.5.2 Comportement des fourmis

Les fourmis peuvent être caractérisées comme des procédures de construction stochastiques qui construisent des solutions en se déplaçant sur le graphe construit $G(C, L)$. Les fourmis ne se déplacent pas arbitrairement, mais elles suivent une politique de construction qui est une fonction des contraintes du problème.

Les traces de phéromone τ peuvent être associées aux composants ou aux connexions du graphe (τ pour un composant, τ pour une connexion) créant une mémoire à long terme sur la totalité du processus de recherche de la fourmi, qui change dynamiquement durant la résolution afin de refléter l'expérience de recherche acquise par les agents, et une valeur heuristique η (η_i, η_{ij} respectivement) représentant une information sur l'instance du problème ou une information sur le temps d'exécution fourni par une source autre que les fourmis. Dans la plupart des cas, η est le coût ou une estimation du coût de l'extension de l'état courant.

3.5.3 La métaheuristique ACO

Dans les algorithmes ACO, les fourmis se déplacent en appliquant une politique de décision stochastique locale qui se base sur l'utilisation des traces de phéromone et d'une information heuristique spécifique au problème traité. Des coefficients α et β permettent de contrôler l'importance relative des deux éléments. Chaque fourmi possède une forme de mémoire afin d'obliger celle-ci à former une solution admissible.

En se déplaçant, les fourmis construisent incrémentalement des solutions au problème d'optimisation. Une fois qu'une fourmi a construit une solution, ou lorsque la solution est en cours de construction, la fourmi évalue la solution (partielle) et dépose la phéromone dans le composant ou la connexion qu'elle a utilisée. Cette information de phéromone dirigera la recherche des futures fourmis.

L'intensité de ces traces décroît dans le temps par un facteur constant appelé coefficient d'évaporation. D'un point de vue pratique, l'évaporation de phéromone est nécessaire

pour éviter une convergence prématurée de l'algorithme vers une région sous-optimale. Ce processus implémente une forme utile d'oubli favorisant l'exploration de nouvelles aires de recherche.

3.6 ACO et la recherche locale

Dans plusieurs applications à des problèmes d'optimisation combinatoire NP difficiles, les algorithmes ACO réalisent de meilleures performances lorsqu'ils sont combinés avec des algorithmes de recherche locale, Ces algorithmes optimisent localement les solutions des fourmis et ces solutions optimisées localement sont utilisées par la suite dans la mise à jour de phéromone. L'utilisation de la recherche locale dans les algorithmes ACO peut être très intéressante comme les deux approches sont complémentaires. En effet, la combinaison peut améliorer largement la qualité des solutions produites par les fourmis.

D'un autre côté, générer des solutions initiales pour les algorithmes de recherche locale n'est pas une tâche facile. Dans les algorithmes de recherche locale, où les solutions initiales sont générées aléatoirement, la qualité des solutions peut être médiocre. En combinant la recherche locale avec l'ACO, les fourmis génèrent stochastiquement, en exploitant les traces de phéromone, des solutions initiales prometteuses pour la recherche locale.

3.7 Contrôle de l'intensification/diversification

Un point critique, lors de l'application d'un algorithme ACO, est de trouver un équilibre entre l'exploitation de l'expérience de recherche acquise par les fourmis et l'exploration de nouvelles zones de l'espace de recherche non encore visitées. ACO possède plusieurs manières pour accomplir une telle balance, essentiellement par la gestion des traces de phéromone. En effet, les traces de phéromone déterminent dans quelles parties de l'espace de recherche les solutions construites auparavant sont localisées.

Une manière de mettre à jour la phéromone, et pour exploiter l'expérience de recherche acquise par les fourmis, est de déposer une quantité de phéromone fonction de la qualité de la solution trouvée par chaque fourmi. Ainsi, les zones correspondant aux meilleures solutions recevront une quantité de phéromone plus élevée et seront plus sollicitées par les fourmis durant leurs déplacements.

Pour contrôler la balance entre l'exploitation de l'espace de recherche et l'exploration de nouvelles zones, ACO dispose de plusieurs paramètres pour gérer l'importance rela-

tives des traces de phéromone, essentiellement les deux paramètres α et ρ.

α détermine le poids des traces de phéromone dans le calcul des probabilités de transition, et ρ détermine l'évaporation de phéromone. Ces deux paramètres ont une influence sur le comportement exploratoire ; pour favoriser la stratégie d'exploration, on peut diminuer le coefficient α , ainsi les fourmis deviennent moins sensibles aux phéromones lors de leurs déplacements et peuvent visiter des zones non explorées, ou diminuer ρ , ainsi la phéromone s'évapore plus lentement et l'intensité des traces de phéromone devient similaire sur les arêtes et donc les fourmis deviennent moins influencées par la phéromone.

Pour favoriser la stratégie d'exploitation, on augmente les valeurs respectives de α et/ou de ρ , ainsi les fourmis seront plus guidées par la phéromone vers des zones 'prometteuses' de l'espace de recherche. Après un certain nombre d'itérations, toutes les fourmis vont converger vers un ensemble de 'bonnes' solutions. Ainsi, on favorise une convergence plus rapide de l'algorithme.

Toutefois, il faut faire attention et éviter une intensification trop forte dans les zones qui apparaissent prometteuses dans l'espace de recherche car cela peut causer une situation de stagnation : une situation dans laquelle toutes les fourmis génèrent la même solution.

Pour éviter une situation pareille de stagnation, une autre solution serait de maintenir un niveau raisonnable d'exploration de l'espace de recherche. Par exemple, dans ACS les fourmis utilisent une règle de mise à jour locale de phéromone durant la construction de solution pour rendre le chemin qu'elles viennent de prendre moins désirable pour les futures fourmis et ainsi, diversifier la recherche. MMAS introduit une limite inférieure pour l'intensité des traces de phéromone pour garantir toujours la présence d'un niveau minimal d'exploration. MMAS utilise aussi une réinitialisation des traces de phéromone, qui est une manière de renforcer l'exploration de l'espace de recherche.[27]

3.8 Conclusion

Dans ce chapitre, nous avons présenté les fondements théoriques essentiels pour comprendre le principe de l'optimisation par colonies de fourmis. Nous avons expliqué les origines de l'inspiration biologique de cette approche, et nous avons présenté son modèle mathématique de base. On a constaté que :

– ACO est une méthode stochastique qui requiert de plus en plus l'attention de la communauté scientifique. Cette métaheuristique a prouvé sa performance pour plusieurs problèmes d'optimisation combinatoire NP-difficiles.

- L'utilisation des traces de phéromone permet d'exploiter l'expérience de recherche acquise par les fourmis et ainsi renforcer l'apprentissage pour la construction de solutions dans les itérations futures de l'algorithme.En même temps, l'information heuristique peut guider les fourmis vers les zones prometteuses de l'espace de recherche.
- Un autre avantage de ACO est que son domaine d'application est vaste. En principe, ACO peut être appliquée à n'importe quel problème d'optimisation discret pour lequel un mécanisme de construction de solution peut être conçu, Dans le cadre de ce PFE, nous verrons dans le sixième chapitre comment adapter l'optimisation par colonies de fourmis au problème de planification de trajectoire pour les robots mobile.
- Toutefois, ACO a ses propres inconvénients qui résident, principalement, dans les problèmes de paramétrage. En effet, ACO nécessite une instanciation des différents paramètres qui dépend du problème. Généralement, les paramètres correspondant aux meilleures solutions prouvées expérimentalement sont retenus.

Chapitre 4

Optimisation par essaim particulaire (PSO)

4.1 Introduction

L'optimisation par essaims particulaires (OEP) est une métaheuristique, inventée par Russel Eberhart (Ingénieur en électricité) et James Kennedy (socio psychologue) en 1995.

Cette méthode s'inspire à l'origine du monde du vivant ; elle s'appuie notamment sur un modèle développé par le biologiste Craig Reynolds à la fin des années quatre-vingt, permettant de simuler le déplacement d'un essaim (inspirer du comportement social des animaux évoluant en essaim). Une autre source d'inspiration, revendiquée par les auteurs est la socio-psychologie.

Cette méthode d'optimisation se base sur la collaboration des individus entre eux. Elle est d'ailleurs une méthode récente parmi les algorithmes évolutionnaires, qui s'appuient sur le concept d'auto-organisation. Cette idée veut qu'un groupe d'individus peu intelligent puisse posséder une organisation globale complexe.

L'échange d'information entre eux fait que, globalement, ils arrivent néanmoins à résoudre des problèmes difficiles, comme c'est le cas, par exemple, chez les abeilles vivant en essaim (exploitation de sources de nourriture, construction de rayon, etc.).

Ainsi grâce à des règles de déplacement très simple (dans l'espace de solution), les particules peuvent converger progressivement à un minimum global. Cette métaheuristique semble cependant mieux fonctionner pour des espaces en variables continues.

L'optimisation par essaims particulaires (OEP), dans sa version historique, est donc une méthode itérative collective anarchique au sens original du terme, mettant l'accent

sur la coopération, partiellement aléatoire et sans sélection. Ça sera l'objet de ce chapitre dont on détaillera ses caractéristiques et les formaliser afin d'obtenir un modèle exploitable.

4.2 Origine

Vers 1927, Karl Von Frish a découvert que les abeilles ramenaient à la ruche non seulement du nectar et du pollen, mais aussi de l'information. Il a patiemment décodé leur langage et l'observateur attentif peut maintenant les comprendre partiellement.

L'optimisation par essaim particulaire (OEP) est une méthode née aux Etats Unis sous le nom de Particle Swarm Optimization (PSO).

Initialement les deux concepteurs, Russel Eberhart et James Kennedy cherchaient à modéliser des interactions sociables entre des «agents» devant atteindre un objectif donné dans un espace de recherche commun, chaque agent ayant une certaine capacité de mémorisation et de traitement de l'information.

La règle de base et qu'il ne devait y avoir aucun chef d'orchestre, ni même aucune connaissance par les agents de l'ensemble des informations seulement des connaissances locales.

Dès les premières simulations, le comportement collectif de ces agents évoquait celui d'un essaim d'êtres vivants convergeant parfois en plusieurs sous-essaims vers des sites intéressants. Ce comportement se trouve dans bien d'autres modèles, explicitement inspirés des systèmes naturels. Ici la métaphore la plus pertinente est probablement celle de l'essaim d'abeilles, particulièrement, du fait qu'une abeille ayant trouvé un site prometteur sait en informer certaines de ces consœurs et que celles-ci vont tenir compte de cette information pour leur prochain déplacement, c'est-à-dire une abeille doit parfois combiner plusieurs informations, celle correspondant à sa propre connaissance du terrain et celle apportée par une butineuse, voire plusieurs presque simultanément. La manière dont elle le fait reste un mystère mais pour nous inspirer du comportement de nos abeilles, il nous faudra quand même la modéliser, donc, inventer une méthode de toute pièce. Le modèle s'est révélé être trop simple pour vraiment simuler un comportement social, mais par contre très efficace en tant qu'outil d'optimisation.

FIGURE 4.1: Etude du comportement des abeilles dans une ruche.[18]

Le fonctionnement d'OEP fait qu'elle peut être classée dans les méthodes itératives (on approche peu à peu de la solution) et stochastiques (on fait appel au hasard). Sous ce terme un peu technique, on retrouve un comportement qui est aussi vieux que la vie elle-même : améliorer sa situation on se déplaçant partiellement au hasard et partiellement selon des règles prédéfinies.

4.3 Description et Principe général de l'OEP

La métaheuristique que nous avons étudiée dans le cadre de notre projet est l'optimisation par essaim de particules. On dispose d'une fonction objective à optimiser dans un sens ou dans l'autre.

Un essaim est un ensemble de particules positionnées dans l'espace de définition de la fonction objective. Le principe de l'algorithme consiste à déplacer ces particules dans l'espace de définition afin de trouver la solution optimale.

Une particule est caractérisée par plusieurs attributs :

– sa position actuelle : c'est-à-dire ses coordonnées dans l'ensemble de définition et la valeur de la fonction objective lui correspond.

– sa meilleure position : c'est la valeur obtenue par la particule et ses coordonnées.
– sa vitesse : cette donnée, recalculée à chaque itération de l'algorithme permet de déduire la position suivante de la particule. Elle est fonction de la meilleure position de la particule depuis le début de la recherche, du voisin le mieux positionné à l'instant actuel et de la vitesse précédente de la particule.
– ses voisins : c'est un ensemble de particule qui influe sur ses déplacements, en particulier celui qui est le mieux positionné.

4.4 Formulation

En OEP, un "site intéressent" correspond à un optimum au moins local d'une certaine fonction définie dans un espace de recherche. Cette fonction peut être donnée par une formule mathématique ou, à défaut, par un algorithme, voir par le résultat du déroulement d'un processus, réel ou simulé. Le tout est que l'on sache calculer sa valeur en chaque point.

Pour sa version classique l'OEP cherche les sites les plus intéressants c'est-à-dire l'optimum global de notre fonction fitness. Pour ce faire, l'OEP s'inspire du comportement coopératif décrit dans notre métaphore : chaque particule est capable de communiquer à certaines autres particules la position et la qualité du meilleur site qu'elle connait, qualité que l'on peut interpréter comme étant sa valeur.

4.4.1 Taille de l'essaim

Tout d'abord, il faut définir un essaim dans l'espace de recherche, mais de quelle taille ?, Les véritables essaims d'abeilles comptent typiquement 20000 individus, nous nous contenterons de taille de l'ordre de 20 à 40. Il s'avère en effet qu'en OEP ces tailles sont très souvent suffisantes.

Ce qui compte plutôt c'est le nombre de fois ou la fonction fitness doit être évaluée, car dans la plupart des problèmes réels, cette évaluation nécessite un temps non négligeable, et évidemment, pour une itération, ce nombre d'évaluations est égal au nombre de particules. Donc si nous voulons réduire le nombre total d'évaluations nécessaires pour trouver une solution, nous sommes au contraire tentés de diminuer la taille de l'essaim. Mais un essaim trop petit risque de mettre très longtemps pour trouver une solution ou même ne pas la trouver du tout.

Donc il y a un compromis à trouver. Les expérimentateurs ont proposé des tailles de l'ordre de 20 à 30 particules qui, en effet, se révèlent tout à fait suffisante pour résoudre

la quasi-totalité des problèmes de test classiques.

4.4.2 Liens d'information

Il faut définir, pour chaque particule, quelles sont ses informatrices. Toujours par analogie avec ce qui se passe dans une ruche, nous pouvons définir au hasard pour chaque particule son groupe d'informées, ce qui, automatiquement, détermine également les informatrices de chaque particule, puisque, formellement, nous établissons un graphe de relations entre les particules.

Combien d'informées, combien d'informatrices ? D'une part, si toutes les particules sont informées par chacune, toute l'information acquise à chaque instant est diffusée immédiatement, ce qui semble favorable. Mais d'autre part le risque est grand d'avoir un comportement trop uniforme : avec les mêmes informations, les particules vont agir de la même manière. Pour les recherches difficiles ce n'est pas efficace. Inversement, si chaque particule n'a que trop peu d'informatrices, nous pourrons obtenir des comportements plus diversifiés, mais le risque est alors que l'information soit mal transmise. Or il est important que si une particule trouve un bon emplacement, toutes les autres, plus au moins directement, puissent avoir connaissance de cette information, pour en tirer parti.

Dans ces conditions, si le choix est fait au hasard à chaque pas de temps, prendre deux ou trois informées pour chaque particule semble un bon compromis. Il y a aussi d'autres versions de l'OEP qui ne font pas ce choix au hasard mais selon une règle tenant compte de certains critères, par exemple une sélection automatique permanente et judicieuse des informatrices.

La nature des informations transmises est évidemment importante, mais plus il y aura d'informations, plus il sera long et difficile à une particule de les traiter. Le plus délicat à modéliser est la manière dont une particule informée va calculer son prochain déplacement. Tout d'abord, notons qu'elle est en général déjà en train de se déplacer : elle possède donc une certaine vitesse. Ensuite, puisqu' elle est informatrice éventuelle d'autres particules, elle connait sa propre meilleure performance. Enfin, elle connait toutes les meilleures performances de ces informatrices et elle va garder la meilleure.

Il nous reste donc trois éléments à combiner : vitesse propre, meilleure performance propre et meilleure des meilleures performances des informatrices.

Imaginons trois cas extrêmes : premier cas, la particule est aventureuse et n'entend suivre que sa propre voie, alors elle va attribuer une confiance nulle aux informations reçues et même à sa propre expérience ; elle se contentera de suivre plus au moins la direction déjà suivie, c'est-à-dire que le prochain déplacement se fera en gros avec la même

vitesse (intensité et direction) que le précédent. Deuxième cas, elle est très conservatrice ; elle va accorder une grande confiance à sa meilleure performance et tendra à y revenir sans cesse. Troisième cas elle ne s'accorde à elle-même aucune confiance ; elle se déplacera selon les indications de sa meilleure informatrice. La figure suivante (figure 4.2) représente les trois éléments fondamentaux pour le calcul du prochain déplacement d'une particule.

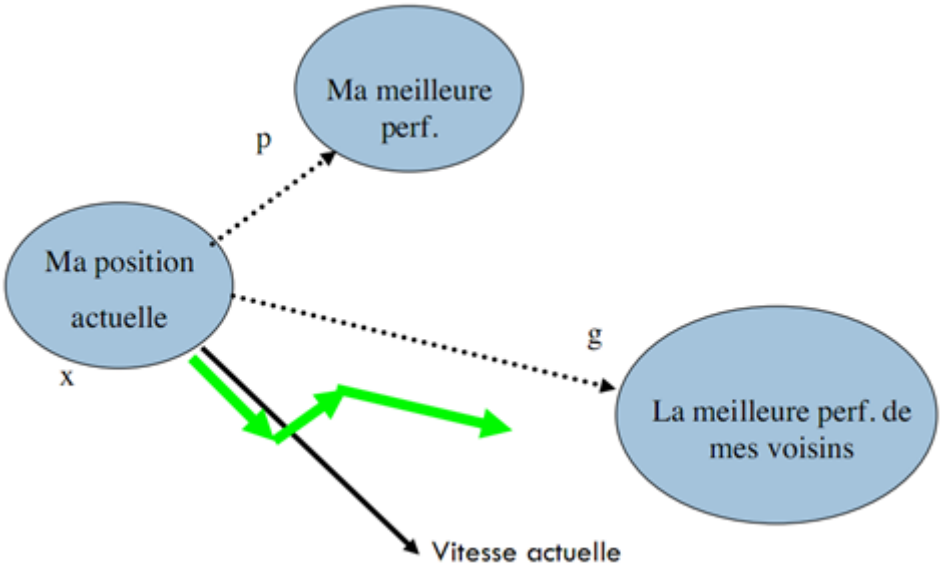


FIGURE 4.2: détermination de la nouvelle position d'une particule dans un processus OEP (les trois flèches grisées représentent les trois effets pris en compte).

