REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Abbès LAGHROUR - Khenchela
Faculté des Sciences et de la technologie
Département de Génie Industriel

Mémoire de fin d'études

Pour l'Obtention de Diplôme Master

Filière : Automatique

Spécialité : Automatique et informatique industrielle

Réalisé par :

Amamari bahaeddin et Sabeg hichem

Intitulé :

Planification de trajectoires pour un robot mobile via un algorithme d'Optimisation par Colonies de Fourmis

---

Dirigée par : Dr. ALLOUANI Fouad

Membres du Jury :

Bououden Sofiane	Prof. à l'université de Khenchela	Président
……	…. à l'université de Khenchela	Examinateur
Allouani Fouad	MCA à l'université de Khenchela	Rapporteur

Année universitaire 2021/2022

ملخص

في أطروحة الماستر هذه، نقدم استخدام خوارزمية ACO في شكلها البدائي، في حل مشكلة تخطيط مسار روبوت ذكي متنقل .الطريقة المتبعة، تسمح للروبوت المتحرك بتجنب مجموعة من العوائق واتباع مسارات مثلى موجودة في بيئته .تستند المسارات التي يتم إنشاؤها على استخدام القاعدتين الرئيسيتين اللتين تميزان خوارزمية ACO، (1)وظيفة الانتقال (أو قاعدة العشوائية) و (2) قاعدة تحديث مسارات الفرمون .تظهر نتائج المحاكاة التي تم الحصول عليها من تطبيق هذه الطريقة قدرتها على قيادة الروبوت لمتابعة مسارات ذات جودة ، وتجنب العوائق الموزعة عشوائياً في بيئته ، والمحددة بأحجام ومواضع مختلفة.

Résumé :

Dans ce mémoire de Master, nous nous présentons l'utilisation d'un algorithme ACO sous sa forme primitive, dans la résolution du problème de planification d'une trajectoire d'un robot mobile intelligent.

La méthode exposée, à permettre au robot mobile d'éviter un ensemble d'obstacles et de suivre des chemins optimaux dans son environnement. Les trajectoires générées, sont basées sur l'utilisation des deux règles principales caractérisant l'algorithme ACO, (i) la fonction (ou la règle) de transition et (ii) la règle de mise-à-jour des pistes de phéromones.

Les résultats de la simulation obtenus de l'application de cette méthode, montrent sa capacité, de conduire le robot de suivre des trajectoires de qualité, en évitant des obstacles distribués aléatoirement dans son environnement, définies avec des tailles et des positions différentes.

Abstract:

In this Master's thesis, we present the use of an ACO algorithm under its primitive form, in solving the problem of an intelligent mobile robot path planning. The used method, allow to the mobile robot to avoid a set of obstacles and to follow a set of optimal paths existed in its environment. The trajectories generated are based on the use of the two main rules characterizing the ACO algorithm, (i) the transition function (or rule) and (ii) the pheromone trail update rule. All obtained simulation results from the application of this method show its ability to drive the robot to follow trajectories with a good quality, avoiding obstacles randomly distributed in its environment, defined with different sizes and positions.

Table des matières

Introduction générale

L'opération de planification des trajectoires des robots mobiles, représente généralement un sujet de recherche très important surtout dans le domaine de la robotique mobile intelligente. En effet, une planification de trajectoire, est une opération de détermination d'un chemin, que le robot mobile objet d'opération de guidage, doit l'emprunter en passant à travers un ensemble de points situés dans un environnement donné et en minimisant en même temps un coût total associé au chemin suivi.

Durant les dernières décennies, de nombreuses recherches ont été consacrées à la résolution du problème de planification des trajectoires des robots mobiles, ceci implique l'apparition de nombreuses techniques, telles que le champ de potentiel artificiel [1], les graphes de visibilité [2] et la méthode de décomposition cellulaire [3], etc….

La méthode dite champ de potentiel [1], est largement utilisée en raison de sa structure simple et de sa facilité d'exécution, mais cette dernière peut être confrontée le problème des minima locaux, qui conduit à une situation de stagnation du robot dans son environnement. Celle dite, le graphe de visibilité [2], elle nécessite plus de précision de contrôle, puisque l'efficacité de sa recherche de chemin est faible comme décrit dans [3]. Concernant l'approche de décomposition cellulaire [3], l'environnement est divisé en un certain nombre de cellules, dont la taille et la forme sont prédéfinies. En effet, cette méthode ne convient pas à la planification de trajectoire en temps réel et peut être utilisée lorsque l'espace de travail du robot est connu. Bien que, ces méthodes ne soient pas forcément exclusives, leur hybridation est souvent utilisée pour modéliser des planificateurs de mouvement.

Les algorithmes de colonies de fourmis sont des algorithmes inspirés du comportement des fourmis et constituent une famille de métaheuristiques d'optimisation pour résoudre naturellement des problèmes complexes. Une telle aptitude s'avère possible en raison de la capacité des fourmis à communiquer entre elles indirectement, par le dépôt dans l'environnement de substances chimiques, appelées *phéromones*. Ce type de communication indirecte est appelé *stigmergie*. En anglais, le terme consacré à la principale classe d'algorithmes est Ant Colony Optimization (ACO). En effet, Le premier algorithme d'optimisation s'inspirant de cette analogie a été proposé par Colorni, Dorigo et Maniezzo [4], afin de résoudre le problème dit de voyageur de commerce [4].

Ce type d'algorithme a été utilisé dans la résolution du problème de planification de trajectoires de robots mobiles, on cite dans ce qui suit quelques travaux :

Montiel-Ross et ses collègues [34], ont développé un logiciel de navigation appelé Ant Colony Test Center. Le logiciel développé est basé dans ces fondements, sur l'aspect des colonies de fourmis, il est

capable de réaliser à la fois, une génération de chemin, une planification et même un suivi d'un chemin virtuel.

You et ses collègues [35], ont développé un nouveau modèle dynamique de recherche, conçu pour le problème de planification de trajectoire d'un robot mobile. En effet, leur modèle a permis d'améliorer significativement la qualité des recherches, en s'appuyant sur les deux caractéristiques principales, la diversification et la vitesse de convergence.

Récemment Wenxiang Gao et ces collègues [36], ont proposé une variante des ACO, appelée en anglais : Enhanced Heuristic Ant Colony Optimization (EH-ACO), basée dans ses améliorations sur quatre stratégies optimisatrices, qui permettent d'obtenir de bons résultats comparant avec les versions des ACOs proposées précédemment.

Le présent travail de Master, vise à présenter une méthode de planification d'un chemin basée sur un algorithme ACO sous sa forme primitive. Il est à noter que, la planification de mouvement d'un robot mobile intelligent est considérée ici, comme un problème d'optimisation, dont la longueur du chemin suivi lors du processus de navigation, représente la grandeur à optimiser.

Le présent manuscrit, est organisé comme suit :

Le premier chapitre, contient un aperçu général sur la robotique mobile,

Le second chapitre, présente une vue générale sur quelques théories relatives au problème de planification d'une trajectoire pour un robot mobile,

Une présentation détaillée de quelques algorithmes (les plus connus) relatifs au problème traité (planification) est exposée dans le troisième chapitre,

La méthode sur laquelle repose ce mémoire, des essais de simulation avec leurs analyses, sont détaillées au quatrième chapitre,

Le présent manuscrit est achevé par une conclusion générale.