A partir des quelques informations dont elle dispose, une particule doit décider de son prochain mouvement, c'est-à-dire décider de sa nouvelle vitesse. Pour ce faire, elle combine linéairement trois informations :

- sa vitesse actuelle ;
- sa meilleure performance ;
- la meilleure performance de ses voisines (ses informatrices).

À l'aide de trois paramètres parfois appelés coefficients de confiance, qui pondèrent trois tendances :

- tendance à suivre sa propre voie ;
- tendance conservatrice (revenir sur ses pas) ;
- tendance à suivre le meilleur voisin.

4.4.3 Initialisation

L'initialisation consiste simplement à placer d'abord au hasard les particules selon une distribution uniforme dans l'espace de recherche. Ceci est une étape que l'on retrouve dans à peu près tous les algorithmes d'optimisation itérative stochastique.

En pratique, il n'est pas souhaitable que trop de particules tendent à sortir de l'espace de recherche dès le premier pas de temps, ni d'ailleurs plus tard. Pour les premières formulations nous nous contentons de tirer au hasard les valeurs des composantes de chaque vitesse, selon une distribution uniforme dans :

$$[(x_{min} - x_{max})/2, (x_{max} - x_{min})/2] \tag{4.1}$$

4.4.4 Equations du mouvement

Une caractéristique très importante de l'OEP du moins dans ses versions classiques est qu'elle ne pratique aucune sélection. L'idée est en effet que les faibles d'aujourd'hui sont peut-être les forts de demain. Les particules à piètre performance sont conservées, avec l'espoir que parmi elles se trouvent précisément les originales, les dissidentes qui permettront de découvrir le meilleur site existant dans l'espace de recherche. Les expérimentations ont d'ailleurs parfaitement justifié cet espoir.

L'espace de recherche est de dimension D. La position courante d'une particule dans cet espace à l'instant t est donc donnée par un vecteur $x(t)$ à D composantes. Sa vitesse courante est $v(t)$. La meilleure position trouvée jusqu'ici par cette particule est donnée par un vecteur $p(t)$. Enfin, la meilleure position trouvée par les informatrices de la particule est indiquée par un vecteur $g(t)$. On indique par l'indice d, la $d_{ième}$ composante de chacun des vecteurs $x(t), v(t), p(t)$ et $g(t)$. Avec ces notations, les équations de mouvement d'une particule sont :

$$\begin{cases} v_d &= c_1 v_d + c_2(p_d - x_d) + c_3(g_d - x_d) \\ x_d &= x_d + v_d \end{cases} \tag{4.2}$$

Les coefficients de confiances sont définis de la manière suivante :

- $c_1$est constante (confiance en son propre mouvement) ;
- c_2 et c_3 (respectivement confiance en sa meilleure performance et en celle de sa meilleure informatrice) sont choisis au hasard à chaque pas de temps selon une distribution uniforme dans un intervalle $[0, c_{max}]$ donné.

C'est pourquoi le système (4.2) peut être réécrit de manière plus explicite, en mettant

en évidence les systèmes aléatoires les éléments aléatoires

$$\begin{cases} v_d & = & c_1 v_d + c_{max} alea[0,1](p_d - x_d) + c_{max} alea[0,1](g_d - x_d) \\ x_d & = & x_d + v_d \end{cases} \tag{4.3}$$

Pour utiliser ce modèle, on doit définir les deux paramètres c_1 et c_{max}, ce dernier peut être vu comme la confiance maximale accordée par la particule à toute performance transmise par une autre particule. Pour chaque problème les « bonnes » valeurs ne peuvent être trouvées qu'expérimentalement, à l'aide cependant de deux réglage, dégagés après de nombreux tests.

La première règle stipule que c_1 doit être de valeur absolue inférieure à 1. Elle se comprend intuitivement si l'on considère ce qui se passe lors de plusieurs pas de temps successifs, dans le cas très particulier où la particule est la meilleure informatrice d'elle-même. Nous avons alors en effet et à chaque pas de temps la vitesse est simplement multipliée par c_1. S'il est de valeur absolue supérieure à 1, la vitesse augmente sans cesse et la convergence est impossible. Notons qu'en théorie rien n'interdit à ce coefficient d'être négatif ; le comportement obtenu étant alors fortement oscillatoire, mais ce n'est jamais le cas en OEP classique. Nous le supposerons donc positif.

En pratique ce coefficient ne doit pas être ni trop petit, ce qui induit une convergence prématurée, ni trop grand, ce qui au contraire peut ralentir exagérément la convergence. Les auteurs des premiers travaux sur l'OEP préconisaient de la prendre égale à 0,7 ou 0,8.

La deuxième règle indique simplement que le paramètre c_{max} ne doit pas être trop grand, une valeur de l'ordre de 1,5 à 1,7 étant considérée comme efficace dans la plupart des cas. A l'époque où elle a été énoncée, cette règle n'avait pas de justification, même intuitive. Elle était purement expérimentale.

En effet, les valeurs préconisées sont très proche de celles déduites plus tard d'analyses mathématique montrant que pour une bonne convergence les valeur de c_1 et c_{max} ne doivent pas être choisies indépendamment, par exemple les couple de valeurs (0,7 1,47) et (0,8 1,62) sont effectivement corrects. Les premiers expérimentateurs ; James Kennedy et Russel Eberhart, auxquels on peut ajouter Yuhui Shi, ont fait un bon travail.

4.4.5 Confinement d'intervalle

Lors des premières expérimentations de l'OEP les fonctions utilisées étaient définies pour toutes valeurs. Lors de l'évolution de l'essaim il pouvait arriver qu'une particule sorte de l'espace de recherche initialement défini, mais cela était sans importance puisque

la valeur de sa position pouvait en fait encore être calculée, sans planter l'exécution.

Cependant, cela n'est pas toujours le cas. Par exemple dans la plupart des langages de programmation et avec la plupart des compilateurs, l'évaluation d'une fonction telle que :

$$f(x) = \sum_{d=1}^{D} \sqrt{x_d} \tag{4.4}$$

Renvoie un message d'erreur dès que l'une des coordonnées x_d est négative.

Par conséquent, il a fallu très vite ajouter un mécanisme pour éviter qu'une particule sorte de l'espace de recherche. Le plus simple de tous est le confinement d'intervalle. Supposons toujours, par simplicité, que l'espace de recherche soit $[x_{min}, x_{max}]^D$ alors ce mécanisme stipule que si une coordonnée xd calculée selon les équations de mouvement sort de l'intervalle $[x_{min},x_{max}]$ on lui attribue en fait la valeur du point frontière le plus proche. En pratique cela revient à remplacer la deuxième ligne du système (4.3) par :

$$x_d = Min(Max(x_d + v_d, x_{min}), x_{max}) \tag{4.5}$$

Cette forme simple, quoique donnant des résultats corrects, a cependant un inconvénient. En effet, nous sommes dans un cas de figure où la vitesse propre de la particule tend à la faire sortir de l'espace de recherche. Le confinement ramène la particule à la frontière de l'espace de recherche mais ne change pas sa vitesse. Celle-ci est donc en générale modifiée au prochain pas de temps, mais il n'est pas rare qu'elle reste plus ou moins orientée de la même manière, alors la particule tendra à nouveau à dépasser la frontière, puis elle sera ramenée par le confinement, etc. En pratique, tout se passera comme si elle était « collée » à cette frontière.

4.4.6 Structure de l'Algorithme

L'OEP est un algorithme à population. Il commence par une initialisation aléatoire de l'essaim dans l'espace de recherche. A chaque itération de l'algorithme, chaque particule est déplacée suivant les équations de mouvement données précédemment (4.3). Une fois le déplacement des particules effectué, les nouvelles positions sont évaluées. Les p_i ainsi que q_i sont alors mis à jour. Cette procédure est résumée par l'algorithme suivant. N est le nombre de particules de l'essaim.

Le critère d'arrêt peut être différent suivant le problème posé. Si l'optimisation globale est connue à priori, on peut définir une "erreur acceptable" comme critère d'arrêt. Sinon, il est commun de fixer un nombre maximum d'évaluations de la fonction objective ou un

nombre maximum d'itérations comme critère d'arrêt. Cependant, au regard du problème posé et des exigences de l'utilisateur, d'autres critères d'arrêt peuvent être utilisés.[7]

Algorithme d'optimisation par essaim particulaire

Initialisation aléatoire des positions et des vitesses de chaque particule
Pour chaque particule i, $p_i = x_i$
Tant que le critère d'arrêt n'est pas atteint **faire**
Pour $i = 1 à N$ **faire**
Déplacement de la particule

$$\begin{cases} v_d & = & c_1 v_d + c_2(p_d - x_d) + c_1(g_d - x_d) \\ x_d & = & x_d + v_d \end{cases}$$

Evaluation des positions
Si $f(x_i) = f(p_i)$
Fin Si
Groupe d'informatrices= {choisir 03 particule aléatoirement}
$g = min(f(groupe' informatrices))$
Fin pour
Fin tant que

Organigramme

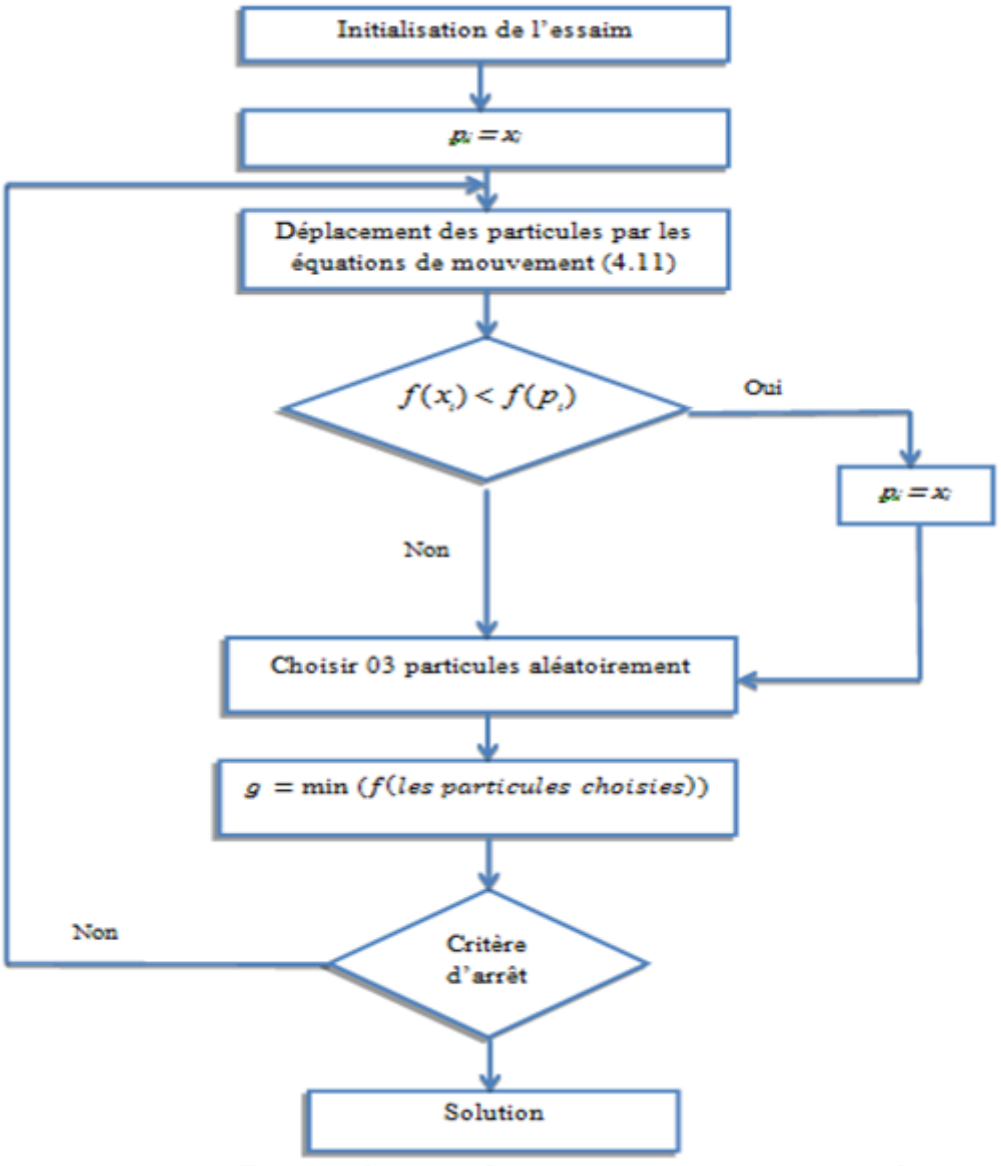


FIGURE 4.3: Organigrame de l'algorithme OEP.

4.5 Conclusion

Nous avons consacré ce chapitre à l'intérêt d'optimisation par la méthode des essaims particulaires OEP. Cette nouvelle méthode inspirée du comportement et déplacement d'animaux en essaim, a rencontré un vif succès depuis sa création. Sa relative simplicité et son efficacité en font un des algorithmes les plus utilisés de nos jours. De nombreux axes de recherche ont pour objet de tirer la meilleure partie du programme d'optimisation par essaim particulaire. D'abord nous avons commencé par citer son origine ainsi que son principe, puis nous avons présenté son modèle mathématique. Après, nous avons parlé d'un algorithme itératif adaptatif qui modifie son comportement en fonction de sa découverte progressive.

Chapitre 5

Formulation du problème de planification

5.1 Introduction

Dans ce chapitre, nous allons présenter l'espace de travail dans lequel on a adapté les algorithmes de planification ACO, A*, PSO ; nous rappelons que cette méthode fait partie de l'approche de décomposition cellulaire citée dans le premier chapitre, elle consiste à représenter l'espace de configuration des robots mobiles en grille et le chemin planifié est obtenu au moyen d'une recherche de graphe.

5.2 L'espace de travail

L'espace de travail correspond à l'espace réel physique dans lequel opère le robot. Mathématiquement, ce sera toujours un sous-ensemble de R3. Dans le cas du déplacement d'un robot mobile dans le plan (voiture, robot à roues. . .), il est possible de se ramener à un espace de dimension 2 en projetant les obstacles au sol. Cet espace peut être également de dimension 3 si le robot considéré se déplace dans l'espace, par exemple un drone.

On considère le robot mobile comme un point dont l'orientation n'est pas prise en compte. Dans la Figure 5.1, le robot se déplace sur un graphe dont les nœuds (les points noirs) représentent les points atteignables par le robot. Une configuration du robot est donc décrite par le couple $q = (x, y)$. Sa configuration initiale est la configuration $q_{init} = (2, 0)$. L'espace des configurations est défini par l'ensemble suivant :
$\{(i, j)|i\epsilon 0, 1, ..., 7, j\epsilon 0, 1, ..., 5\}$ Cet exemple contient donc un nombre fini de configurations possibles : 48 dans ce cas.

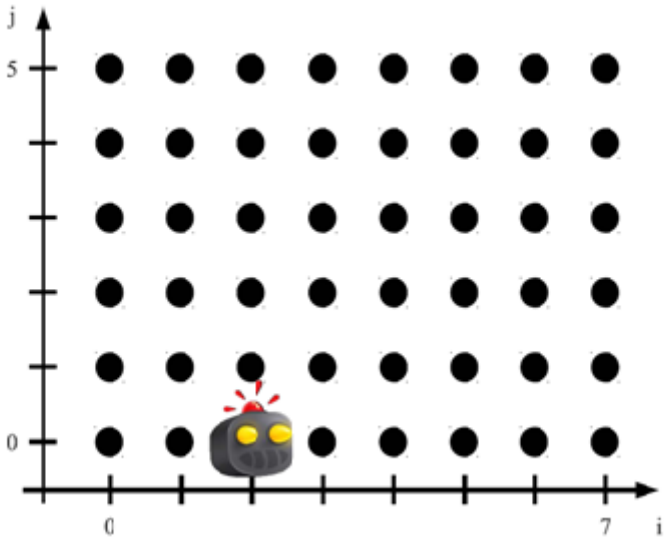


FIGURE 5.1: Espace de travail du robot mobile.

L'espace de travail contient donc directement toutes les positions admissibles du robot.

5.3 L'espace des configurations libres

Lors d'une planification d'un chemin, l'une des contraintes qui est posée dans le planificateur de trajectoire est de déterminer un chemin qui évite les obstacles. L'espace des configurations libres représente donc l'ensemble des configurations pour lesquelles les points correspondants dans l'espace de travail ne rentrent pas en collision avec un obstacle. L'espace des configurations libres est noté $C_{free} \subset C$. La navigation dans un environnement avec obstacles mobiles est un problème encore plus difficile : l'espace des configurations libres évolue alors en fonction du temps.

Il est donc facile de retrouver les configurations libres à partir de l'espace de travail. D'après la Figure 5.2, les positions grisées entrent en collision avec les obstacles dans l'espace de travail. Ainsi, l'espace des configurations libres est alors :

$$C_{free} = C–(1,2),(2,2),(3,2),(4,4),(5,4),(6,4)$$

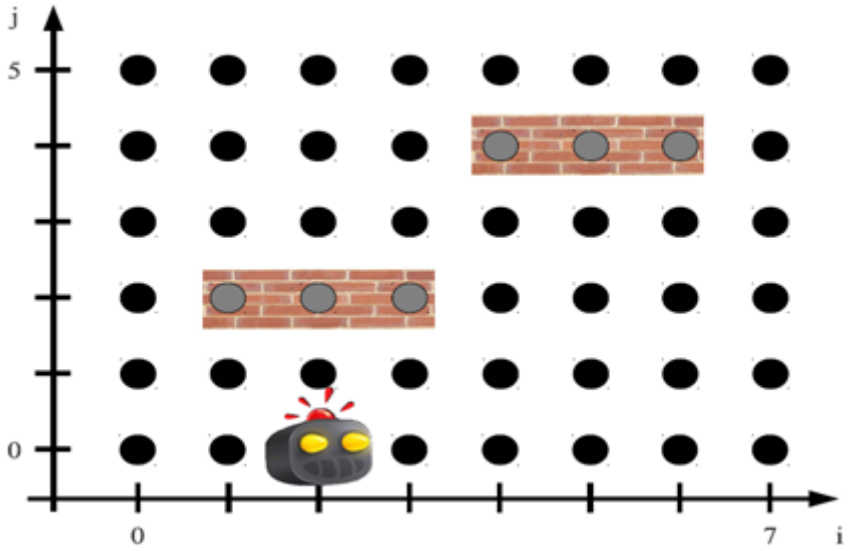


FIGURE 5.2: Exemple d'espace de configuration libre.