Chapitre I

Généralité sur la robotique mobile

I.1. Introduction

L'objectif de ce chapitre est de donner un aperçu général sur la robotique mobile. Après une présentation des différents types des robots mobiles, et des contraintes de terrain sur lequel ils sont conçus pour évoluer, nous aborderons la classification des plateformes mobiles et leurs principales structures cinématiques. Enfin nous étudions les outils permettant aux robots de percevoir leur environnement, étape indispensable et nécessaire à l'autonomie totale des robots mobiles.

I.2. Définition d'un robot mobile

En réalité, il existe deux principaux types de robots à savoir [5] : les robots manipulateurs et les robots mobiles. En effet, les robots manipulateurs ont une base fixe contrairement aux robots mobiles qui peuvent se déplacer. Les robots mobiles à roues sont en effet les systèmes les plus étudiés, parce qu'ils sont plus simples à réaliser que les autres types de robots mobiles, ce qui permet d'en venir plus rapidement à l'étude de leur navigation [5].

Ce type de robots est notamment très souvent utilisé pour l'étude des systèmes autonomes. Vient ensuite la robotique mobile à pattes, avec spécialement la robotique humanoïde, mais également des robots avec un nombre de pattes plus élevées qui offrent de bonnes propriétés pour la locomotion en milieu pénible (milieux forestiers et agricoles). La stabilité des mouvements de ce type de robots est en particulier un thème de recherche important [3,20].

Enfin il existe également de nombreux autres types de robots mobiles (robots marins, sous-marins, drones volants, micro et nano robots), généralement l'étude de ce type de robots se fait dans des thématiques spécifiques avec des problèmes particuliers à l'application visée.

I.3. Autonomie d'un robot mobile

Généralement, Il existe deux principaux modes de fonctionnement pour un robot mobile [5] : téléopéré et autonome. Concernant le premier mode (téléopéré), une personne travail à piloter le robot

à distance. Elle donne ses ordres à travers une interface de commande (joystick, clavier/souris…), et ceux-ci sont envoyés au robot via un lien de communication (internet, satellite …). D'ailleurs, suivant le niveau de télé-opération, le terme ''robotique'' est plus ou moins justifié. Le robot doit donc obéir aux ordres de l'opérateur qui perçoit l'environnement autour du robot, par différents moyens (retour d'image, retour d'effort, …), de manière à donner des ordres adaptés au robot. Dans ce domaine, les efforts de recherche sont beaucoup portés sur les problèmes liés au réseau de télécommunication (retards dans le réseau de communication, problèmes de commande, pertes de données) et sur l'amélioration de la perception de l'environnement par l'opérateur (interfaces images, retours d'efforts).

A l'autre côté, c-à-d en mode autonome le robot doit prendre ses propres décisions. Cela signifie qu'il doit être capable à la fois de découvrir correctement son environnement, mais également de savoir comment réagir (en temps réel) en conséquence, suivant le niveau d'autonomie. C'est à lui de planifier son parcours et de déterminer avec quels mouvements il va atteindre son objectif. Les recherches dans ce domaine portent principalement d'une part sur la *localisation* du véhicule autonome et la *cartographie* de son environnement, d'autre part sur le contrôle de tels véhicules (structure de contrôle, stratégies de commande, planification).

Cette notion d'autonomie prise en exemple ci-dessus, que nous pourrions qualifier de décisionnelle, ne doit pas être confondue avec celle d'autonomie énergétique (capacité du robot à gérer efficacement son énergie, à la préserver, voire à se ravitailler), même si ces deux notions sont étroitement liées : idéalement une des préoccupations principales d'un robot mobile totalement autonome (du point de vue décisionnel), serait en effet de pouvoir gérer de lui-même ses réserves d'énergie.

Dans ce qui suit, nous présentons les différents types d'environnement dans lesquels les robots mobiles sont amenés à se déplacer.

I.4. Les différents types de terrains

Généralement, nous rencontrons spécialement trois types d'espaces de navigation : qui sont respectivement les terrains plats, les terrains accidentés et les espaces 3D. Les terrains plats sont généralement utilisés pour modéliser les milieux urbains et les intérieurs de bâtiments. Le robot évolue sur un plan 2D considéré sans pentes, et tout objet qui sort de cet espace 2D est considéré comme un obstacle (voir Figure I.1).

Cette représentation est la plus simple à étudier et la plus répandue pour les robots mobiles à roues. En première approche, elle permet de se concentrer sur les problèmes de contrôle et de navigation autonome du robot.

Figure I.1. Robot Taxi.

Les terrains accidentés correspondent généralement aux milieux en extérieur, comme des forêts, des champs en robotique agricole, ou encore des terrains rocheux (Figure I.2). La différence avec les terrains plats est la présence de pentes, de bosses et de creux sur le terrain d'évolution du robot. Cela interdit d'utiliser une métrique standard 2D et cela complique pour beaucoup la détection d'obstacles et la modélisation des déplacements du robot. De plus il devient également important de vérifier que le robot ne bascule pas quand il escalade une pente ou enjambe un obstacle. Le système de déplacement du robot doit dans ce cas être adapté à la topologie du terrain.

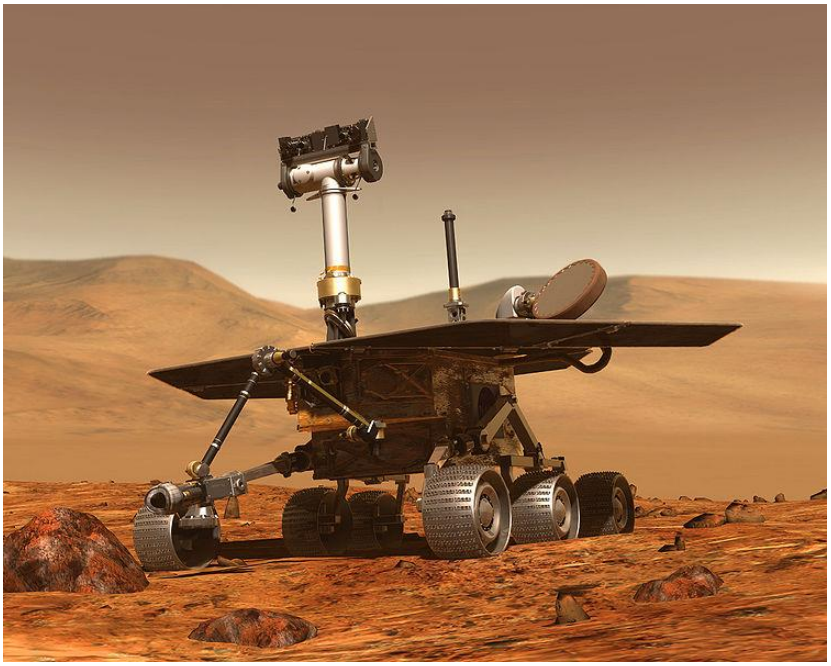


Figure I.1. Le robot martien Rover.

Enfin les espaces d'évolution 3D sont par exemple utilisés pour modéliser la navigation des robots sous-marins (Figure I.3) et des drones volants (Figure I.4). Les problèmes rencontrés sont spécifiques à l'application visée.

Figure I.3. Robot sous-marin.

Figure I.4. Drone volant.

Il est à noter que, chaque type de terrain correspond à des problématiques bien spécifiques.

Pour la modélisation terrain plat, nous définissons un repère absolu (fixé dans l'environnement) $R = (O, X, Y)$, donc l'axe z est perpendiculaire au sol. Nous définissons un repère mobile lié au robot $R^r = (O_r, X_r, Y_r)$ dit égocentrique. Le point O_r est le point de contrôle du robot. Généralement, sur un robot type voiture, le point de contrôle est fixé au centre de l'essieu non directeur. Ce repère égocentrique se déplace avec le robot. Pour réaliser une navigation, l'état du robot est totalement défini par le vecteur :

$$X = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \quad \text{(I.1)}$$

Dans lequel (x, y) désignent la position du robot dans le plan $(o, \vec{x}, \vec{y})$ et θ son orientation (Figure I.5).

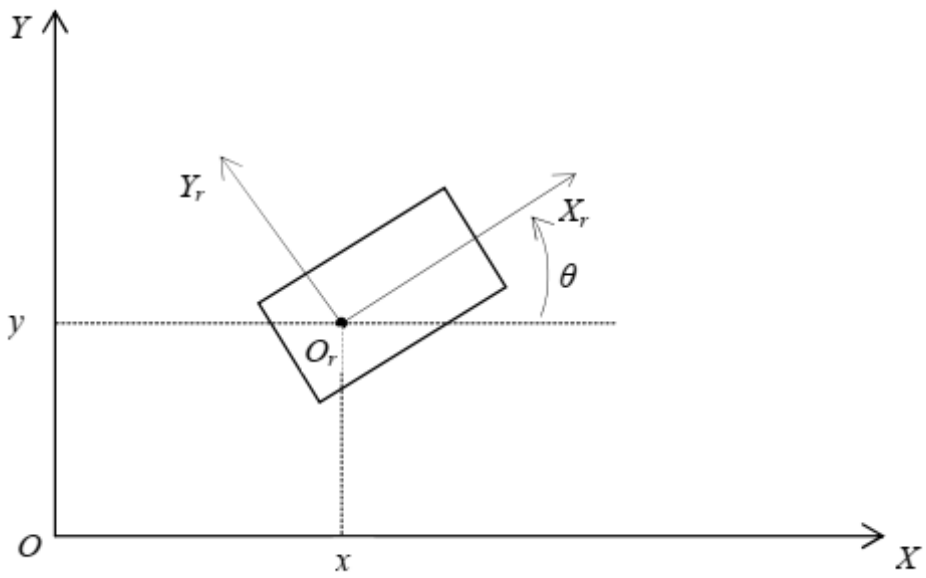


Figure I.5. Modélisation du robot dans le repère absolu.

La section suivante présente les différents aspects de la modélisation sur sol plat : la classification des différents types de roues, les configurations non holonomes et holonomes, et la gestion des glissements dans le modèle.

I.5. Modélisation des robots mobiles

I.5.1. Classification des types des roues

La mobilité d'un robot mobile dépend grandement du type de roues utilisées. Une classification des différents types de roues rencontrées en robotique mobile est illustrée ci-dessous :

- la roue fixe : cette roue n'autorise qu'un déplacement dans la direction de son plan médian, l'orientation n'est pas modifiable,

- la roue centrée orientable : elle possède un axe d'orientation en plus de l'axe de rotation, et cet axe d'orientation passe par le centre de la roue,

- la roue décentrée orientable ou roue folle : son axe d'orientation ne passe pas par le centre de la roue (c'est le cas par exemple des roues des chaises de bureau).

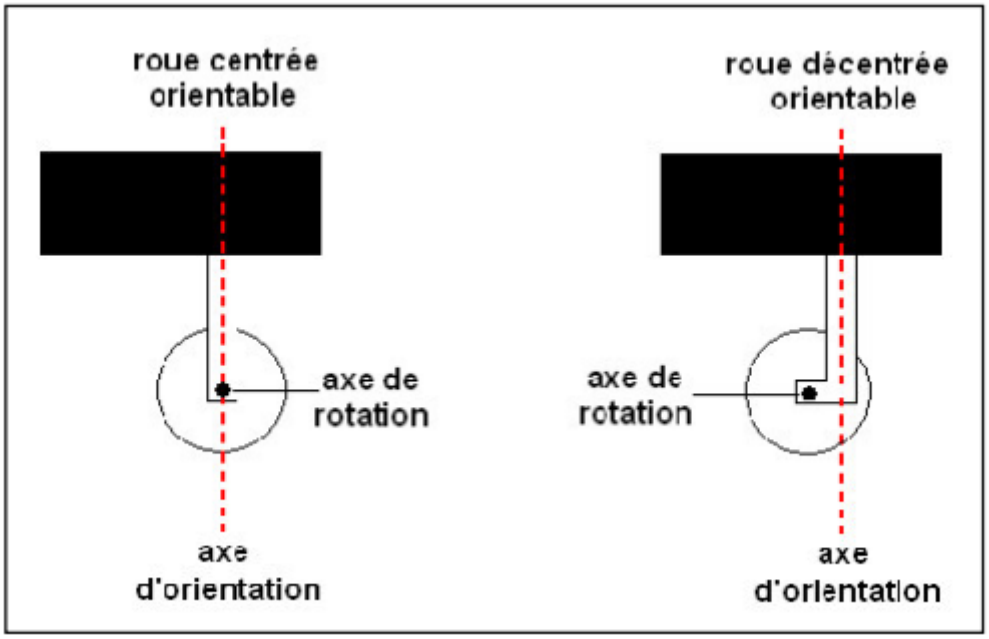


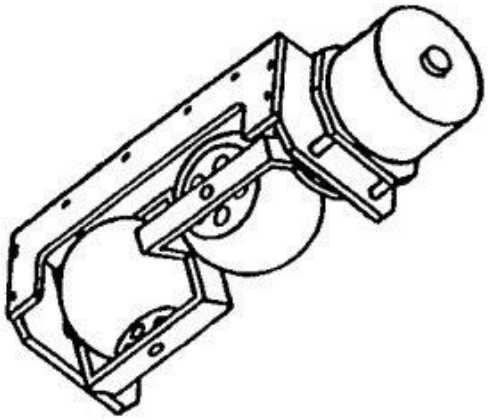
Figure I.6. Roue centrée et roue décentrée.