5.4 Empattement du robot

Dans le cas où le robot est ramené à un point dans le processus de planification de trajectoire, il faut tenir compte de son empattement réel pour assurer l'évitement des obstacles présents dans l'environnement. Soit r le rayon du disque englobant la surface réelle du robot (projetée au sol).

Le point de référence sur cette surface est également le centre du cercle. La taille de tous les obstacles est alors augmentée du diamètre $2 \times r$ (région qui sera généralement grisée sur les schémas). Cette règle permet de garantir l'évitement des obstacles en toute situation. Toutefois, elle surestime largement la taille du robot : certaines configurations, pourtant acceptables dans l'espace de travail, peuvent devenir inaccessibles, comme le montre la Figure 5.3. La gestion de la forme exacte est un problème extrêmement complexe à formuler dans l'espace des configurations.

Distance de sécurité

Le terme distance de sécurité est utilisé en référence à la distance d'évitement du robot aux obstacles fixes ou mobiles. Cette distance tient compte au minimum de l'empattement du robot. Toutefois, un planificateur de trajectoire prenant en compte cette seule distance peut générer des solutions amenant le robot à frôler les obstacles. La distance de sécurité dobst ajoute un écartement e supplémentaire au rayon r servant à l'estimation de la taille du robot. Cette distance de sécurité ainsi majorée permet d'obtenir des

solutions présentant des évitements plus larges. Compte tenu de l'incertitude qu'il peut y avoir lors des déplacements des robots en expérimentation, cet écartement se révèle souvent nécessaire.[13]

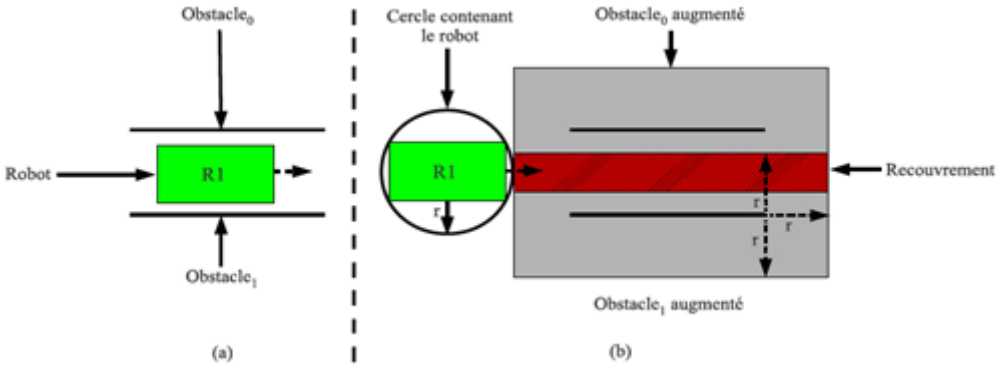


FIGURE 5.3: Défaut dans l'approximation de la taille d'un robot dans le cas d'un passage exigu : (a) si sa forme exacte est prise en compte, le robot peut passer dans le couloir et (b) si le robot est ramené à un point, soit r le rayon du disque recouvrant complètement sa surface, si la taille des obstacles est augmentée de ce rayon, alors le passage n'apparaît plus franchissable car les obstacles se recouvrent dans la surface hachurée en rouge.

5.5 Problème de planification de chemin

Pour déplacer le robot, il faut trouver une succession de configurations admissibles (succession de points) lui permettant d'atteindre son objectif. Le problème de planification de chemin correspond à la recherche d'une séquence de configurations appartenant à C_{free} depuis un état de départ Start jusqu'à un état d'arrivée Goal. Cette séquence est appelée chemin.

L'espace des configurations contient un ensemble de positions qui correspond au maillage d'une grille. Comme représenté par la Figure 5.4, l'état de départ correspond à la position du robot tel qu'il est représenté. Le chemin correspond à la séquence de points en bleu rejoignant le point d'arrivée représenté en rouge. Dans le cas discret, Le chemin *path* est une séquence discrète de $N + 1$ configurations libres ou points :

$$path : 0 \rightarrow (2, 0) = Start$$

$$1/N \rightarrow (3, 0),$$

$$2/N \rightarrow (4, 0),$$

$$3/N \rightarrow (4,1),$$

$$\ldots\ldots$$

$$N \rightarrow (5,5) = Goal$$

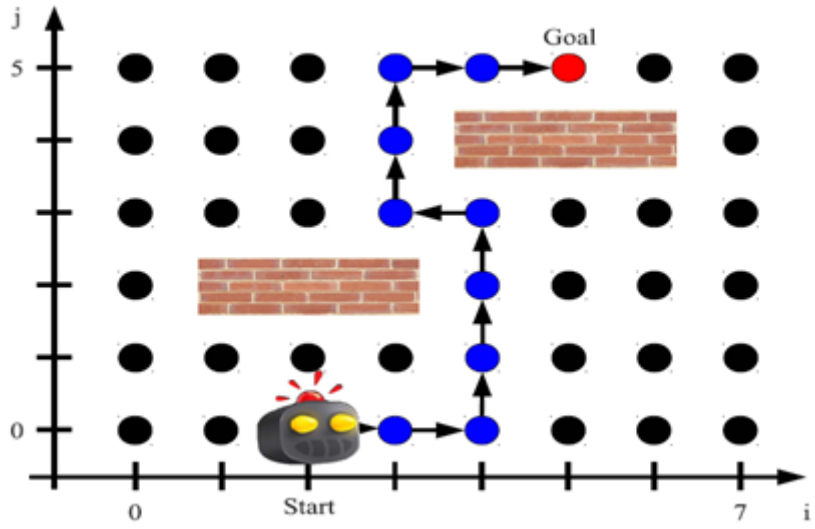


FIGURE 5.4: Chemin planifié.

Il existe plusieurs approches permettant d'obtenir le meilleur chemin reliant un état de départ "Star" à un état d'arrivée "Goal" tel que l'algorithme de Djikstra, A*, colonie de fourmis (ACO), essaim particulaire (PSO), colonie de bactérie, Artificial swarm fish (ASFA), système immunitaire, etc.

5.6 Conclusion

Dans ce chapitre nous avons défini l'espace de travail où nous allons appliquer les algorithmes : ACO au sixième chapitre, A* au septième chapitre, cet espace est un maillage régulier de l'environnement, et le robot mobile dans tous nos applications est considéré comme un point qui n'a pas d'orientation, et supposé qu'il a une connaissance parfaite sur l'environnement (positions d'obstacles, position de départ et du but).

Chapitre 6

Planification de trajectoire par ACO

6.1 Description de l'algorithme

L'algorithme de planification proposé ici utilise le concept original de l'algorithme d'ACO avec une certaine modification. L'idée de l'algorithme proposé est comme suit :

À partir d'un point de départ dans la grille, une fourmi se déplace itérativement d'un point à un de ses points voisins. Lorsqu' elle est au $i_{éme}$ point de la grille repéré par ces coordonnées (x, y), la fourmi k peut sélectionner le prochain point $(j_{éme})$ en choisissant un de ses 8 positions voisines :$[(x+1, y+1), (x+1, y), (x+1, y-1), \dots]\epsilon N$, où N en général est l'ensemble de tous les positions voisines de la position courante. La fourmi prend sa prochaine mesure aléatoirement, basé sur la probabilité donnée par :

$$P_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{l \epsilon N_j^k} \left([\tau_{il}(t)]^{\alpha} \times [\eta_{il}]^{\beta} \right)} & si \quad j \epsilon N_i^k \\ 0 & sinon \end{cases} \tag{6.1}$$

- où $\tau_{ij}(t)$ est la quantité de la phéromone accumulée sur l'arête (i, j) de la grille, lorsque le point i de la grille est la position courante de la fourmi k à l' instant t.
- N est le voisinage faisable de la fourmi k c'est à dire l'ensemble des points possibles lorsque la fourmi k est au $i_{éme}$ point.
- η_{ij} est le produit scalaire du vecteur $\overrightarrow{ij}$ formé par le point i et le point j, avec le vecteur $\overrightarrow{idest}$ formé par le point i et le point de destination.

– α, β sont deux paramètres qui déterminent l'influence relative de la trace de phéromone et de l'information heuristique.

Le premier terme dans le numérateur est associé à la quantité de phéromone dans les 8 points voisins (phéromone locale). Le deuxième terme est associé à une attraction globale. Le but global est de guider l'agent (fourmi) dans la direction désirée de sorte qu'un grand nombre de fourmis puissent atteindre le point but dans un nombre limité de pas.

(x-1,y+1)	(x,y+1)	(x+1,y+1)
(x-1,y)	**(x,y)**	(x+1,y)
(x-1,y-1)	(x-1,y)	(x+1,y-1)

FIGURE 6.1: La position actuelle d'un agent avec 8 positions possibles.

Lors de l'exécution d'un cycle de calcul relatif a un groupe de fourmis, nous disposons d'un ensemble de solutions finies représenté par les fourmis ayant atteint le but dans un nombre imposé de pas (selon la taille de la grille). Toutes les autres fourmis de ce groupe seront ignorées dans les étapes restantes de ce cycle.. Cependant, la phéromone sera déposée seulement si la fourmi atteint la position de destination en moins d'un certain nombre de pas. Ainsi, après qu'un groupe d'agents ait fini son excursion, la phéromone dans la carte entière sera mis à jour par :

$$\tau_{ij}(t+1) = (1-\rho) \times \tau_{ij}(t) + \sum_{k=1}^{M} \Delta\tau_{ij}^{k} \tag{6.2}$$

Où $0 < \rho < 1$ est le facteur d'évaporation de la phéromone. Ce paramètre protège la grille contre l'accumulation illimitée de la phéromone. $\Delta\tau_{ij}^{k}$ est la quantité de phéromone que la fourmi k dépose sur l'arête(i, j). Elle est définie comme :

$$\Delta\tau_{ij}^{k} = \frac{Q}{L^{k}} \tag{6.3}$$

Où Q est une constante de l'algorithme, $L^k(t)$ est la longueur de la trajectoire de la fourmi k. Quand la fourmi k ne peut pas atteindre la destination dans un certain nombre de pas, on prend $L^k(t) = \infty$.

6.2 Stratégie d'élitiste

L'idée principale de la stratégie d'élitiste est de donner un poids additionnel significatif à la meilleure trajectoire construite d'un groupe d'agents. En d'autres termes, chaque fois que les traces de phéromone sont déposées, les arêtes qui appartiennent à la

meilleure excursion du groupe (meilleur chemin) obtiennent une quantité additionnelle de phéromone. Pour ces points de grille, suivant l'équation (6.3) :

$$\tau_{ij} \leftarrow \tau_{ij} + \begin{cases} e\frac{Q}{L^{gb}} & si \quad (i,j)\epsilon T^* \\ 0 & sinon \end{cases} \tag{6.4}$$

Encore, le $L^{gb}(t)$ représente la longueur de la meilleur trajectoire, et e est un nombre positif qui renforce le meilleur chemin global, T^*est la meilleure trajectoire.[5]

Pseudo Code D'algorithme

Pour $itération = 1, 2.....N$
Pour$fourmi = 1, 2....K$
Pour $pas = 1, 2....M$

* Déterminer les points voisins de la position courante
* Calculer la probabilité pour chaque point voisin.
* Déplacer vers la nouvelle position suivant la probabilité (6.1).
* Enregistrer la position précédente dans un tableau.

Si (la position courante égale la destination)
alors Interrompre la boucle du pas
Fin de si

Fin de pour
* Enregistrer la distance de la trajectoire faite par la fourmi k.
* Calculer la quantité de phéromone qui est produite par la fourmi k, donnée Par la formule (6.3)

Fin de pour
* Mise à jour de la quantité de la phéromone dans la grille entière suivant la Formule (6.2)
Fin de pour

6.3 Résultats de simulation (planification par ACO)

Les simulations ont été faites avec MATLAB R2007b sur un ordinateur personnel de 4GO de RAM (DDR2) et un microprocesseur Intel duel core de fréquence 2.1Ghz.

Les obstacles peuvent être situés à n'importe quel endroit sur la carte excepté le point de départ et le point de destination. Les expériences ont été faites avec un nombre différent d'obstacles et de différentes formes, de différentes tailles de population (nombre de fourmis), et d'un nombre différent d'itérations. En outre, ces études incluent les effets de l'élitisme.

– **Expérience 1 :** Environnement sans obstacles.
On prend le nombre de fourmis=10, taille de la carte=20*20, nombre maximum de pas=2*taille de la carte, nombre d'itération=5,$\rho = 0.5, \alpha = 1, \beta = 2, Q = 10$.

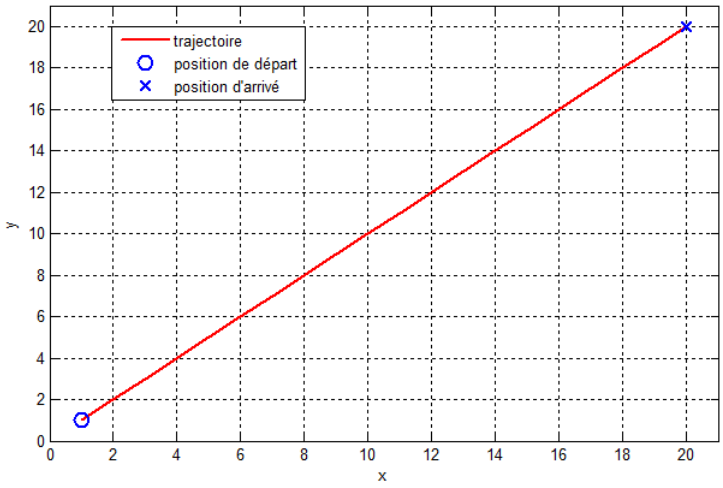


FIGURE 6.2: Meilleure trajectoire planifiée par ACO dans un environnement sans obstacles.

- Temps d'exécution est 4.10 s
- Longueur de la meilleure trajectoire est 26.87 um [1]

– **Expérience 2 :** Environnement contient des obstacles de forme cercles de même rayon, distribués régulièrement (on garde les mêmes paramètres).

1. um : unité de maillage.

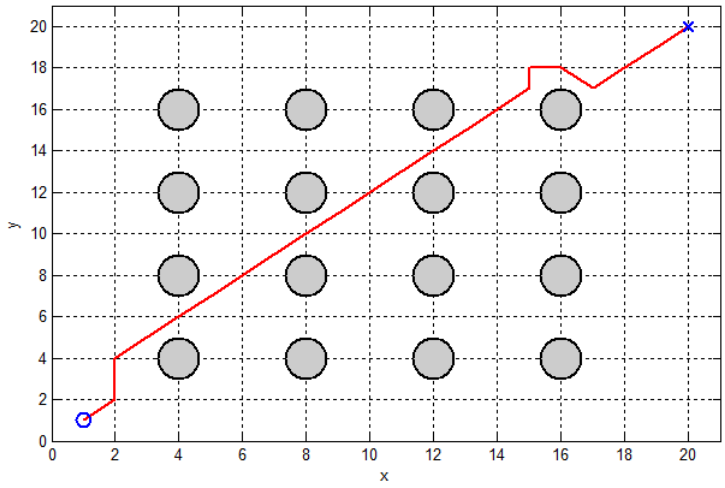


FIGURE 6.3: Meilleure trajectoire planifiée par ACO dans un environnement contient des obstacles de forme cercles réguliers.

- Le temps d'exécution est 6.43 s
- La longueur de la meilleure trajectoire est 29.46 um

– **Expérience 3 :** Environnement contient des obstacles de forme cercles de différents rayons (on garde les mêmes paramètres).

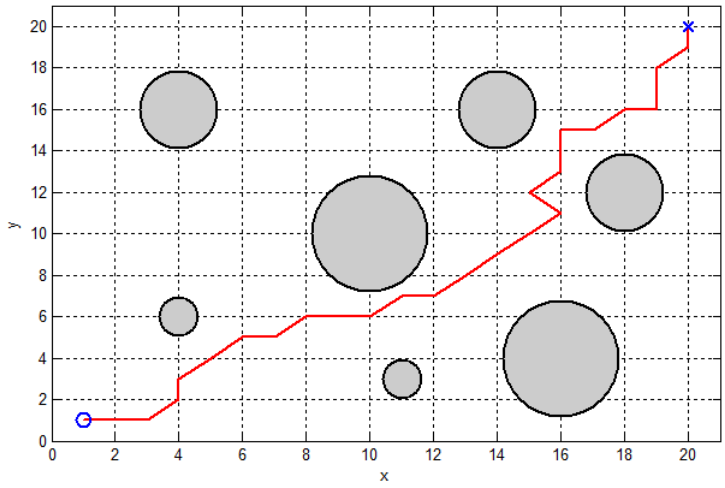


FIGURE 6.4: Meilleure trajectoire planifiée par ACO dans un environnement contient des obstacles de forme cercles irréguliers.

- Le temps d'exécution est 8.88 s
- La longueur de la meilleure trajectoire est 32.38 um

Remarque 1 : On peut appliquer cette modélisation à un environnement qui contient des obstacles de différentes formes et de différentes tailles ou chaque obstacle est remplacé par un cercle qui couvre entièrement l'obstacle.

Remarque 2 : dans les expériences 1,2 et 3 à cause de la simplicité de l'environnement, on a favorisé la stratégie d'exploration en diminuant le paramètre α qui détermine l'influence des traces de phéromone et le facteur d'oubli de phéromone ρ.

– **Expérience 4 :** (Environnement contient des murs comme obstacles).
Dans ce cas, l'expérimentation nous a obligé de changer les paramètres de l'algorithme afin d'obtenir de meilleurs résultats et des trajectoires faisables par le robot mobile. On a près les valeurs suivantes : $\rho = 0.7, \alpha = 0.5, \beta = 0.5, Q = 70$.