En plus de ces roues classiques (Figure I.7), d'autres roues ont été développées pour accroitre la mobilité du robot. Elles permettent d'augmenter les capacités de déplacement dans toutes les directions du plan (Figure I.7). Néanmoins, elles ne sont commandables que dans certaines de ces directions. Dans cette catégorie nous trouvons notamment :

- les roues suédoises : ces roues autorisent les glissements latéraux grâce à un système de galets remplaçant la bande de roulement classique, montés en inclinaison par rapport au plan de la roue. La combinaison de la rotation de la roue avec la rotation libre du galet en contact avec le sol permet le déplacement sans glissement sur le sol dans toutes les directions. Cependant le couple moteur que l'on peut transmettre à ces roues est très limité, ce qui réduit son utilisation en pratique.

- les roues tronco-sphériques (ou orthogonal wheels) : cette structure utilisant deux roues libres en quadrature présente l'avantage de pouvoir transmettre un couple intéressant par rapport aux roues suédoises, mais souffre de petits problèmes de sauts au moment de la transition d'une roue support à

l'autre.



roue troncospherique

roue suedoise

Figure I.7. Roue tronc-sphérique et roue suédoise.

I.5.2. Types des plates formes mobiles

En associant les différents types de roues selon une structure mécanique donnée, le robot mobile disposera de plus ou moins de mobilité. Le nombre, le type et la disposition des roues engendrera ou non la contrainte de non-holonomie du robot. Si on néglige les phénomènes dynamiques tels que l'inertie, un robot holonome est un robot capable à chaque instant de se déplacer dans n'importe quelle direction du plan, sans avoir à effectuer une reconfiguration de ses roues.

Tout système évoluant dans un plan 2D possède 3 degrés de liberté : une translation selon l'axe x, une translation selon l'axe y et une rotation autour d'un axe z normal à $(\vec{x}, \vec{y})$. Cependant une roue classique ne possède que 2 degrés de mobilité : elle ne peut que faire une translation (avancer ou reculer), ou une rotation sur elle-même. Elle ne peut pas déraper transversalement pour effectuer un mouvement de translation horizontal. Cette contrainte empêche la plupart des véhicules traditionnels d'effectuer un déplacement instantané transversal (parallèlement à l'axe de rotation de la roue). Une voiture ne peut pas effectuer de créneau pour se garer, sans faire un certain nombre de manœuvres. C'est une contrainte que l'on retrouve sur tous les robots mobiles de type voiture ou à roues différentielles. De tels véhicules, possédant un nombre de degrés de mobilité inférieur au nombre de degrés de liberté, sont dits non-holonomes, et cette contrainte touche principalement les robots mobiles à roues.

II.5.2.1. Plates formes non-holonomes

Les systèmes mobiles dit non-holonomes sont ceux que l'on rencontre le plus dans la vie courante (voiture particulière, bus, camion, …etc.). Ces systèmes ont une structure mécanique relativement simple (des roues motrices, des roues directrices et des roues libres).

Les configurations non-holonomes les plus courantes sont :

a) Plates formes de type tricycle et voiture : ces deux structures sont constituées d'un axe fixe (généralement à l'arrière) et d'un axe directeur (Figure I.8). Dans le cas du tricycle, seule une roue est présente sur l'axe directeur, contrairement à la voiture qui en possède deux. La théorie d'Ackerman-Jeantaud donne les conditions théoriques de non glissement et non dérapage pour les configurations de type voiture. Notamment les axes de rotation des quatre roues doivent s'intersecter en un point unique, le Centre Instantané de rotation. Pour cela, la vitesse de la roue extérieure doit être légèrement supérieure à celle de la roue intérieure. La structure de type voiture peut être modélisée par une structure équivalente à trois roues, ce qui revient au modèle du tricycle.

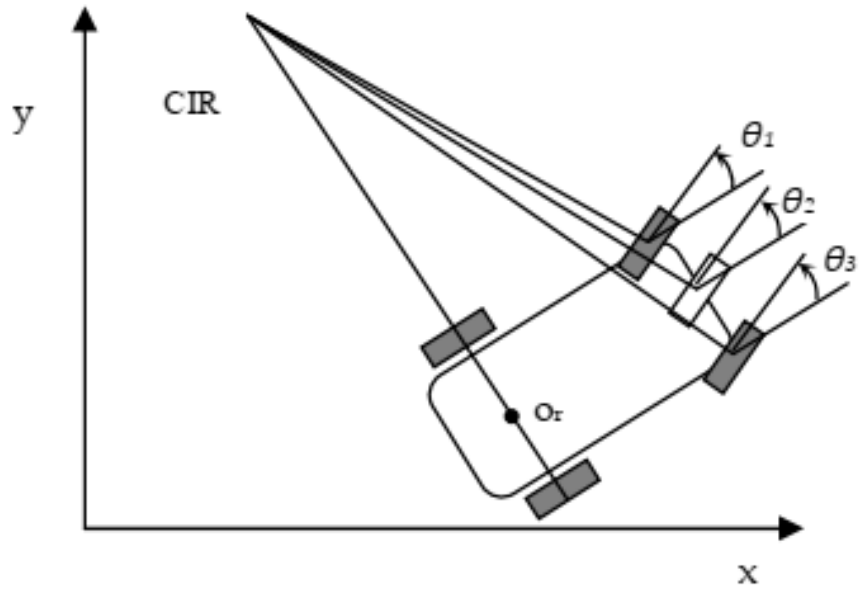


Figure I.8. Equivalence modèle voiture et tricycle.

b) Plates formes à roues différentielles : cette structure est constituée de deux roues motrices placées sur le même axe (Figure I.9), et d'au moins un appui supplémentaire (généralement une ou deux roues folles). L'avantage de cette structure est qu'elle permet au véhicule de tourner sur place, suivant si les vitesses de rotation des deux roues motrices sont de signe opposé ou pas. Ainsi le robot peut pivoter rapidement, ce qui donne des capacités de déplacement intéressantes. Cependant, le déplacement latéral n'étant pas directement réalisable, cette structure n'est pas non plus holonome.

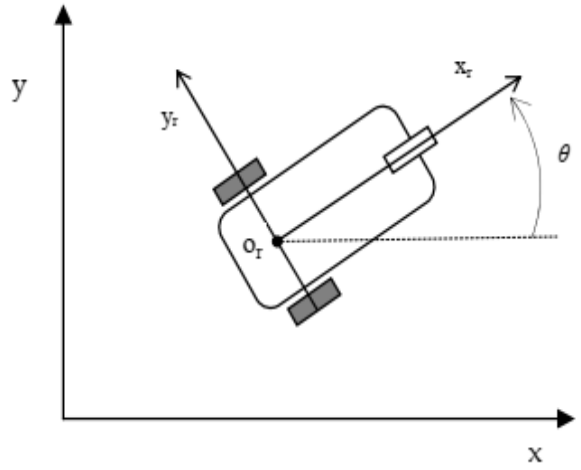


Figure I.9. Modèle de type tricycle à roues différentielles.

I.5.2.2. Plates formes holonomes

En utilisant des roues telles que les roues suédoises ou tronco-sphériques sur des plateformes mobiles, des robots ayant la capacité de se mouvoir dans toutes les directions ont été créés. Ces robots, à trois degrés de mobilité dits omnidirectionnels, permettent de s'affranchir de la contrainte de non holonomie. Leurs structures spécifiques leur permettent de se déplacer instantanément (à la dynamique près) dans toutes les directions en ayant n'importe quelle orientation, rendant possible le suivi de trajectoires de forme quelconque (Figure I.10).

Figure I.10. Le robot omnidirectionnel Nomadic XR4000.

I.5.3. Roulement avec ou sans glissement

La locomotion se fait grâce au frottement entre la roue du véhicule et le sol, et l'efficacité du mouvement dépend notamment du type de sol. Pour que l'hypothèse du roulement sans glissement soit validée, il faudrait théoriquement que le contact sol/roue ne se fasse qu'en un point, que le sol soit parfaitement plat, et que le rayon de la roue soit parfaitement constant sur toute sa périphérie (Figure I.11).

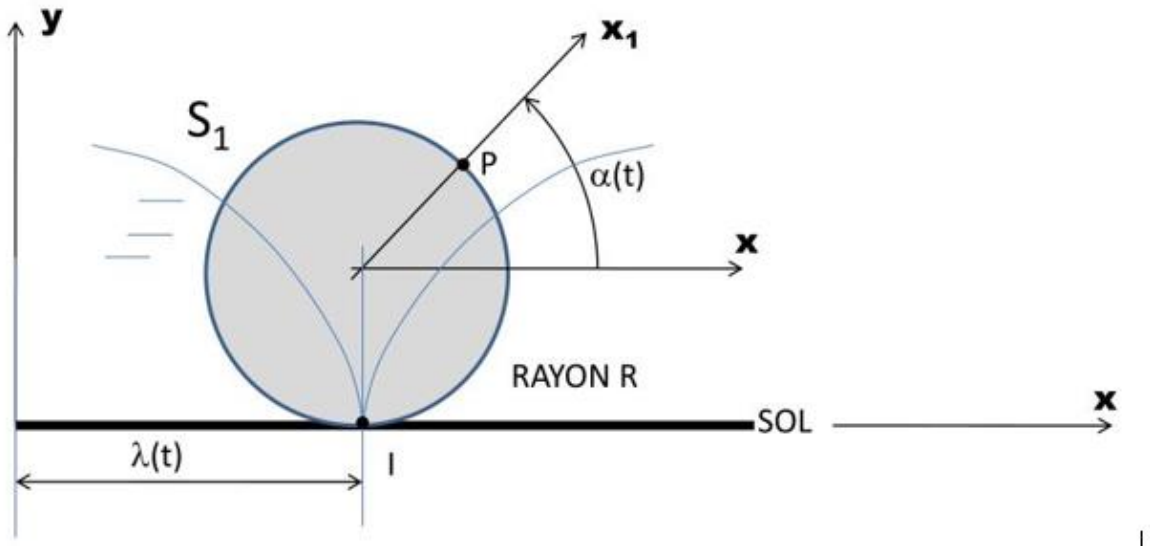


Figure I.11. Roulement sur le sol.

En effet, le contact sur le sol se fait sur une surface avec le pneu de la roue. Les glissements sur le sol sont une source d'erreur importante pour certaines méthodes de localisation. Cela est notamment le cas pour l'odométrie classique, qui s'appuie sur cette hypothèse pour déterminer la position relative d'un robot par rapport à son point de départ, à partir de la mesure du nombre de tours parcourus par chaque roue. On sait que dans ce cas l'intégration des dérives dues aux glissements, entraine sur un parcours complet d'importantes erreurs de localisation.

Dans certaines applications robotiques, la précision du déplacement est un facteur important, il est alors nécessaire de prendre en compte les glissements dans la modélisation du robot. Il existe deux principales approches pour les intégrer : soit on passe à un modèle dynamique du robot, soit on reste sur un modèle cinématique classique (roulement sans glissement) dans lequel on introduit les effets du glissement sous forme de paramètres supplémentaires.

La modélisation dynamique des phénomènes de glissements est plus complète mais elle requiert d'une part la mesure ou l'estimation, en temps réel, d'un grand nombre de paramètres tels que les coefficients de frictions, et d'autre part de paramètres supplémentaires inhérents à

l'état des pneumatiques. Ces nombreux paramètres ne sont pas triviaux à obtenir en ligne, ce qui limite l'utilisation de tels modèles en pratique. Les modèles cinématiques, modifiés pour intégrer les glissements, s'avèrent plus simple à mettre en œuvre de part le nombre réduit de paramètres à estimer (on ne modélise plus les phénomènes complets de glissement, mais juste leur effet sur la cinématique du véhicule), et ils permettent tout de même de prendre en compte avec une très bonne précision les phénomènes de glissement. Par exemple, dans [6], Luka et ses collègues développent un modèle cinématique étendue qui leur permet d'améliorer la précision du suivi de trajectoire par un véhicule sur un terrain particulièrement glissant.

I.6. Détection des obstacles et localisation

La perception de son environnement d'évolution est la base de tout système autonome. Sans une bonne perception et interprétation de ce qui l'entoure, un robot ne peut pas prendre de décision correcte. Cette partie vise à décrire les différents moyens mis à disposition au robot pour localiser les obstacles qui l'entourent. Ensuite les différentes méthodes de localisation du robot lui-même sont abordées. L'idée est de permettre au final de créer un modèle, plus ou moins simplifié, des interactions entre le robot et son environnement. Cette étape est nécessaire et primordiale pour la navigation d'un robot mobile autonome.

Pour cela, un robot est équipé de capteurs proprioceptifs qui fournissent des informations sur le robot lui-même, et extéroceptifs qui fournissent des informations sur ce qu'il y a autour de lui (son environnement).

I.6.1. Détection des obstacles et cartographie

Les capteurs permettant de fournir des informations sur l'environnement extérieur peuvent être classés en deux catégories, passifs et actifs [7]. Dans le premier cas, on se contente de recueillir et d'analyser une énergie fournie par l'environnement, typiquement la lumière. Dans le second cas, c'est au capteur de générer une énergie, et de récupérer cette énergie après interaction sur le milieu extérieur. C'est le principe de base des télémètres (capteurs de mesures de distances), qui sont largement utilisés pour tracer des cartes en ligne de l'environnement dans lequel évolue le robot. Les télémètres laser à balayage sont généralement utilisés pour la navigation de robots avec de très bonnes performances notamment en intérieur.

Le principe de ces télémètres repose sur le calcul du temps aller-retour mis par une impulsion lumineuse pour revenir sur le capteur. Une onde infrarouge de faible puissance est émise par la diode laser, et au même moment un chronomètre informatique est lancé. L'onde se réfléchit sur le premier objet rencontré en chemin, et revient sur le détecteur du capteur.

Le temps mis par l'onde pour faire l'aller-retour permet de déterminer la distance de l'objet. Un

miroir tournant motorisé permet de balayer toute une gamme d'angles devant le télémètre, dans le plan de balayage qui est parallèle au sol. La précision de ces appareils et leur robustesse aux variations de température en font des outils très intéressants pour les applications en robotique mobile de faible et moyenne vitesse.