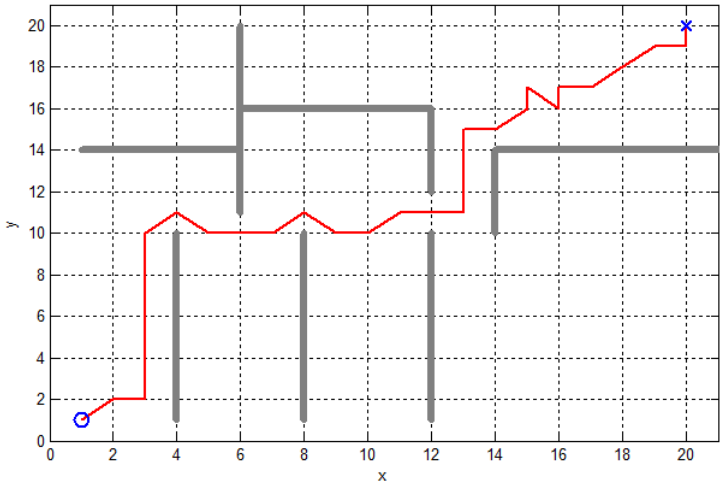


FIGURE 6.5: Meilleure trajectoire planifiée dans un environnement contient des murs comme obstacles.

- temps d'exécution est 7.52 s
- La longueur de la meilleure trajectoire est 38.14 um

– **Expérience 5 :** Robot mobile dans un labyrinthe (on garde les paramètres de l'algorithme précédent).

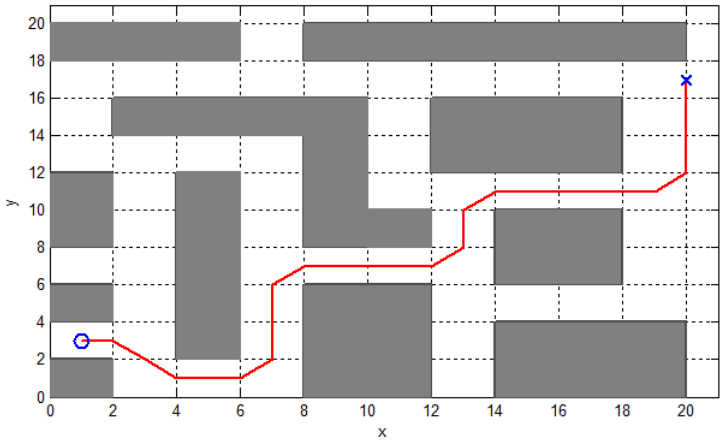


FIGURE 6.6: Meilleure trajectoire planifiée par ACO dans un labyrinthe.

-Le temps d'exécution est 8.22 s
-La longueur de la meilleure trajectoire est 32.90 um

	Exp 1	**Exp 2**	**Exp 3**	**Exp 4**	**Exp 5**
temps d'exécution d'ACO (s)	4.10	6.43	8.88	7.72	8.22
longueur de la trajectoire planifiée (um)	26.87	29.46	32.38	38.14	32.90

TABLE 6.1: Temps d'exécution et longueur de la meilleur trajectoire planifiée par ACO pour chaque expérience.

Analyse et discussion

On remarque que l'algorithme est capable de trouver des trajectoires acceptables dans n'importe quel environnement et dans un peu de temps (entre 4s et 9 s dans nos expériences), mais ce temps d'exécution reste très grande par rapport aux autre algorithmes de planification (par exemple A*) comme on le verra dans le septième chapitre, et les trajectoires générés restent moins lisses ce qui rend le travail du robot mobile difficile lors du suivi de ces trajectoires, pour remédier à ce problème on propose dans un premier temps un simple algorithme (on va l'appeler algorithme d'amélioration) .

6.4 Algorithme d'amélioration

Cet algorithme permet d'améliorer le lissage et la longueur du chemin de la trajectoire générée et cela par :

- annulation des points déviés de la trajectoire causés par de mauvaises décisions comme par exemple dans le cas où une fourmi déplace vers un point, mais le seul point suivant possible pour ce dernier est le point précédent donc la fourmi va faire un retour.
- annulation des points intermédiaires comme par exemple dans le cas où une fourmi déplace du point A vers le point B passant par le point C, ces point (A, B, C) forment un angle de 90°degré alors qu'elle peut déplacer directement du point A vers le point B diagonalement.

Le pseudo code de l'algorithme d'amélioration

******** annulation des points répétés ********

Initialisez le compteur à 1 ($i = 1$)
Tant que (i<nombre de points de la trajectoire)
Si (le point($i + 1$)=le point i) **alors**
Supprimez le point ($i + 1$)
Fin de si
Incrémentez le compteur
Fin de tant que

******** annulation des points déviés de la trajectoire **********

Réinitialisez le compteur ($i = 1$)
Tant que (i<nombre de points de la trajectoire-1)
Si (le point ($i + 2$)=le point i) **alors**
Supprimez le point ($i + 1$)
Fin de si

******** annulation des points intermédiaires ********

Si (l'ordonnée du point (i + 1)=l'ordonnée du point i et l'abscisse du point (i + 2)=l'abscisse du point ($i + 1$)) **alors**
Supprimez le point $i + 1$
Fin de si

Si (l'abscisse du point $i+1$=l'abscisse du point i et l'ordonnée du point $i+2$=l'ordonnée du point $i + 1$) alors
Supprimez le point $i + 1$
Fin de si

Incrémentez le compteur
Fin de tant que

On montre ci-dessous des exemples qui montrent l'effet de cet algorithme :

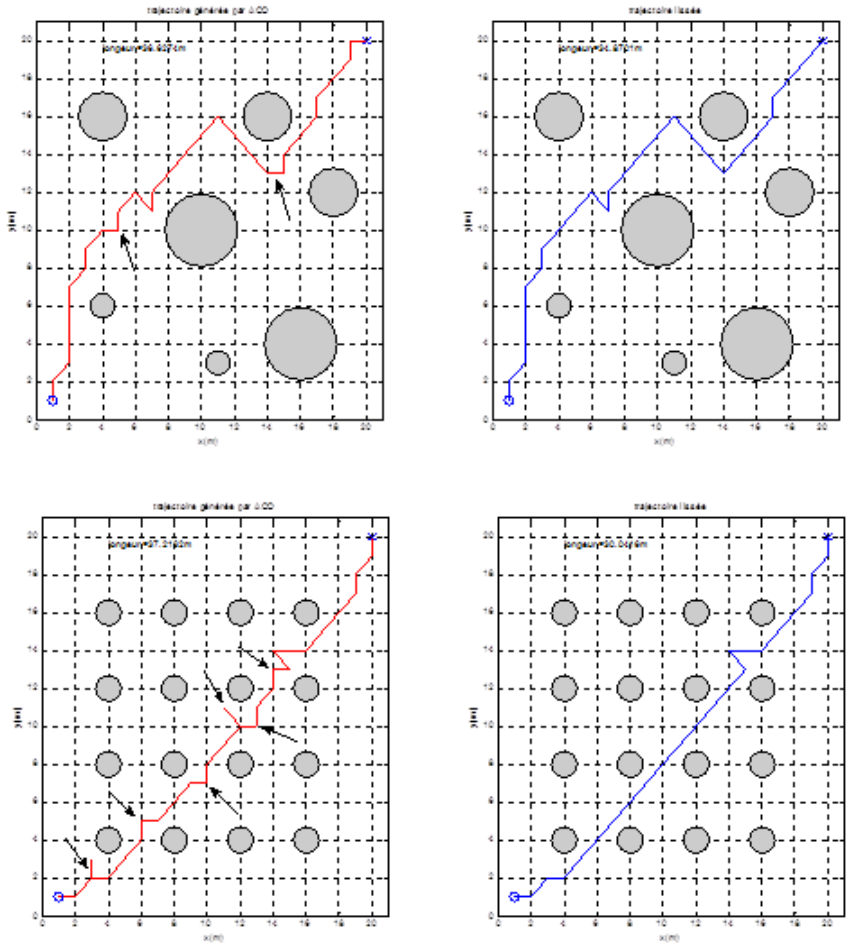


FIGURE 6.7: Exemples qui montrent la correction des trajectoires par l'algorithme proposé.

Remarque : Les flèches noires indiquent les endroits où la correction est appliquée.

Même avec ces corrections les trajectoires générées restent moins lisses et le robot vue à sa dynamique, il va rencontrer des difficultés lors du suivi de ces trajectoires, pour remédier encore une fois ce problème, nous proposons une technique efficace de lissage basée sur l'interpolation cubique spline.

6.5 Lissage du chemin

Le chemin planifié est un ensemble de coordonnées à traverser par le robot. Comme nous pouvons constater dans les figures précédentes chacun de ces points ou positions à être atteints par le robot, sont liés entre eux par des segments de ligne droite. Ceci fait en sorte que notre chemin présente une irrégularité élevée qui n'est pas désirée.

Pour faire face à ce problème, on a redéfini le chemin de base (chemin crée par les algorithmes de planification), en appliquant aux ensembles de points qui définissent le chemin de base, l'algorithme d'interpolation par spline cubique.

Nous pouvons nous rendre compte que le chemin créé par l'interpolation spline cubique, même s'il ne passe pas sur tous les points qui font partie du chemin de base, nous fournit une bonne approximation et demeure près du chemin initialement défini. En réduisant les changements abruptes de direction, nous obtenons un deuxième chemin moins accidente et plus facile à suivre en tenant compte de la cinématique des robots mobiles.

6.5.1 Interpolation par spline cubique

Idée générale :

Il est plus efficace de réaliser une interpolation polynômiale par morceaux, qu'une interpolation globale. On approche la courbe par morceaux (localement) et on prend des polynômes de degré faible (3) pour éviter les oscillations.

Définition :

On appelle spline cubique d'interpolation une fonction notée g, qui vérifie les propriétés suivantes :

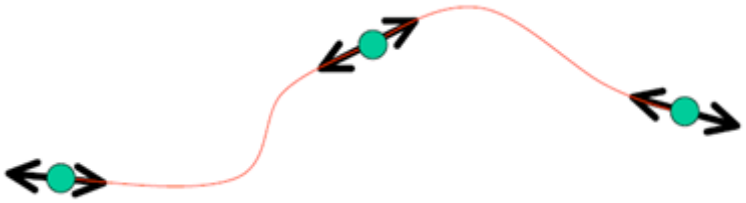
- $g \epsilon C^2[a;b]$ (g est deux fois continûment dérivable),
- g coincide sur chaque intervalle $[x_i, x_{i+1}]$ avec un polynôme de degré inférieur ou égal à 3,
- $g(x_i) = y_i$ pour $i = 0 \ldots n$.
- $g^{"}$ est de degré 1 et est déterminé par 2 valeurs : $m_i = g^{"}(x_i)$ et $m_{i+1} = g^{"}(x_{i+1})$ (moment au nœud $n^o i$).

Remarque : Il faut des conditions supplémentaires pour définir la spline d'interpolation de façon unique par exemple de conditions supplémentaires :

- $g^{"}(a) = g^{"}(b) = 0.$

Illustration :

$$P_2(x) = a_2(x - x_2)^3 + b_2(x - x_2)^2 + c_2(x - x_2) + d$$



$$P_1(x) = \alpha_1 x^3 + \beta_1 x^2 + \gamma_1 x + \delta_1 = a_2(x - x_2)^3 + b_2(x - x_2)^2 + c_2(x - x_2) + d$$

FIGURE 6.8: Illustration de l'interpolation polynomiale locale par cubique spline.

L'idée est de remplacer chaque segment entre deux points de la trajectoire par un polynôme d'ordre 2,3.., on présente ci-dessous les résultats obtenus

6.6 Résultats de simulation (lissage du chemin)

– Environnement contient des obstacles de forme cercles de différents rayons.

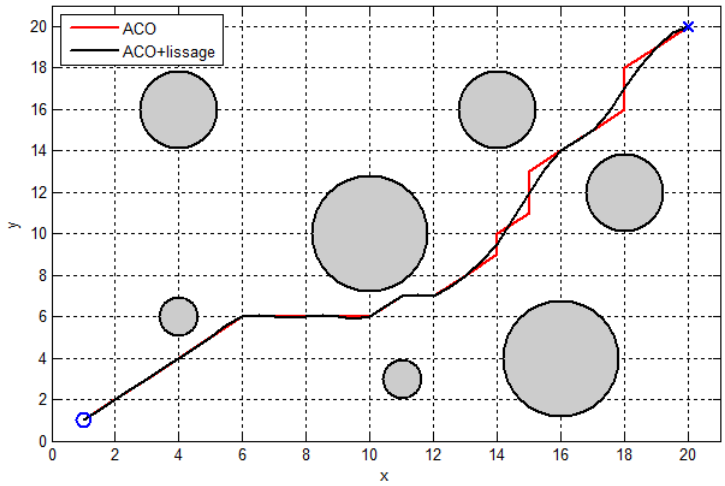


FIGURE 6.9: Trajectoire planifiée par ACO et lissée par cubique spline dans un environnement contient des obstacles de forme cercles irréguliers.

– Environnement contient des murs comme obstacles.

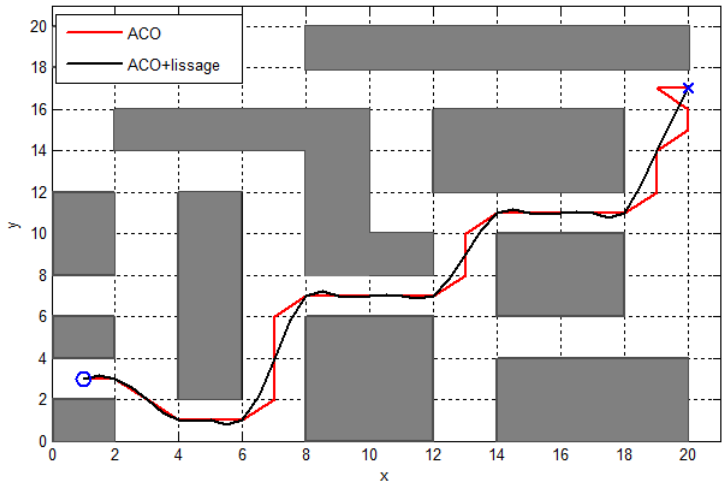


FIGURE 6.10: Trajectoire planifiée par ACO et lissée par cubique spline dans un labyrinth.

Remarque : le temps de calcul ajouté par l'inrodution du cubique spline est negligeable.

6.7 Conclusion

Dans ce chapitre, les résultats de la recherche détaillée sur l'algorithme ACO appliqué à un problème d'optimisation de chemin ont été présentés. Les résultats de simulation démontrent que cette approche offre une possibilité intéressante d'être une solution prometteuse au problème de planification des robots mobiles.

En fonction de la nature de l'environnement, les paramètres de l'algorithme ACO sont choisis, un robot mobile dans une situation moins complexe on le favorise la stratégie d'exploration et la stratégie d'exploitation dans les situations compliquées.

En dépit de la bonne exécution, quelques limitations étaient trouvées telles que le temps d'exécution qui reste encore grand à cause de la complexité de l'algorithme qui peut atteindre approximativement $O(NC_{max}.n^2.m)$ avec (NC_{max} : nombre de cycles, n : nombre de cellule dans la grille, m : nombre de fourmis), et le grand problème d'ACO réside dans son paramétrage qui rend cette approche pas pratique dans le faite que les paramétres dépendent fortement de l'environnement où évolue le robot.

Surmonter ces limitations représente un défi pour la future recherche. La conclusion générale décrit les travaux futurs possibles.

Chapitre 7

Planification de trajectoire par A star (A*)

7.1 Principe de l'algorithme

Au premier d'abord, on pourrait se dire que pour trouver un chemin d'un point à un autre il faut commencer par se diriger vers la destination. Et bien ... c'est justement cette idée qu'utilise l'algorithme A*. L'idée est très simple : à chaque itération (c'est un algorithme itératif), on va tenter de se rapprocher de la destination, on va donc privilégier les possibilités directement plus proches de la destination, en mettant de côté toutes les autres.

Toutes les possibilités ne permettant pas de se rapprocher de la destination sont mises de côté, mais pas supprimées. Elles sont simplement mises dans une liste de possibilités à explorer si jamais la solution explorée actuellement s'avère mauvaise. En effet, on ne peut pas savoir à l'avance si un chemin va aboutir ou sera le plus court. Il suffit que ce chemin amène à une impasse pour que cette solution devienne inexploitable.

L'algorithme va donc d'abord se diriger vers les chemins les plus directs. Et si ces chemins n'aboutissent pas ou bien s'avèrent mauvais par la suite, il examinera les solutions mises de côté. C'est ce retour en arrière pour examiner les solutions mises de côté qui nous garantit que l'algorithme nous trouvera toujours une solution (si tenté qu'elle existe, bien sûr).

On peut donc lui donner un terrain avec autant d'obstacles qu'on veut, aussi tordus soient-ils, s'il y a une solution, A* la trouvera.

7.2 Déroulement de l'algorithme

Son fonctionnement est plus en moins le même que l'algorithme de Dijkstra [1], à la différence qu'on ne cherche pas à choisir le nœud le plus «proche » du nœud de départ, mais celui qui semble être le plus dans la direction du nœud d'arrivée.

Dans une exploration en graphe, à savoir une application dans le domaine de la planification de trajectoire, il est relativement facile d'utiliser la distance par rapport au nœud d'arrivée comme heuristique. Une heuristique $h(n)$ est une estimée du cout minimal pour aller du nœud courant n jusqu' au nœud d'arrivée. Connaissant le cout $g(n)$ entre le nœud de départ et le nœud courant n, on possède donc une estimée du cout totale la solution $f(n)$. La stratégie d'exploration de l'algorithme A* va donc être de choisir le nœud suivant possédant le plus faible « cout total estimé » $f(n)$.

$$f(n) = g(n) + h(n) \tag{7.1}$$

où

$$h(n) = distance(n, n_{final}) \tag{7.2}$$

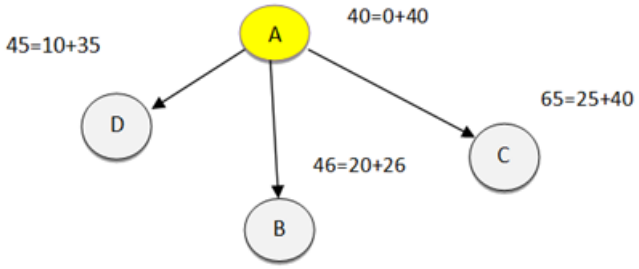
et

$$g(n) = \sum_{départ}^{n} cout(noeud_i, noeud_j) \tag{7.3}$$

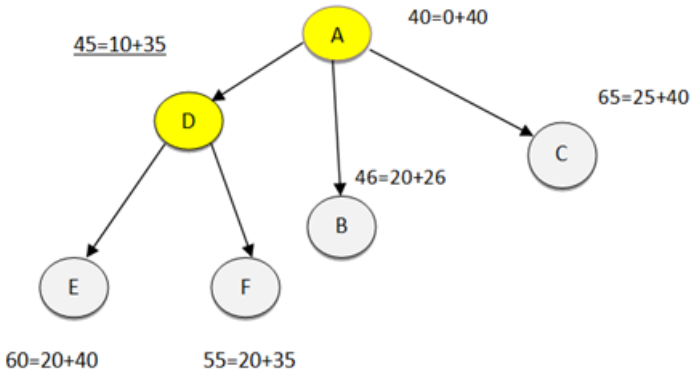
Le processus graphique ci-dessus décrivent le déroulement de l'exploration A* où les nœuds de départ et d'arrivé sont respectivement A, I.

1. **Djikstra problème :** Etant donné un ensemble de points et un ensemble de liaisons entre ces points caractérisé par un certain poids, comment trouver le plus court chemin, le plus agréable entre 02 points.

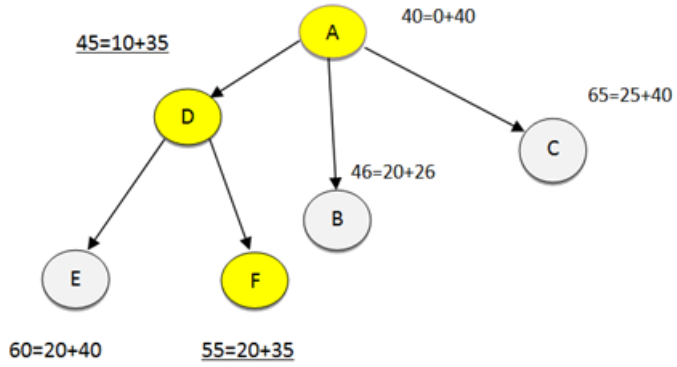
1. Développement du nœud A



2. Développement du nœud D



3. Développement du nœud F



4. Arrivée au nœud I

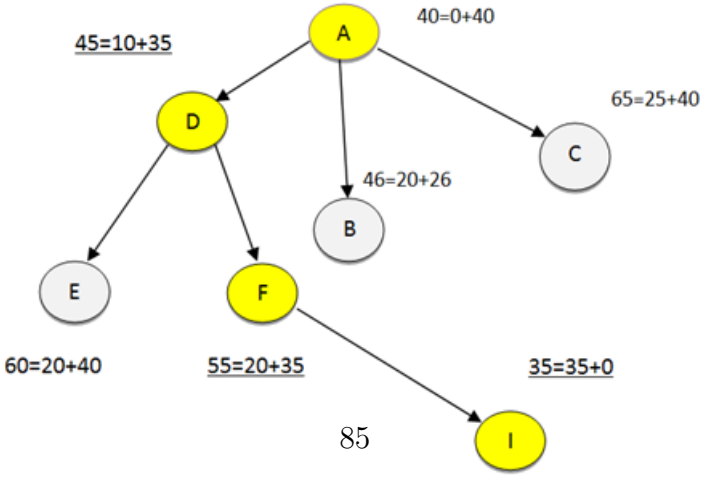


FIGURE 7.1: Fonctionnement de l'algorithme A*.

A chaque fois qu'on développe un nœud, c'est-à-dire qu'on regarde les nœuds suivants disponibles à ce nœud, on étend notre arbre de recherche. Cependant l'exploration en graphe doit prendre en compte la possibilité de répétition d'état, c'est pour cela qu'à chaque fois que l'on arrive à un état déjà visité, on ne garde que le chemin menant à ce nœud avec le chemin le moins couteux.

Nous avons indiqué plus haut que l'algorithme A* est optimal sous certaines conditions, nous allons les décrire dans ce paragraphe. Dans le cas de l'exploration en graphe, la condition pour que A* soit optimal est que l'heuristique doit être consistante (Russel, 1995). Une heuristique $h(n)$ est consistante si pour tout nœud n_1 et pour tout voisin n_2 de n_1, le coût estimé $h(n_1)$ du chemin entre n_1 et le nœud d'arrivé n'est pas supérieur au cout de l'étape entre n_1 et n_2 plus le coût estimé $h(n_2)$ du chemin entre n_2 et le nœud d'arrivé :

$$h(n_1) \leq coût(n_1, n_2) + h(n_2) \tag{7.4}$$

Cette condition peut être respectée si on prend par exemple la distance cartésienne entre le nœud n_1 et le nœud d'arrivé :

$$h(n) = dist(n, n_{final}) = \sqrt{(x - x_{final})^2 + (y - y_{final})^2} \tag{7.5}$$

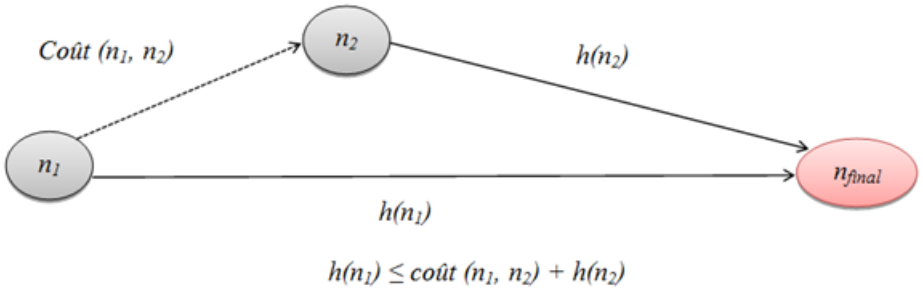


FIGURE 7.2: Consistance de l'exploration A*.

Dans ce cas, l'inégalité triangulaire impose que : $dist(n_1, n_{final}) \leq dist(n_1, n_2) + dist(n_2, n_{final})$, et vu que $coût(n_1, n_2) \geq dist(n_1, n_2)$ notre heuristique respecte bien la condition de consistance (figure 7.2). Il serait facile de vérifier cela expérimentalement, en effet, la condition de consistance impose que la suite de nœuds développés est dans l'ordre non décroissant des valeurs de $f(n)$ (comme on peut le voir dans la figure 7.1)

Il en découle aussi que la première fois que l'on voudra développer le nœud final, on aura atteint notre objectif avec le chemin optimal. Comme on peut l'observer dans la figure 7.3, supposons que le nœud I fut déjà parmi les voisins du nœud A. L'algorithme d'exploration ne se serait jamais arrêté à ce moment-là vu que l'on dispose encore de nœuds avec des meilleures estimées.[28]

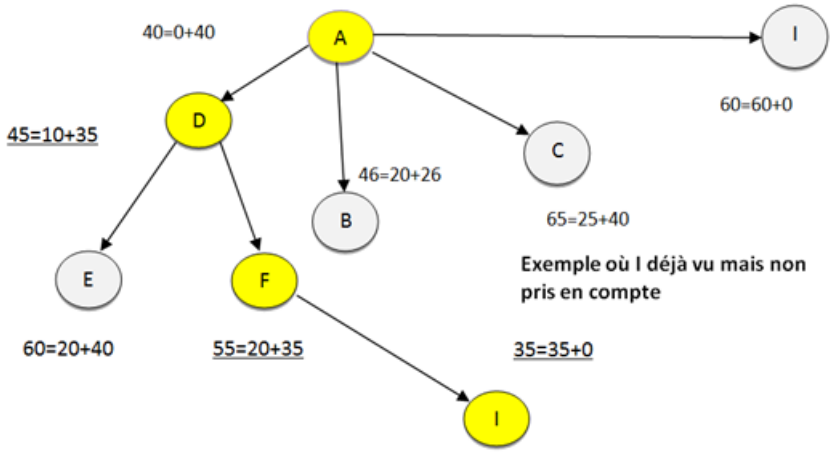


FIGURE 7.3: Illustration de l'optimalité de A*.

7.3 Description de l'algorithme

7.3.1 Les listes A*

A*utilise deux listes, ces listes contiennent des points. Pour être plus général, on peut même dire que ces listes contiennent des nœuds d'un graphe, lequel graphe représentant notre terrain.

La première liste, appelée liste ouverte, va contenir tous les nœuds étudiés. Dès que l'algorithme va se pencher sur un nœud du graphe, il passera dans la liste ouverte (sauf s'il y est déjà).

La seconde liste, appelée liste fermée, contiendra tous les nœuds qui, à un moment où à un autre, ont été considérés comme faisant partie du chemin solution. Avant de passer dans la liste fermée, un nœud doit d'abord passer dans la liste ouverte, en effet, il doit d'abord être étudié avant d'être jugé comme bon.

On ne sait pas grande chose du chemin solution si ce n'est que le point qui pourra nous rapprocher de la solution est un point voisin du point que nous étudions. On va

donc étudier chacun des nœuds voisins du nœud courant pour déterminer celui qui a le plus de chances de faire partie du chemin solution.

La recherche du chemin commence par le premier point, en étudiant tous ses voisins, en calculant leur qualité, et en choisissant le meilleur pour continuer. Chaque point étudié est mis dans la liste ouverte et le meilleur de cette liste passe dans la liste fermée, il va servir de base pour la recherche suivante.

Remarque : Pour déterminer le meilleur point pour continuer le chemin, il ne faut pas uniquement chercher celui qui a la meilleure qualité dans ses voisins, mais dans toute la liste ouverte. C'est comme ça qu'on pourra abandonner un chemin qui avait l'air bon au début et qui ne l'était pas.

Ainsi, à chaque itération on va regarder parmi tous les nœuds qui ont été étudiés (et qui n'ont pas encore été choisis) celui qui a la meilleure qualité. Et il est tout à fait possible que le meilleur ne soit pas un voisin direct du point courant.

Cela signifiera que le point courant nous éloigne de la solution et qu'il faut le corriger.L'algorithme s'arrête quand la destination a été atteinte ou bien lorsque toutes les solutions mises de côté ont été étudiées et qu'aucune ne s'est révélée bonne, c'est le cas où il n'y a pas de solution.

7.3.2 Détermination des nœuds voisins

Si on considère notre domaine de recherche comme une carte, ce sont tous les points adjacents (en haut, en bas, à gauche, à droite et les points en diagonale) sauf si ceux-ci contiennent des obstacles infranchissables (mur, montagne, rivière ...).

Donc, avant de se lancer dans l'étude de la qualité de chacun des nœuds adjacents, il ne faut prendre que ceux qui sont vraiment utilisables. Il faut aussi mettre de côté tous les nœuds déjà présents dans la liste ouverte ou dans la liste fermée. Et pour être plus précis, je dirais qu'il ne faut pas prendre un point s'il est déjà dans la liste ouverte, à moins qu'il ne soit meilleur, auquel cas on va mettre à jour la liste ouverte.

Ce qu'il faut regarder sur chacun des nœuds voisins potentiels :

- Est-ce que est un obstacle ? Si oui, on oublie ce nœud.
- Est-il dans la liste fermée ? Si oui, ce nœud a déjà été étudié ou bien est en cours d'étude, on ne fait rien.
- Est-il dans la liste ouverte ? si oui, on calcule la qualité de ce nœud, et si elle est meilleure que celle de son homologue (ayant le même emplacement dans la grille)

dans la liste ouverte, on modifie le f(n) et le parent du nœud présent dans la liste ouverte.
– S'il n'est pas dans la liste ouverte, on l'ajoute dans cette liste et on calcule sa qualité f(n).

7.3.3 Notion de parent d'un nœud

Chaque nœud à un parent, c'est par lui qu'on arrive là où on est. Le parent représente le meilleur chemin entre deux nœuds. Le parent est très important à la fin de l'algorithme, pour retrouver le chemin. Il joue aussi un rôle important lors de la mise à jour d'un nœud dans la liste ouverte. Nous avons vu qu'il fallait mettre à jour la liste ouverte dans le cas où un nœud avait une meilleure qualité que ce même nœud dans la liste ouverte.

Nous avons mis à jour la qualité du nœud dans la liste ouverte, mais il faut aussi mettre à jour son parent, pour bien signifier que cette qualité est possible par tel parent précis. Une fois que la destination a été atteinte, il faut retrouver le chemin en suivant à chaque fois les parents des nœuds présents dans la liste fermée. On remonte le fil jusqu'à arriver au point de départ.

7.3.4 Résumé des étapes

– On commence par le nœud de départ, c'est le nœud courant ;
– On regarde tous ses nœuds voisins ;
– Si un nœud voisin est un obstacle, on l'oublie ;
– Si un nœud voisin est déjà dans la liste fermée, on l'oublie ;
– Si un nœud voisin est déjà dans la liste ouverte, on met à jour la liste ouverte si le nœud dans la liste ouverte à une moins bonne qualité (et on n'oublie pas de mettre à jour son parent) ;
– Sinon, on ajoute le nœud voisin dans la liste ouverte avec comme parent le nœud courant ;
– On cherche le meilleur nœud de toute la liste ouverte. Si la liste ouverte est vide, il n'y a pas de solution, fin de l'algorithme ;
– On le met dans la liste fermée et on le retire de la liste ouverte ;
– On réitère avec ce nœud comme nœud courant jusqu'à ce que le nœud courant soit le nœud de destination.

Voici une illustration d'une itération de l'algorithme. On souhaite aller du point orange au point bleu. Les nœuds voisins de la case orange sont les cases marquées en vert, ils passent en liste ouverte. Et de chacune d'elles on calcule les couts G et H pour aller à

la case orange et pour aller à la destination. J'ai choisi la distance à vol d'oiseau. Et la case qui aura le cout le plus faible sera la case verte du dessous, elle va donc passer en liste fermée. L'algorithme va réitérer à partir de cette case.

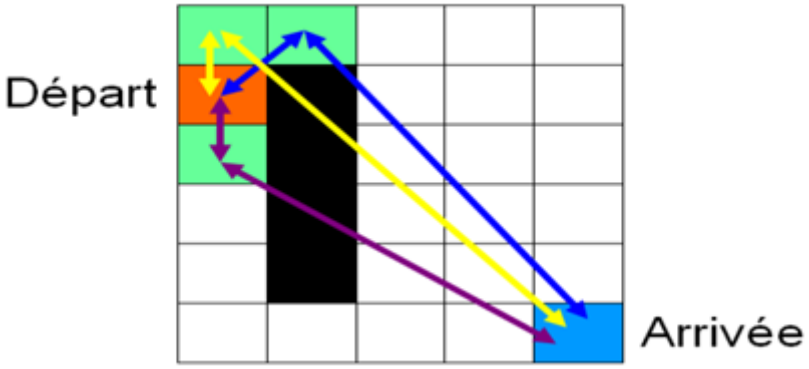


FIGURE 7.4: Illustration d'une itération de l'algorithme A* .

7.4 Propriétés de A*

- **Complétude :** À moins qu'il y est une infinité de nœuds .
- **Complexité de temps :** Exponentielle, Selon la longueur de la solution.
- **Complexité en espace :** Exponentielle (1- Selon la longueur de la solution. 2- Elle garde tous les nœuds en mémoire).
- **Optimale :** il demande énormément d'espace mémoire de travail que de temps de calcul.[17]

7.5 Résultats de simulation

Dans cette section, nous allons appliquer l'algorithme de planification A* pour un robot mobile dans des environnements différents, et nous allons présenter le temps d'exécution et la longueur de chemin pour chaque expérience, ces deux paramètres nous permettrons de comparer les performances de cet algorithme avec d'autre algorithme de planification, et pour assurer le lissage du chemin, nous allons appliquer l'algorithme d'interpolation cubique spline cité dans le chapitre précèdent, sur les chemins trouvés.

L'algorithme A* a la position du départ, la position du but, la position des obstacles comme des entrées et le chemin planifié defini par un ensemble de points comme sortie.

Remarque : toutes les simulations ont été faites avec MATLAB R2007b sur un ordinateur personnel de 4GO de RAM (DDR2) et un microprocesseur Intel duel core de fréquence 2.1Ghz.

– **Expérience 1 :** Environnement sans obstacles.

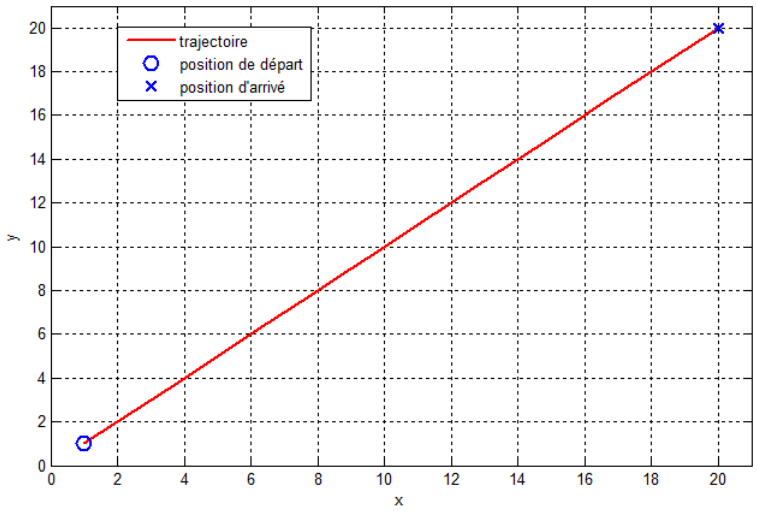


FIGURE 7.5: Trajectoire planifiée par A* dans un environnement sans obstacles.

- Temps d'exécution : 0.023 s
- Longueur de chemin : 26.87 um

– **Expérience 2 :** Environnement contient des obstacles de forme cercle de même rayon et disposés de façon homogène.

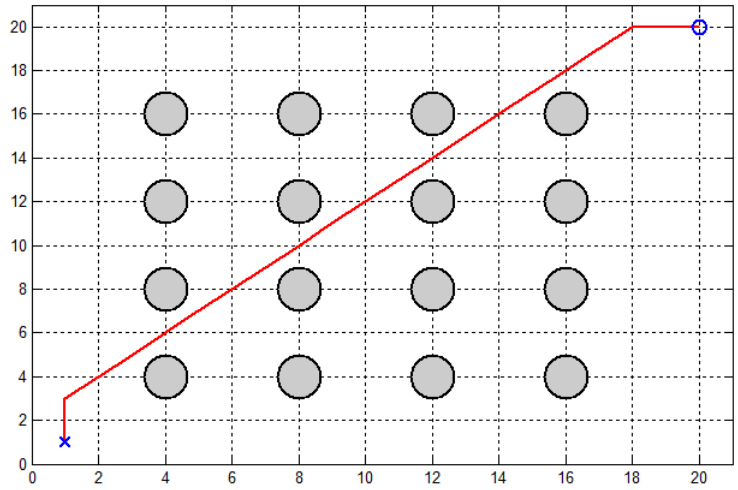


FIGURE 7.6: Trajectoire planifiée par A* dans un environnement contient des obstacles de forme cercles réguliers.