Les capteurs ultrasonores utilisent des ondes sonores de fréquence non perceptible par l'oreille humaine, généralement dans la fourchette 20-200 khz. De la même manière que les télémètres laser, ils sont basés sur le principe de la mesure du temps aller-retour lors de la réflexion sur un obstacle. C'est la méthode employée par certains animaux pour percevoir leur environnement, comme les chauves-souris ou les dauphins : l'écholocalisation. Un avantage de ces capteurs est que contrairement aux télémètres, l'onde qu'ils émettent n'étant pas focalisée, ils perçoivent beaucoup plus facilement des éléments filiformes comme des pieds de chaises ou des grillages. Par contre leur portée est faible, et ils sont moins adaptés aux milieux de propagation non isotropes comme l'air.

Un des inconvénients des capteurs ultrasonores par rapport aux télémètres lasers est la divergence importante du faisceau ultrasonore, qui s'apparente plus à un cône qu'à un faisceau.

Généralement l'ouverture de l'angle est de plusieurs dizaines de degrés, ce qui rend la localisation des obstacles imprécise. Ces capteurs sont donc plutôt utilisés pour des mesures à courte distance (de quelques centimètres à quelques mètres). Ils sont relativement sensibles aux variations de température, et la fréquence de mesure dépend de la distance maximale de détection (plus cette distance est grande, moins la fréquence d'acquisition des mesures est élevée). L'avantage de ces capteurs est qu'ils sont moins onéreux qu'un télémètre laser, et ils sont souvent utilisés dans des applications en intérieur avec des espaces de navigation assez restreints.

Les capteurs passifs se servent directement de l'énergie émise par l'environnement. C'est typiquement le cas des systèmes de vision par caméra en stéréo vision. La reconnaissance de primitives entre deux images permet d'évaluer la position/orientation d'un objet, et ainsi d'évaluer la profondeur. L'utilisation simultanée de deux caméras est cependant nécessaire pour y parvenir. Plus de deux caméras peuvent également être utilisées, de manière à améliorer la robustesse de la méthode. La vision omnidirectionnelle s'avère également très intéressante dans le cadre d'applications en robotique mobile, dans le sens où elle permet de surveiller en même temps tout ce qui se passe autour du robot. La caméra est placée face à un miroir parabolique ou hyperbolique. L'image est complètement distordue ce qui complique la mesure de distances, mais l'aspect vision panoramique offre des avantages pour l'évitement d'obstacles dynamiques en environnement structuré. Les verticales deviennent des radiales, et les horizontales des arcs de cercle. L'utilisation de systèmes à base de vision est fortement développée, et pas seulement dans le domaine de la robotique. Mais globalement, ce type de système reste fortement tributaire de la qualité de l'énergie recueillie : influence de la luminosité ou encore du contraste.

I.6.2 Fusion de données multi-capteurs et cartographie

La localisation d'un robot mobile s'effectue par la mise en correspondance de différentes sources extéroceptives et proprioceptives. Généralement il s'agira de confronter les mesures de déplacements prises par odométrie avec une méthode de localisation absolue : soit reconnaissance et calcul de distance par rapport à des balises de position connue, soit mise en correspondance avec une carte construite en ligne et/ou présente dans une base de données, soit encore localisation externe du robot par des capteurs dans l'environnement (GPS).

En effet, le principe le plus simple pour effectuer cette mise en correspondance, consiste à utiliser les mesures de localisation absolue pour recaler périodiquement l'état du robot, obtenu par intégration des déplacements mesurés par l'odomètre. Cette méthode, bien que simple à utiliser, présente le problème de ne pas utiliser conjointement les différents moyens de mesure, mais successivement. Ainsi on ne tient pas compte des incertitudes liées tant à l'odométrie qu'à la méthode de localisation absolue. Cependant nous savons que quel que soit la technologie utilisée pour la prise d'informations, aucune mesure n'est parfaite, il existe toujours une part d'incertitude sur celle-ci. Ces incertitudes peuvent provenir soit du principe de mesure lui-même, soit des imperfections technologiques.

Typiquement pour une mesure de distance avec un télémètre laser, on trouve des erreurs systématiques d'une quinzaine de millimètres en moyenne (erreur constante intrinsèque au télémètre utilisé), et une erreur statistique de 05 mm environ. Lorsque plusieurs méthodes de mesure sont utilisées conjointement, le principe utilisé pour mettre en concordance les informations consiste à effectuer une moyenne pondérée des différentes mesures par la confiance que l'on accorde à chacune (inversement à leur variance donc). Soit deux mesures z_1 et z_2 d'une même variable x, obtenues par des capteurs différents, avec des variances associées σ_1 et σ_2, alors la loi de Bayes nous donne la valeur estimée de x :

$$\hat{x} = \frac{\sigma_2^2}{\sigma_1^2+\sigma_2^2} Z_1 + \frac{\sigma_1^2}{\sigma_1^2+\sigma_2^2} Z_2 \qquad \text{(I.2)}$$

et la variance associée à l'estimée :

$$\sigma = \frac{1}{\frac{1}{\sigma_1}+\frac{1}{\sigma_2}} \qquad \text{(I.3)}$$

La variance associée à cette estimation est plus faible que les variances de chacune des mesures prises séparément (ce qui est logique, cela traduit simplement le fait que plus on recoupe d'informations provenant de sources différentes, plus on diminue les incertitudes). Cette méthode est intéressante si l'on dispose de ressources limitées pour le calcul, mais l'estimation peut être largement améliorée en

utilisant les informations sur les mesures passées, et en filtrant avec un filtre de Kalman.

Pour des obstacles statiques, nous obtenons des mesures récurrentes selon une certaine fréquence d'acquisition. Pour des mesures récursives, la précision peut être améliorée en utilisant un filtre de Kalman. Ce type de filtrage, très utilisé notamment en automatique, est un filtre statistique qui permet de réduire les incertitudes au fur et à mesure de l'acquisition de nouvelles mesures. Cette méthode est particulièrement utilisée en robotique pour la localisation du robot relativement aux obstacles [8]. L'algorithme utilise les connaissances sur la dynamique du robot et du système de mesure et sur les incertitudes associées à chaque mesure. Le calcul s'effectue en deux phases : une phase de prédiction de la mesure et de sa variance, suivie d'une phase de mise à jour de celle-ci par l'acquisition de nouvelles mesures.

L'historique des mesures n'a pas besoin d'être gardé en mémoire, et le processus est récursif. Pour pouvoir planifier les déplacements du robot, il est nécessaire d'établir une modélisation de l'environnement à partir des mesures des positions relatives des obstacles par rapport au robot. Concrètement il s'agit d'établir une cartographie locale des espaces où le robot pourra circuler, ou non, en considérant la position absolue du robot comme connue. Il existe deux grands types de représentation pour l'environnement local : les cartes géométriques et les grilles d'occupation. Les premières peuvent être obtenues par traitement des données issues de mesures télémétriques, et en effectuant une reconnaissance des formes simples (murs, coins).

A partir d'une connaissance des déplacements du robot et en comparant la carte courante avec des cartes mises en mémoire au fur et à mesure que le robot se déplace, la robustesse de la carte peut être améliorée. Les grilles d'occupation sont des cartes discrétisées, généralement sous forme de grille d'un certain nombre de lignes et de colonnes. A chaque case de la grille est soit associée une valeur booléenne pour dire si la case est accessible par le robot ou non (occupée par un obstacle ou libre), soit une probabilité d'occupation (loi de Bayes).