- Temps d'exécution : 0.075 s
- Longueur de chemin : 28.04 um

Après le lissage du chemin

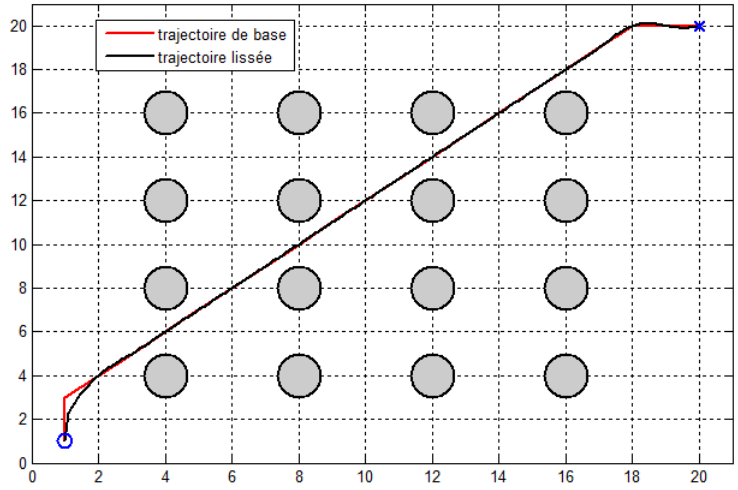


FIGURE 7.7: Trajectoire planifiée par A* et lissée par cubique spline dans un environnement contient des obstacles de forme cercles réguliers.

Remarque : Le lissage du chemin est fait par l'algorithme d'interpolation cubique spline cité dans le chapitre précèdent.

– **Expérience 3 :** Environnement contient des obstacles dispersés de forme cercle et de rayons différents.

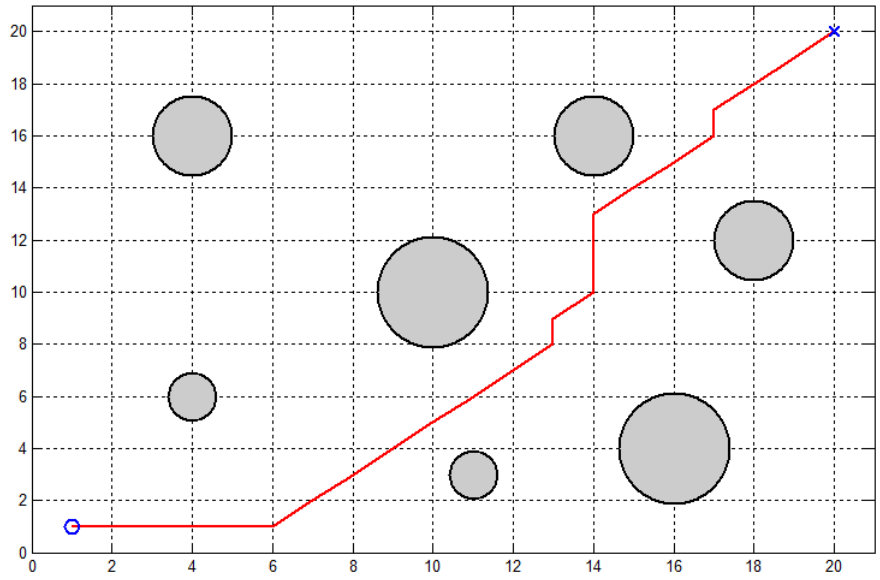


FIGURE 7.8: Trajectoire planifiée par A* dans un environnement contient des obstacles de forme cercle irréguliés.

- Temps d'exécution : 0.074 s
- Longueur de chemin : 29.80 um

Après le lissage du chemin

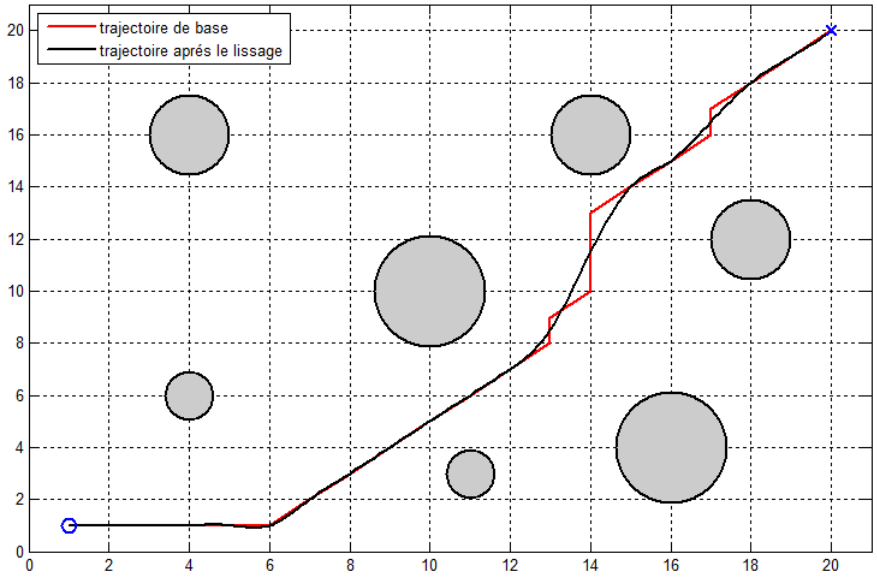


FIGURE 7.9: Trajectoire planifiée par A* et lissée par cubique spline dans un environnement contient des obstacles de forme cercles irréguliers.

– **Expérience 4 :** Environnement contient des murs comme obstacles.

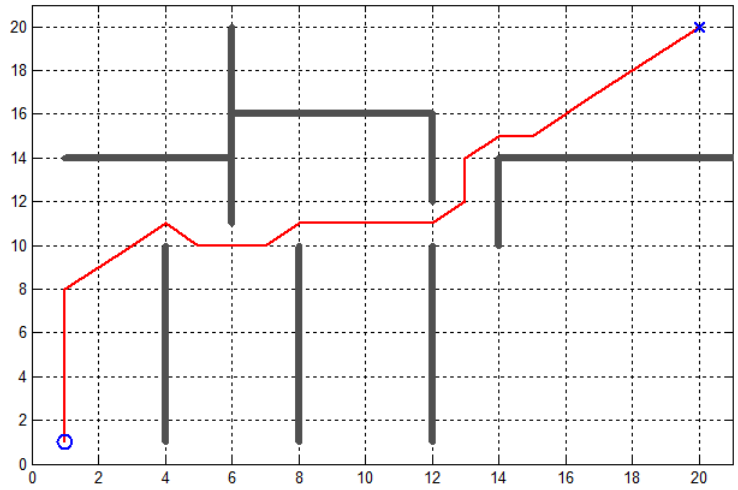


FIGURE 7.10: Trajectoire planifiée par A* dans environnement contient des murs comme obstacles.

- Temps d'exécution : 0.050s
- Longueur de chemin : 32.97 um

Après le lissage du chemin

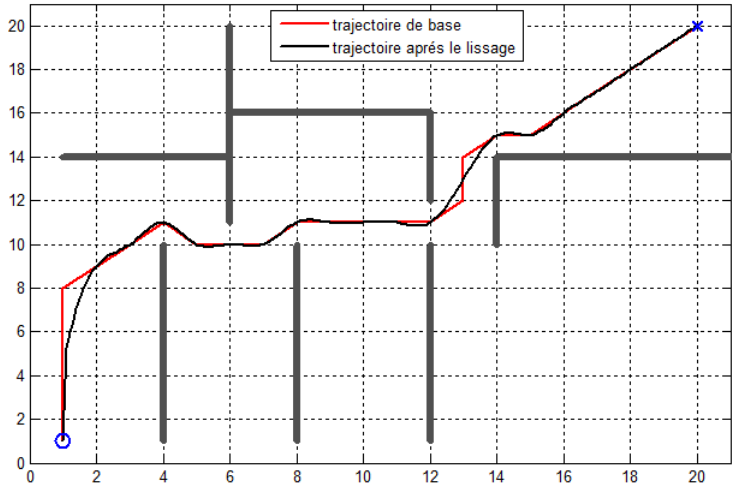


FIGURE 7.11: Trajectoire planifiée par A* et lissée par cubique spline dans un environnement contient des murs comme obstacles.

– **Expérience 5 :** Robot mobile dans un labyrinthe.

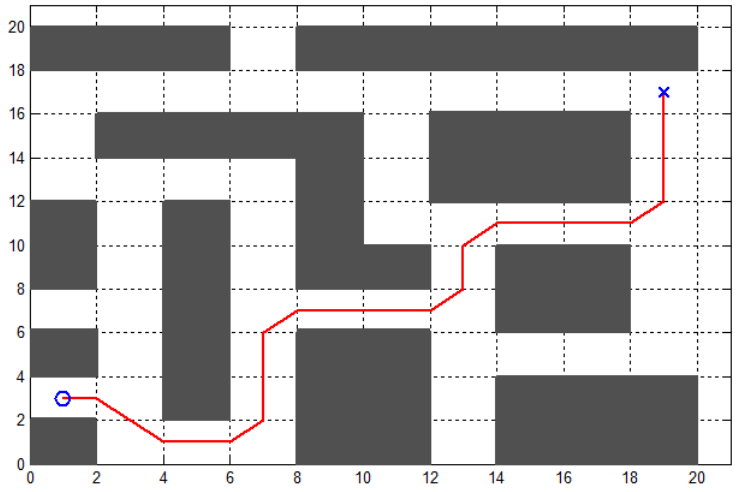


FIGURE 7.12: Trajectoire planifiée par A* dans un labyrinthe.

- Temps d'exécution : 0.024 s
- Longueur de chemin : 31.90 um

Après le lissage du chemin

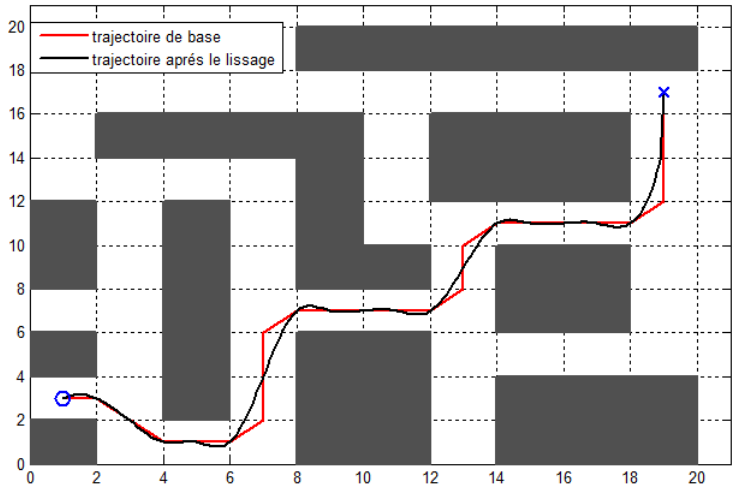


FIGURE 7.13: Trajectoire planifiée par A* et lissée par cubique spline dans un labyrinthe.

	Exp 1	**Exp 2**	**Exp 3**	**Exp 4**	**Exp 5**
temps d'exécution de A* (s)	0.023	0.075	0.074	0.050	0.024
longueur de la trajectoire planifiée (um)	26.87	28.04	29.80	32.97	31.90

TABLE 7.1: Temps d'exécution et longueur de trajectoire planifiée par A* pour chaque expérience.

7.6 Conclusion

D'après les simulations qu'on a fait, on peut conclure sur l'algorithme A* ce que suit :

- L'algorithme A* permet aux robots mobiles de planifier leurs trajectoires dans n'importe qu'elle environnement (2D), quelle que soit sa complexité (nombre d'obstacles et leurs formes).
- Le temps d'exécution minime, et l'optimalité des chemins crées par cet algorithme, ces deux performances sont reconnues pour être les efficaces dans ce type d'application, A* a été préféré comme solution pour la planification.

Pour améliorer plus les trajectoires crées par A*, on va essayer dans le chapitre suivant d'adapter l'algorithme PSO sur l'optimisation des trajectoires en se basant sur le critère la longueur de chemin.

Chapitre 8

Optimisation de trajectoire par PSO

8.1 Introduction

Dans ce chapitre on va appliquer le concept de l'algorithme d'optimisation par essaimes particulaires PSO pour mieux optimiser les trajectoires crées par ACO et A*.

8.2 Description de l'algorithme

On commence par créer un ensemble de trajectoires faisables (c.à.d. trajectoires qui relient le point de départ et le point d'arrivé et qui évitent la collision avec les obstacles), chaque trajectoire est créé à partir de la trajectoire de base (trajectoire obtenue par ACO ou A*) en modifiant aléatoirement les coordonnées de chaque point de cette trajectoire toute en restant au voisinage de ce point pour éviter la collision avec les obstacles.

$$\begin{cases} x_b^i - \varepsilon \leq \ x_g^i \ \leq x_b^i + \varepsilon \\ x_b^i - \varepsilon \leq \ x_g^i \ \leq x_b^i + \varepsilon \end{cases} \tag{8.1}$$

- x_b^i l'abscisse du point i de la trajectoire de base.
-x_g^i l'abscisse du point i de la trajectoire générée.
-y_b^i l'ordonnée du point i de la trajectoire de base.
- y_b^i l'ordonnée du point i de la trajectoire généré.
- ε un réel positive qui permet de contrôler le voisinage de changement des coordonnées (il doit être petit pour éviter la collision avec les obstacles).

Après avoir obtenu l'ensemble de trajectoires, on applique les principes de l'algorithme PSO pour trouver une trajectoire optimale (chaque particule représente une trajectoire composé par un ensemble de point). Le critère d'optimisation ou la fonction objectif est

la longueur du chemin.

Initialisation de l'algorithme

On initialise les particules par les trajectoires crées apartir de la trajectoire base crée par ACO ou A*, et on initialise aussiP^i_{best},P^g_{best}, la vitesse de chaque particule et comme ce dernier est une trajectoire (vecteur de dimension $n * 2$) donc la vitesse est aussi un vecteur de même dimension.

- P^i_{best} meilleure trajectoire trouvée par la particule i.
- P^g_{best} meilleure trajectoire trouvée par toutes les particules.

Pseudo code de l'algorithme

Initialisation des positions et des vitesses de chaque particule.
Pour $itération = 1, 2.....N$
Pour $particule = 1, 2....P$

Déplacement de la particule suivant l'équation du mouvement (4.3).

Pour $k = 1$: dimension de la particule

Si (le point k cas sort de l'espace de recherche) **alors**
Remplacer le point k par le point le plus proche situé sur le limite.
Fin de si

Fin de pour

Fin de pour
Mise à jour de : P^i_{best}, P^g_{best}.
Fin de pour
Remplacez la trajectoire de base par P^g_{best} trouvée.

8.3 Résultats de simulation

Pour toutes les expériences on a près les paramètres suivants $c_{max} = 1.47, c_1 = 0.7, \varepsilon = 0.4, N = 10, P = 20$.

Le choix de ces valeurs n'est pas basé sur une méthode systématique, mais par l'expérimentation. les valeurs du couple $(c_1, cmax)$ sont préconisées par les auteurs des travaux sur OEP.

8.3.1 Optimisation d'ACO par PSO

Pour vérifier l'efficacité de cet algorithme, on va mettre dans le robot mobile dans plusieurs situations et on compare la longueur de la nouvelle trajectoire crée par PSO avec celle de base crée par ACO. Afin de montrer la convergence de l'algorithme on va tracer la longueur du chemin en fonction du nombre d'itérations.

– **Expérience 1 :** Environnement contient des obstacles de forme cercle de même rayon, distribués régulièrement.

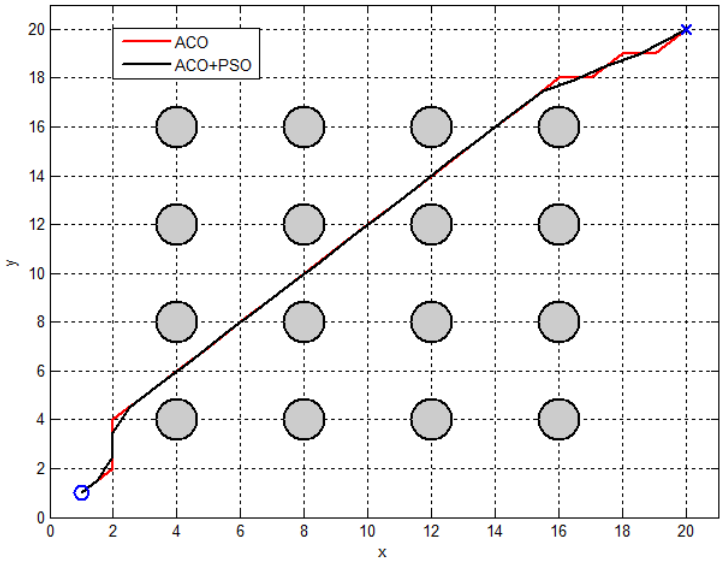


FIGURE 8.1: Trajectoire planifiée par ACO et optimisée par PSO dans un environnement contient des obstacles de type cercles réguliers.

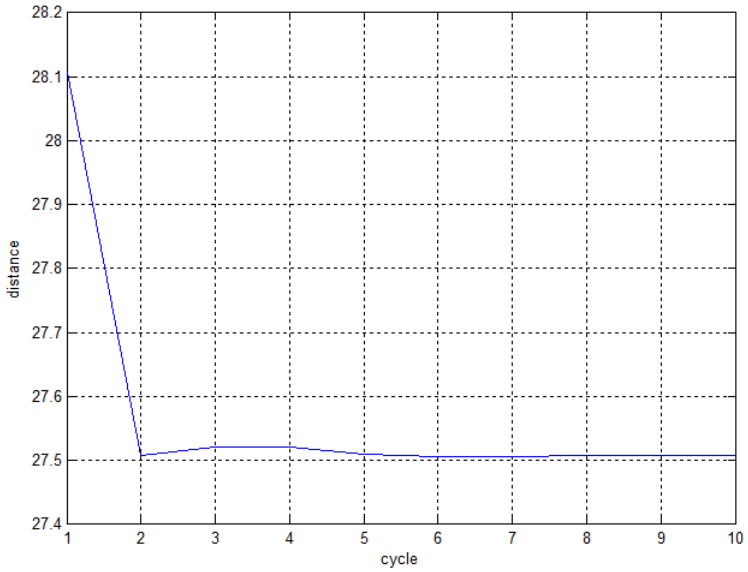


FIGURE 8.2: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 1).

- Longueur de chemin (ACO) : 28.10 um
- Longueur de chemin (ACO+PSO) : 27.51um

– **Expérience 2 :** Environnement contient des obstacles de forme cercle de différents rayons.

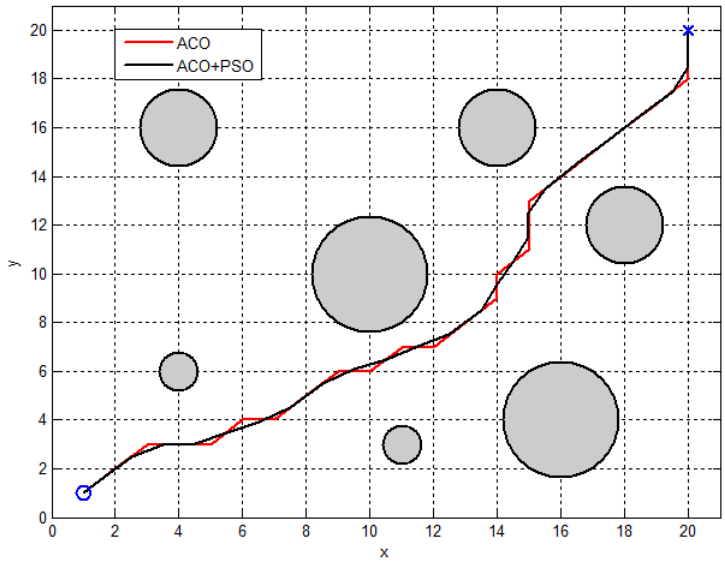


FIGURE 8.3: Trajectoire planifiée par ACO et optimisée par PSO dans un environnement contient des obstacles de forme cercles irréguliers.

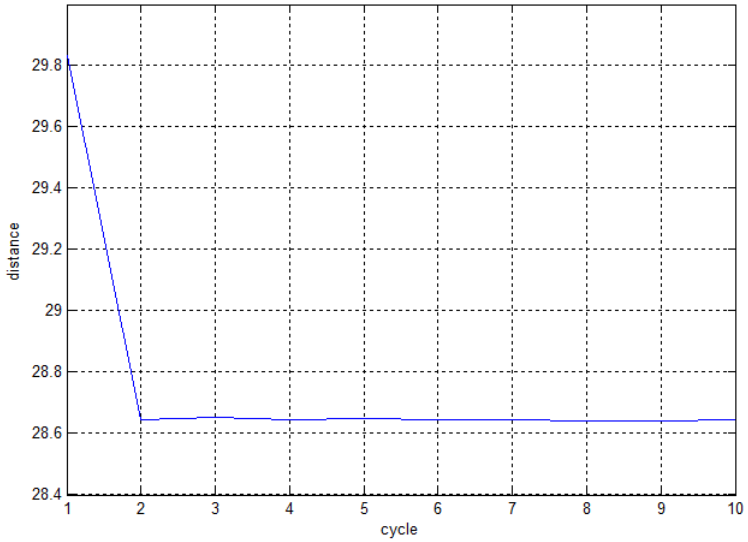


FIGURE 8.4: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 2).

- Longueur de chemin (ACO) : 29.80 um

- Longueur de chemin (ACO+PSO) : 28.64 um

– **Expérience 3 :** Environnement contient des murs comme obstacles.

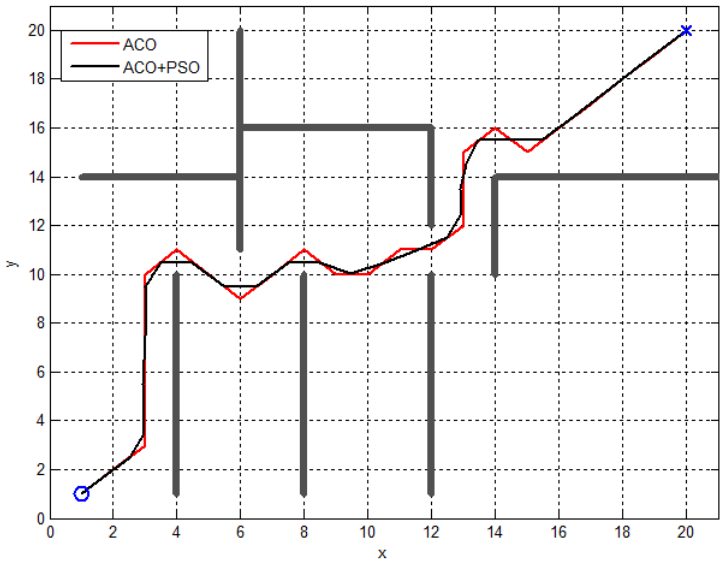


FIGURE 8.5: Trajectoire planifiée par ACO et optimisée par PSO dans un environnement contient des murs comme obstacles.

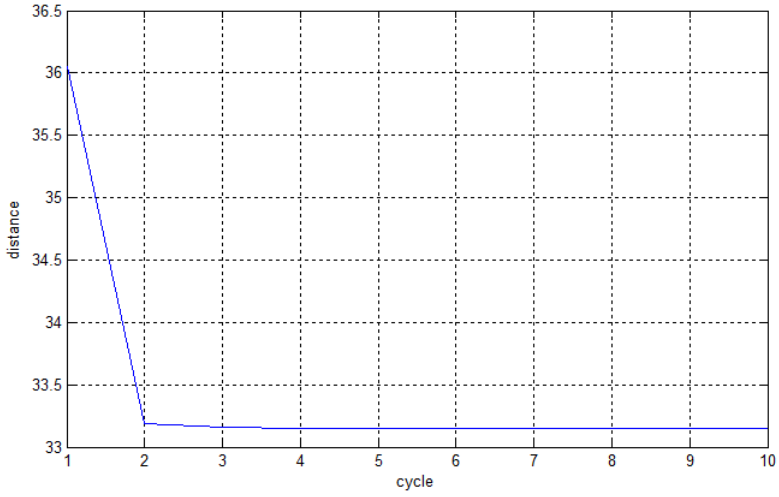


FIGURE 8.6: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 3).

- Longueur de chemin (ACO) : 36.04 um
- Longueur de chemin (ACO+PSO) : 33.15 um

– **Expérience 4 :** Robot mobile dans un labyrinthe.

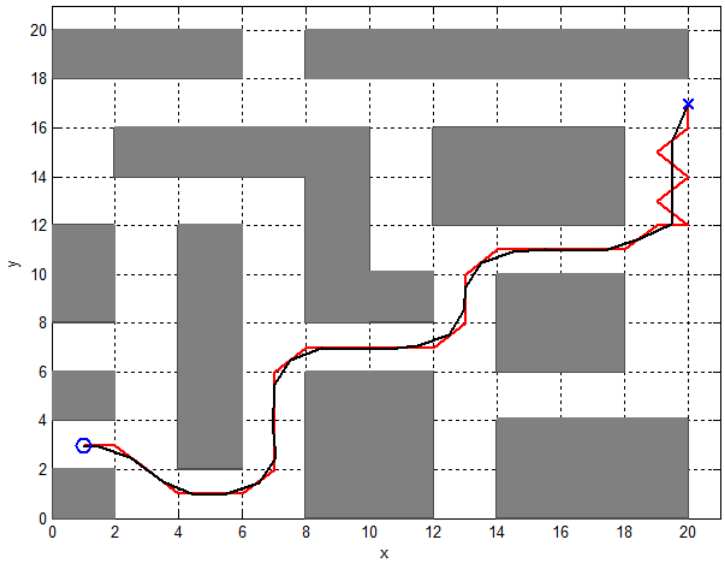


FIGURE 8.7: Trajectoire planifiée par ACO et optimisée par PSO dans un labyrinthe.

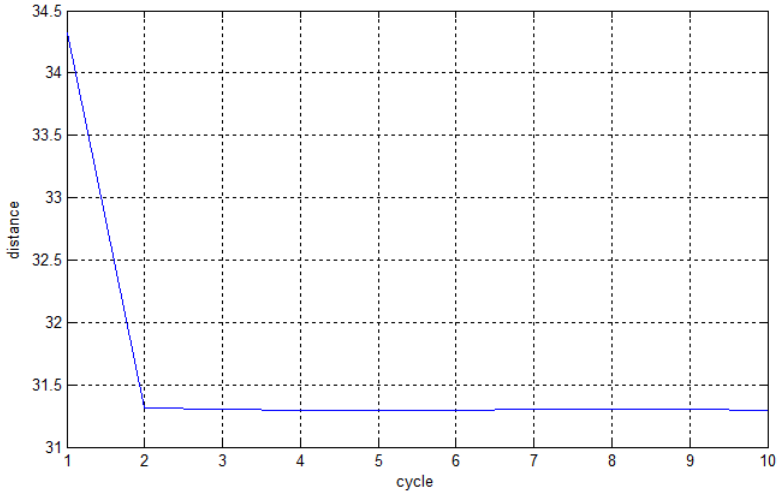


FIGURE 8.8: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 4).

- Longueur de chemin (ACO) : 34.56 um
- Longueur de chemin (ACO+PSO) : 31.30 um

– **Expérience 5 :** Environnement contient des obstacles disposés de façon aléatoire.

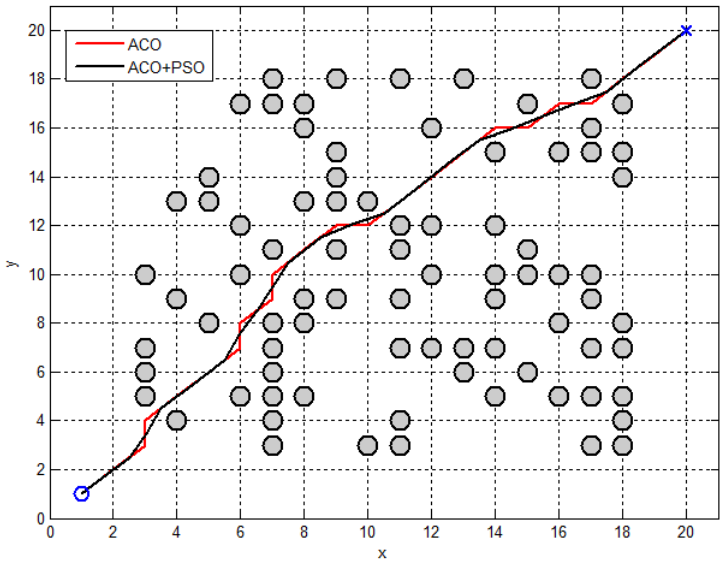


FIGURE 8.9: Trajectoire planifiée par ACO et optimisée par PSO dans un environnement contient des obstacles disposés de façon aléatoire.

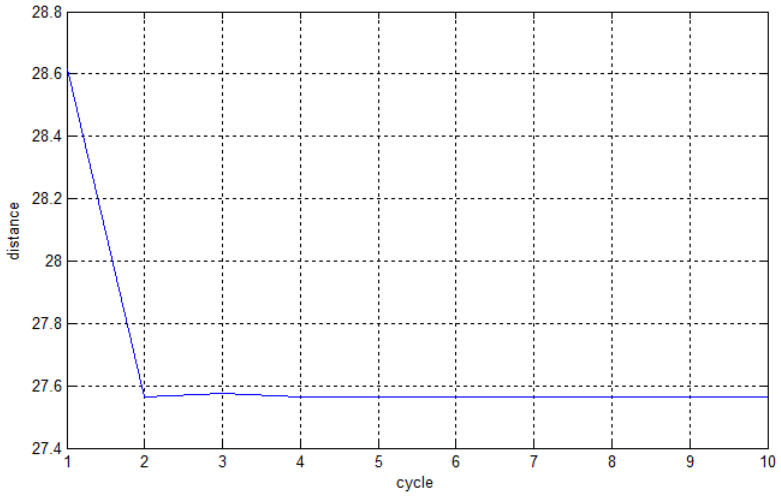


FIGURE 8.10: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 5).

- Longueur de chemin (ACO) : 28.63 um
- Longueur de chemin (ACO+PSO) : 27.56 um

	Exp 1	**Exp 2**	**Exp 3**	**Exp 4**	**Exp 5**
longueur trajectoire ACO (um)	28.10	29.80	36.04	34.56	28.63
longueur trajectoire ACO+PSO (um)	27.51	28.64	33.15	31.30	27.56

TABLE 8.1: Longueurs des trajectoires planifiées par ACO et celles planifiées par ACO et optimisée par PSO.

8.3.2 Optimisation de A* par PSO

– **Expérience 1 :** Environnement contient des obstacles de forme cercle de même rayon, disposés régulièrement.

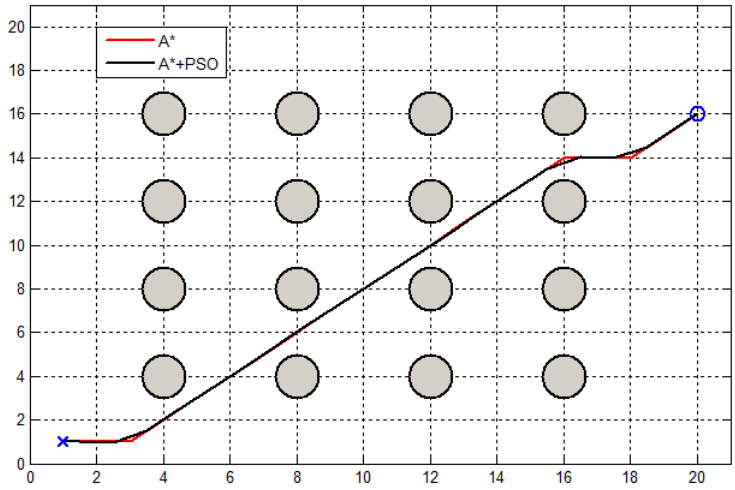


FIGURE 8.11: Trajectoire planifiée par A* et optimisée par PSO dans un environnement contient des obstacles de forme cercles réguliers.

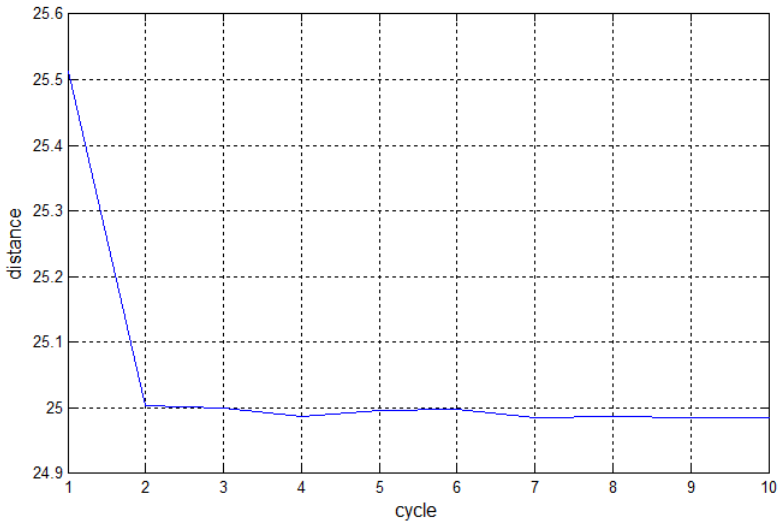


FIGURE 8.12: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 1).

- Longueur de chemin (A*) : 25.21 um
- Longueur de chemin (A*+PSO) : 24.99 um

– **Expérience 2 :** Environnement contient des obstacles de forme cercle de différents rayons.

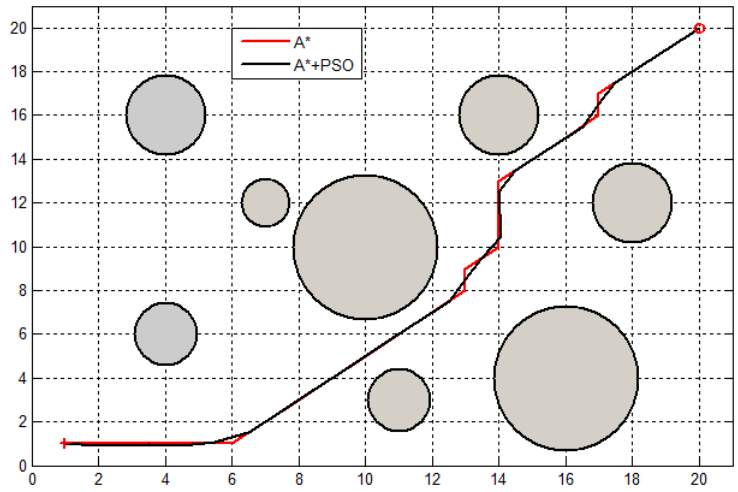


FIGURE 8.13: Trajectoire planifiée par A* et optimisée par PSO dans un environnement contient des obstacles de forme cercles irréguliers.

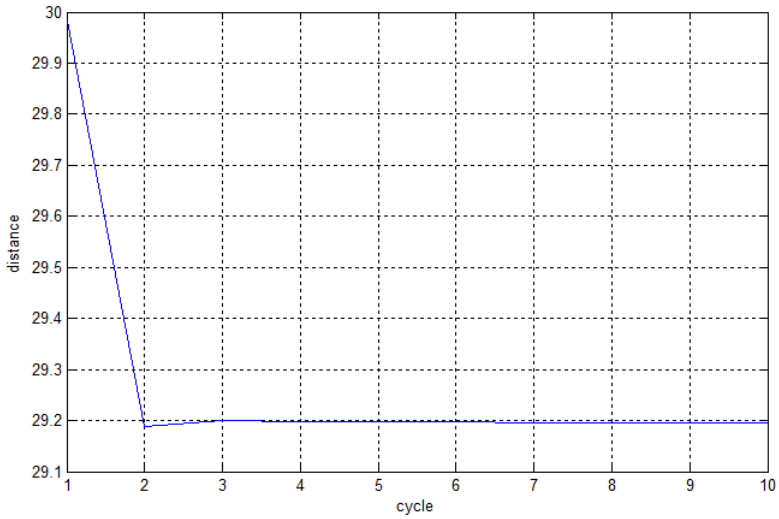


FIGURE 8.14: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 2).