I.6.3 Localisation d'un robot mobile

Les outils permettant la localisation d'un robot dans son environnement peuvent être classés en deux catégories : ceux par localisation à l'estime et ceux par localisation absolue [8]. Le principe de la première catégorie consiste à intégrer des informations sur les vitesses ou les accélérations fournies par des capteurs proprioceptifs (odomètres, centrales inertielles). L'avantage de ces méthodes est qu'elles sont indépendantes de l'environnement, par contre leur souci est leur manque de précision dû à la dérive temporelle.

En effet, les erreurs s'intégrant elles aussi au fur et à mesure du temps, il est nécessaire d'apporter régulièrement des recalages (Figure I.12).

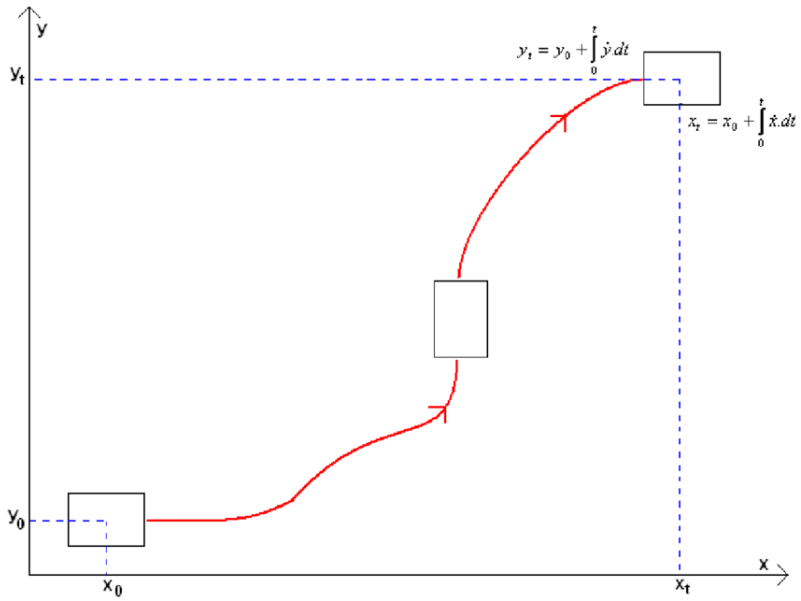


Figure I.12. Localisation à l'estime.

Parmi les méthodes de localisation à l'estime, le système le plus simple et le plus couramment utilisé pour la mesure de déplacement du robot est l'odométrie. L'hypothèse de roulement sans glissement que nous avons vu précédemment, nous permet de relier directement les déplacements du robot à la vitesse de rotation des roues. Par intégration des déplacements à chaque instant, on en déduit la position relative du robot par rapport à son point de départ.

L'odométrie est une méthode de localisation très courante, simple, mais également très rapidement imprécise. En effet à cause du glissement des roues sur le sol, les erreurs s'accumulent au fur et à mesure que le robot avance, ce qui implique d'importantes erreurs sur les longs parcours s'il n'y a pas de recalage régulier. Cette méthode est de ce fait fortement tributaire de la qualité du sol sur lequel le robot se déplace.

Les incertitudes sur le diamètre exact des roues, sur les paramètres géométriques du robot, sur la résolution des codeurs, génèrent des erreurs de type systématique, qui vont s'accumuler très rapidement en odométrie. Cependant, ces erreurs peuvent être identifiées et évaluées pour faire un recalibrage du système et ainsi améliorer sa précision.

Les erreurs non systématiques comme les glissements ou les irrégularités du sol, génèrent moins rapidement des erreurs, mais ne peuvent par contre pas être recalibrées puisqu'on ne peut pas les prévoir. Pour l'exploration martienne, où le terrain est fortement accidenté, l'utilisation de système d'odométrie classique est impossible.

Le second type de méthode pour la localisation est la localisation absolue. Ces méthodes utilisent des éléments repérables par le robot dans l'environnement de navigation, de position connue, pour permettre au robot de se repérer relativement à ceux-ci. Ces éléments sont appelés des balises ou

amers et sont dits soit réels, s'ils ont été placés spécialement pour permettre la localisation, soit virtuels s'il s'agit d'éléments présents naturellement (Figure I.13).

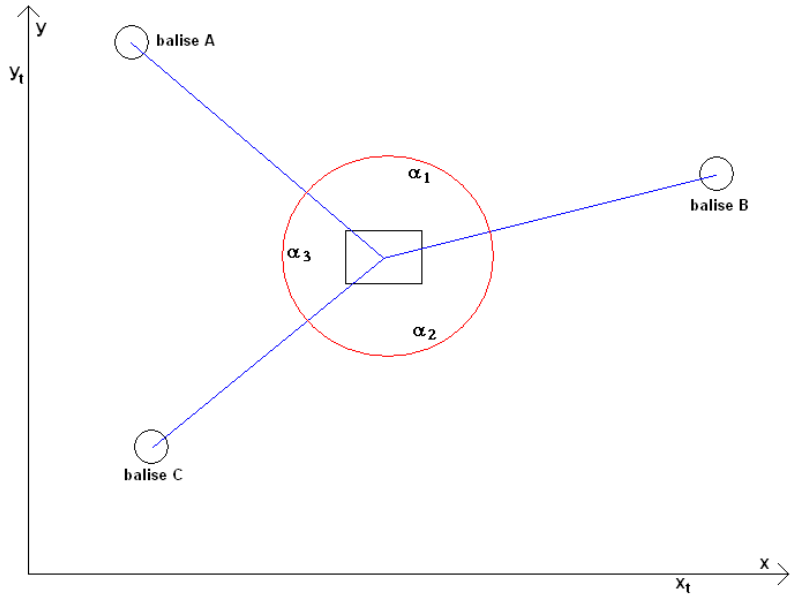


Figure I.13. Localisation absolue (méthode par triangulation).

Les balises réelles sont dites passives si elles ont pour but de réfléchir un signal émis par un appareil de mesure du robot (laser ou infrarouge). Il existe deux méthodes pour utiliser ces balises pour la localisation du robot : la méthode télémétrique (calcul de la distance robot/balise), qui nécessite la présence de deux balises pour calculer la position du robot dans le plan ; et la méthode par triangulation, qui consiste à mesurer les angles entre chaque balise et le robot, et qui elle nécessite l'utilisation de trois balises.

Les balises réelles sont dites actives si elles émettent un signal captable par le robot. En milieu extérieur, le système GPS (Global Positioning System) peut être utilisé pour obtenir des positions d'une précision de l'ordre du mètre. A la base développée par l'armée américaine dans les années 80 (lancement du premier satellite GPS en 1978), il fut ouvert aux civils en 1995.