- Longueur de chemin (A*) : 29.80 um

- Longueur de chemin (A*+PSO) : 29.20 um

– **Expérience 3 :** Environnement contient des murs comme obstacles.

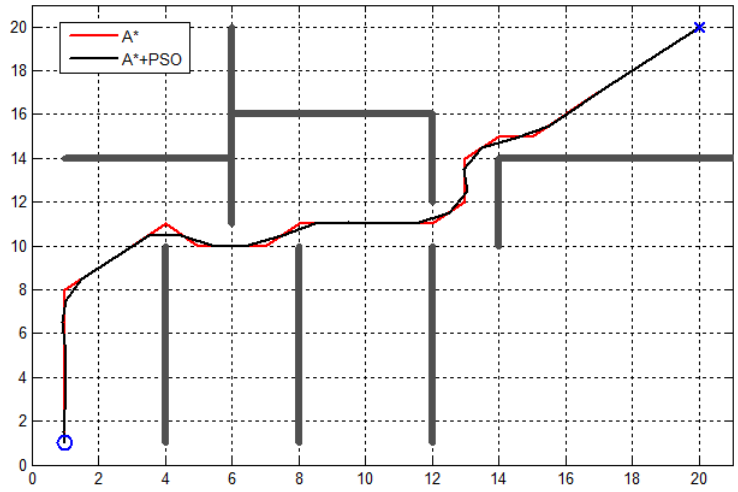


FIGURE 8.15: Trajectoire planifiée ACO et optimisée par PSO dans un environnement contient des murs comme obstacles.

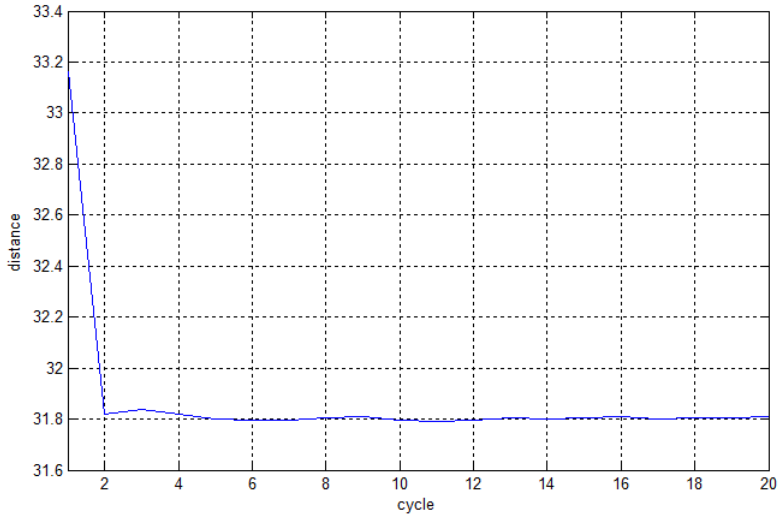


FIGURE 8.16: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 3).

- Longueur de chemin (A*) : 32.97 um
- Longueur de chemin (A*+PSO) : 31.81 um

– **Expérience 4 :** Robot mobile dans un labyrinthe.

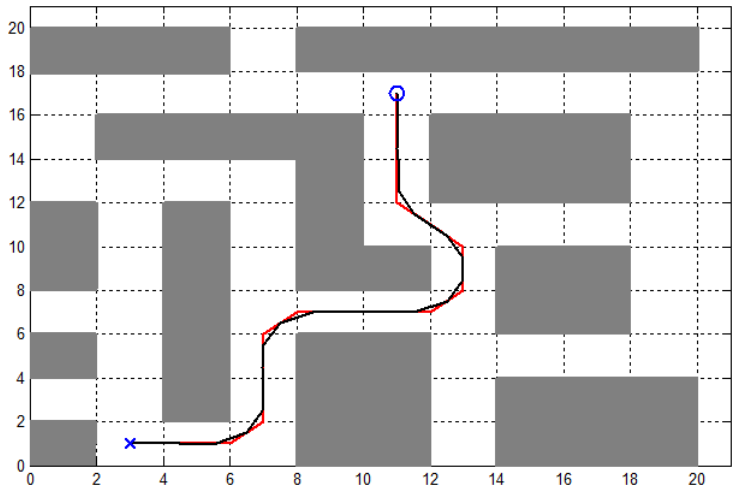


FIGURE 8.17: Trajectoire planifiée par A* et optimisée par PSO dans un labyrinthe.

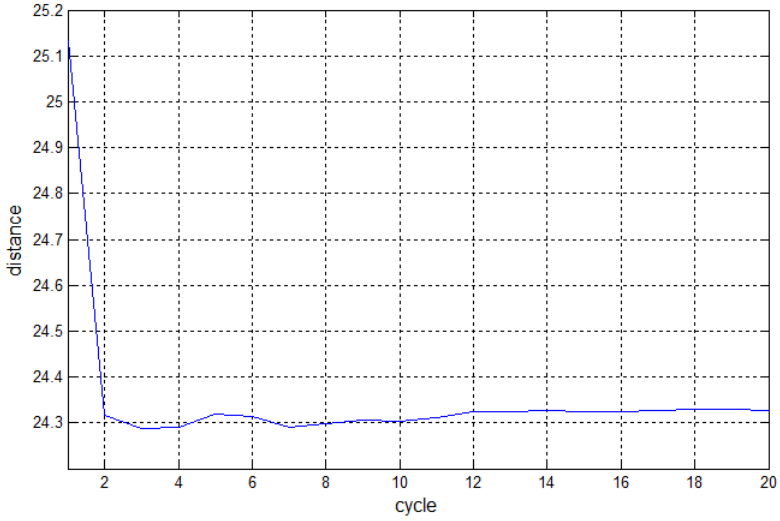


FIGURE 8.18: Distance de la meilleur trajectoire touvée par PSO pour chaque cycle (expérience 4).

- Longueur de chemin (A*) : 25.07 um
- Longueur de chemin (A*+PSO) : 24.33 um

	Exp 1	**Exp 2**	**Exp 3**	**Exp 5**
longueur trajectoire A* (um)	25.21	29.80	32.97	25.07
longueur trajectoire A*+PSO (um)	24.99	29.20	31.81	24.33

TABLE 8.2: Longueurs des trajectoires planifiées par A* et celles planifiées par A* et optimisée par PSO.

8.4 Conclusion

Dans ce chapitre nous avons proposé un algorithme d'optimisation de longueur de chemin par essaim particulaire (PSO), il prend en entrée une trajectoire de base reliant le point de départ et le point de destination évitable aux obstacles.

Les résultats de simulation montrent son efficacité, il n'assure pas que la minimisation de la longueur mais aussi le lissage du chemin, et on peut constater dans les figures précédentes que la convergence est atteinte dès la deuxième itération qui est un avantage très important dans un domaine où le temps est un paramètre très précieux.

Conclusion générale et perspective

Dans ce travail, nous avons étudié la question de recherche suivante : «en exploitant les algorithmes évolutionnaires, comment un robot peut-il, de façon autonome, générer un chemin sécuritaire et optimal entre une position initiale et une destination connue dans une représentation 2D de son environnement ? ».

Nous avons tout d'abord exposé brièvement les trois méthodes existantes de la planification de trajectoire : la roadmaps, le champs de potentiel artificiel et la décomposition cellulaire. Dans notre travail, nous avons adopté cette dernière où d'une part l'espace de configuration du robot est décomposé en un nombre fini de cellules puis d'autre part un algorithme recherche, dans l'ensemble des cellules (le maillage), un chemin qui répond à certains critères d'optimalité.

Par la suite, nous avons détaillé l'utilisation de l'algorithme ACO pour la planification de la trajectoire dans un maillage. Nous avons testé cet algorithme pour un robot mobile se trouvant dans différents environnements 2D encombrés d'obstacles. Les résultats obtenus ont montré que cet algorithme a permis de planifier un chemin proche de l'optimal dans ces différents environnements. Cependant, ces trajectoires présentent des rugosités qui sont inadmissibles pour un robot réel tenant compte de sa cinématique.

Pour éliminer ces rugosités (imperfections de la trajectoire), nous avons proposé en premier lieu un algorithme amélioré qui élimine la plupart de ce type d'irrégularités mais qui s'est avéré insuffisant dans certains cas. Aussi, nous avons exploité les techniques de lissage de courbes par spline cubique pour lisser complètement les trajectoires obtenues par l'algorithme ACO.

En dernier lieu, nous sommes parvenus à adapter l'algorithme d'optimisation par essaim particulaire (PSO) pour résoudre la question de l'amélioration des performances relatives aux trajectoires générées par d'autres algorithmes. Ces performances s'expriment en termes de la réduction de la longueur du chemin et de sa rugosité. Les résultats de simulation et la convergence rapide de l'algorithme ont montré son efficacité.

La planification par ACO, développée récemment, peut être une solution prometteuse

au problème de navigation des robots mobiles. Cependant et relativement aux autres algorithmes de planification, son problème réside dans le paramétrage de l'algorithme et le temps d'exécution. Afin de surmonter ces limitations, nous proposons en perspectives d'explorer éventuellement quelques solutions.

Les travaux futurs qui pourraient être poursuivis dans le cadre de ce projet de fin d'études, sont entre autres :

- Hybridation de l'algorithme ACO avec d'autres algorithmes tel que : les algorithmes génétiques ou PSO, pour remédier au problème du paramétrage, l'exécution de l'algorithme peut être considérablement augmentée mais la fiabilité du choix des paramètres peut être garantie.

- Pour résoudre le problème de planification, d'essayer avec les nouveaux algorithmes évolutionnaires tels que : les colonies de bactérie, le système immunitaire, artificial fish swarm algorithm (AFSA).....

- Prendre en compte lors de la planification les dimensions réels du robot et ces contraintes mécaniques (la non holonomie, etc.).

- Développer ces algorithmes pour qu'ils retournent non seulement un chemin mais aussi un corridor de sécurité. ce corridor sert à quantifier dans quelle mesure le robot peut dériver de son chemin sans compromettre sa sécurité.

- Traiter le problème de perception, car l'obtention d'un maillage régulier de l'environnement n'est pas encore une chose aisée malgré le développement des techniques de perception.

Bibliographie

[1] Alejandro AGUDELO, "Planification de chemin pour un robot mobile dans un environnement partiellement connu ", Mémoire pour l'obtention du diplôme de maitrise des sciences appliquées (Automatique), Ecole polytechnique de Montréal, Décembre 2007.

[2] Joe SFEIR, "Navigation d'un robot mobile en environnement inconnu utilisant les champs de potentiels artificiels", Mémoire pour l'obtention du diplôme de la maitrise en génie de la production automatisée, Ecole de technologie supérieure (Université de Québec), Novembre 2009.

[3] Khaled BELGHITH, " Une nouvelle approche flexible de planification de trajectoire de robot dans un environnement complexe", Mémoire pour l'obtention de la maîtrise en informatique (option génie logiciel), Université du Québec à Montréal, Février 2005.

[4] David GINGRAS, " Planification de chemins pour robot mobile explorateur de planète", Mémoire de maitrise (Spécialité : génie électrique), Université de Sherbrooke (Québec), Août 2010.

[5] Kyung min Han, " Collision free path planning algorithms for robot navigation problem ", Master science thesis, University of Missouri-Columbia, August 2007.

[6] R. ATTIA, M. ADJADJI, "Différentes techniques d'optimisation de la logique floue, Application sur la colonne d'absorption»,Projet de fin d'étude, Ecole Nationale Polytechnique d'Alger, juin 2009.

[7] A. AYOUAZ, B. HAMMOUDA, " Application de l'algorithme de fourmis artificielles et l'algorithme des essaims particulaires et trois critères d'interprétation DUVAL, CEI et ROGERS pour le Diagnostic de l'huile de transformateurs", Projet de fin d'étude, Ecole Nationale Polytechnique d'Alger, juin 2011.

[8] A.LAZINICA,"Particle swarm optimization», Edition In-Tech, January 2009.

[9] W.TFAILI , "Conception d'un algorithme de colonie de fourmis pour l'optimisation continue dynamique", Thèse de Doctorat, Université Paris 12 VAL DE MARNE.

[10] M. CLERC, "Optimisation par essaim particulaire», Tutoriel de OEP, France télécom R&D, 2003.

[11] Steven M. Lavalle, "Planning Algorithm", Cambridge university press, 2006.

[12] J. Dreo, " Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical", Thèse de Doctorat, Université de Paris 12 Val de Marne, Décembre 2003.

[13] F.Guillard, "Approche cognitive pour la planification de trajectoire sous contraintes", These de Doctorat, Février 2012.

[14] Mitul Ashwin Adhiya, " Navigation of robot mobile using occupancy grids", Master of Applied Science, Ottawa-Carleton Institute for Mechanical and Aerospace Engineering, April 2007.

[15] Mitul Ashwin Adhiya, " Graph-based Path Planning for Mobile Robots", Thesis of Doctoate, School of Electrical and computer Engineering, Georgia Institute of technologie, December 2006.

[16] Paul I. Muntean, "Mobile robot navigation on partially known map using the A* algorithm ", AQTR 2010 THETA 17th edition May 28-30 2010 Cluj-Napoca Romania, Proceedings of 2010 IEEE International Conference on Automation, Quality and Testing, Robotics.

[17] Safia Kedad-Sidhoum, "Plus courts chemins : Algorithme A* ", cours d'algorithmique avancée, Laboratoire d'Informatique de Paris 6 (LIP6), 2005-2006.

[18] Song-Hiang Chia, Kuo-Lan Su, Jr-Hung Guo, Cheng-Yun Chung " Ant Colony System Based Mobile Robot Path Planning", 2010 Fourth International Conference on Genetic and Evolutionary Computing.

[19] Rong Du, Xiaobin Zhangy, Cailian Chen, Xinping Guan," Path Planning-With Obstacle Avoidance In PEGs : Ant Colony Optimization Method ", School of Electronic, Information and Electrical Engineering , Shanghai Jiao Tong University.

[20] TAN Guan-Zheng, HE Huan, SLOMAN Aaron," Ant Colony System Algorithm for Real-Time Globally Optimal Path Planning of Mobile Robots ", Acta Automatica Sinica, Science Direct, March 2007.

[21] WANG Hong-jian, XIONG Wei," Research on global path planning based on ant colony optimization for AUV", College of Automation, Harbin Engineering University, Harbin 150001, China, 2009.

[22] COSTANZO Andrea, LUONG Thé Van, MARILL Guillaume," Optimisation par colonies de fourmis", 19 mai 2006.

[23] Guanjun Ma, Haibin Duan, Senqi Liu," Improved Ant Colony Algorithm for Global Optimal Trajectory Planning of UAV under Complex Environment", International Journal of Computer Science & Applications, 2007.

[24] Michael Brand, Michael Masuda, Nicole Wehner, Xiao-Hua Yu" Ant Colony Optimization Algorithm for Robot Path Planning ", 2010 International Conference On Computer Design And Appliations (ICCDA 2010).

[25] Qiang Zhao and Shaoze Yan," Collision-Free Path Planning for Mobile Robots Using Chaotic Particle Swarm Optimization", Department of Precision Instruments and Mechanology, Tsinghua University, Beijing 100084, P.R. China.

[26] Z.Xiaoguang, W. Zhangqi, H.Qingyao," Mobile Robot Path Planning Based on Multi-parameters Optimization Ant Colony Algorithm", Journal of Convergence Information Technology, Volume6, Number 8, August 2011.

[27] I.Alaya, "Optimisation multi-objectif par colonies de fourmis Cas des problémes de sac à dos", thése de doctorat, université de la MANOUBA, Ecole Nationale des sciences de l'informatique, May 2009.

[28] D.Garcia, "Planification de trajectoire dans un atlas de cartes", mémoire pour l'obtetion de maitraise des sciences appliquées (automatique et systémes), Ecole Polytechnique de Montréal, Mars 2008.

ملخص

ان مشكل تحديد المسار يعتبر نقطة حرجة و مهمة في ميدان الروبوتات ذاتية التحكم.

في هذه الاطروحة قدمنا خوارزمية لتحديد المسارتعتمد على مفهوم تحقيق الحل الامثل باستخدام مستعمرة النمل و قمنا بتمليس هذا المسارعن طريق الاستفياء بكثيرات الحدود، بعد ذلك قدمنا الخوارزمية التى تعتبرالاكثراستخداما في الميدان التطبيقي Astar (A*) **وذلك لتقديرفعالية الخوارزمية الاولى، بعد ذلك اقترحنا خوارزمية لتحسين المسارالمحدد مسبقا، هذه الخوارزمية تستعمل مفهوم تحقيق الحل الامثل باستخدام الاسراب الجزيئية.**

نتائج المحاكاة ب MATLAB **تظهرقدرة هذه الخوارزميات على ايجاد حلول للمشكل المذكوراعلاه.**

الكلمات المفتاحية: تحقيق الحل الامثل باستخدام مستعمرة النمل، تحديد المسا، تحقيق الحل الامثل باستخدام الاسراب الجزيئية.

Résumé :

Le problème de planification de trajectoire est un point critique et important dans le domaine de la robotique autonome.

Dans ce projet de fin d'étude nous avons présenté un algorithme de planification basé sur le concept d'optimisation par colonie de fourmis (OCF) et nous avons lissé cette trajectoire par interpolation polynomiale, puis nous avons présenté l'algorithme Astar (A*) qu'est très utilisé en pratique afin d'évaluer les performances du premier algorithme, par la suite nous avons proposé un algorithme pour optimiser la trajectoire déjà planifiée, cet algorithme utilise le concept d'optimisation par essaims particulaires (OEP).

Les résultats de simulation sur MATLAB montrent la capacité de ces algorithmes de résoudre le problème posé.

Mots clés : Optimisation par Colonie de Fourmis, Optimisation par Essaims Particulaires, planification de trajectoire.

Abstract :

The problem of path planning is a critical and important point in the field of autonomous robotics.

In this thesis we have presented an algorithm of planning based on the concept of ants colony optimization algorithm (ACO) and we have smoothed this trajectory by polynomial interpolation, then we have presented the algorithm Astar (A*) which is very much used in practice in order to evaluate the performances of the first algorithm, thereafter we have proposed an algorithm to optimize the already planned trajectory, this algorithm uses the concept of particles swarms optimization (PSO).

The results of simulation on MATLAB show the capacity of these algorithms to solve the problem mentioned above.

Key Words : Ant Colony optimization, Particles swarms optimization, path planning.