Jusqu'en 2000 les mesures étaient volontairement entachées d'une erreur d'une centaine de mètres, l'armée américaine craignant que ce système soit un avantage pour leurs ennemis. Malgré le retrait de cette erreur volontaire, la précision du système restait de l'ordre du mètre, à cause des incertitudes sur l'orbite et l'horloge des satellites, ainsi que les retards engendrés par la traversée des couches atmosphériques.

Pour améliorer cette précision, nous pouvons utiliser les GPS différentiels : avec un second récepteur GPS sur une base fixe et de position connue, il devient possible de mesurer l'erreur et d'en déduire la correction à apporter pour la zone environnante. Pour que ce système fonctionne, il faut que la base

mobile reste à une certaine portée de la base fixe. Cette distance varie suivant la gamme de fréquence utilisée pour l'envoi des corrections, et peut atteindre quelques dizaines de kilomètres pour les besoins de la navigation maritime.

Des satellites géostationnaires permettent également de corriger certaines erreurs de position. Ils envoient des corrections sur les orbites et les horloges des satellites GPS. Ainsi pour l'Europe c'est le système EGNOS (European Geostationary Navigation Overlay System) qui se charge d'envoyer ces corrections. La précision atteinte est de l'ordre de 3m. Le système de GPS RTK (Real Time Kinematics) permet d'améliorer la précision à quelques centimètres, en utilisant la différence de phase de l'onde porteuse du signal, sa longueur d'onde étant d'une vingtaine de centimètres.

En contrepartie la portée de la station de référence avec cette méthode n'est plus que de quelques kilomètres. Pour augmenter cette portée on a recours au RTK réseau, qui va utiliser la redondance d'informations pour communiquer les corrections à l'appareil, via un serveur de calcul à distance. Les systèmes de localisation GPS sont très intéressants en rase campagne ou en banlieue. Cependant ils s'avèrent beaucoup moins efficaces en pleine ville ou en forêt.

En effet pour fonctionner correctement un GPS a besoin de recevoir les informations de quatre satellites au moins, or la présence d'obstacles tels que les ponts et grands bâtiments en ville empêche parfois cette réception. Ainsi il n'est pas rare de perdre la localisation GPS pendant quelques minutes. C'est pour cela que beaucoup de recherches dans le domaine de la localisation sont aujourd'hui portées sur les méthodes alternatives au GPS en milieu urbain.

La vision peut être un moyen pour compléter la localisation par GPS. Ainsi on peut utiliser un modèle 3D de caméras embarquées sur le robot pour déterminer précisément la position du robot mobile.

I.6.4. Localisation et cartographie simultanées

En robotique mobile, le SLAM (Simultaneous Localization And Mapping) consiste, pour un robot évoluant en milieu inconnu, à tracer une carte de l'environnement et localiser simultanément le robot dans celle-ci. La carte est construite de manière incrémentale au fur et à mesure que le robot évolue dans le terrain. En croisant les données perçues avec les informations géographiques dont il dispose en mémoire, le véhicule est capable de se localiser par rapport à des cartes préexistantes. L'idée du SLAM est donc de traiter conjointement les problèmes connexes que sont la navigation et la localisation d'un robot autonome.

La méthode alternative, moins gourmande en temps de calcul, consiste à utiliser la mémoire visuelle du véhicule, en repérant des éléments caractéristiques, pour se localiser par rapport à ceux-ci. Ces éléments sont extraits sous forme de primitives visuelles, que le robot cherchera à retrouver dans les images qu'il perçoit pour suivre un chemin précis.

I.7 Les thématiques scientifiques de la robotique mobile

Il existe de nombreux sujets de recherche dans le milieu de la robotique mobile autonome, ce qui montre qu'aujourd'hui encore le problème spécifique des robots mobiles autonomes est entier. La communauté des chercheurs dans le domaine de la robotique a dégagé quatre grands axes de travail autour desquels s'articulent les recherches actuelles en robotique mobile [8] :

- Techniques de localisation et cartographie : cet axe regroupe tous les développements autour de la perception et de la localisation du robot. On y retrouve notamment les méthodes SLAM (Localisation et Cartographie Simultanées). Plus récemment l'utilisation de bases de données sous forme de cartes 2D ou 3D, mais également sous forme SIG (Système d'Informations Géographiques) a ouvert de nouvelles perspectives dans ce domaine. De manière générale la fusion de données est également un thème important, tant la nécessité de coupler diverses sources de mesures apparaît nécessaire pour améliorer la précision et garantir l'intégrité des informations,

- Contrôle et commande des véhicules : cet axe regroupe les thématiques liées à la planification de chemin, la génération de trajectoires, et la commande des robots de manière générale. Une prise en compte de plus en plus poussée des contraintes et de la dynamique des robots est nécessaire, pour adapter au mieux les robots à leur environnement. La bonne gestion des obstacles et la prise en compte des incertitudes de mesures sont également des points clés de cette thématique,

- La communication inter-véhicules : on retrouve ici tous les travaux liés à la coopération entre robots, et le contrôle de flottilles de véhicules,

- L'interprétation de scènes : les recherches dans ce domaine visent à pousser plus loin la perception de son environnement par le robot, que la simple reconnaissance des objets. En effet dans certaines applications il est nécessaire que le robot appréhende plus finement son environnement que par une simple détection et localisation des obstacles. Les travaux concernent notamment la perception multi-capteurs et la représentation dynamique des scènes.

I.8 Conclusion

Le développement de l'intelligence artificielle classique conduit à la réalisation des premiers robots mobiles, évoluant dans des environnements strictement contrôlés. Leurs actions dans le monde réel n'étaient que l'expression physique d'opérations symboliques réalisées dans un modèle abstrait. Ces robots étaient incapables de réagir de manière appropriée à des perturbations ou à des événements inattendus, ce qui a rendu impraticable leur utilisation dans le monde réel.

La perception d'une part et la commande d'autre part sont donc les deux thèmes majeurs de recherche pour obtenir un robot mobile parfaitement autonome. Parmi les problématiques liées à la commande, celle de la navigation tient un rôle important : elle consiste à déterminer les trajectoires

que le robot doit suivre pour évoluer correctement au milieu d'obstacles, en considérant qu'il dispose d'une méthode de navigation.

Dans le chapitre suivant, nous parlerons au sens général sur quelques théories principales liées au problème de planification d'une trajectoire pour un robot mobile.

Chapitre II

Planification d'une trajectoire pour un robot mobile (aspect théorique général)

II.1. Introduction

La planification d'une trajectoire relative à un robot mobile placé dans un point A à un autre point B, avec un évitement simultané des obstacles, associé à une réaction aux changements d'environnement sont logiquement, des tâches très simples pour les humains mais ce n'est pas le cas pour un véhicule autonome (robot mobile).

Ces tâches présentent généralement, des défis majeurs que chaque robot mobile à roues doit surmonter pour devenir autonome. Un robot utilise des capteurs pour percevoir son environnement (avec des degrés d'incertitudes) et construire ou mettre-à-jour sa carte environnementale. En effet, afin de déterminer les actions de mouvement appropriées qui amènent (le robot) à l'emplacement objectif, des algorithmes dits de décision et de planification doivent être utilisés. Il est à noter que, dans un processus de planification d'une trajectoire, les contraintes cinématiques et dynamiques du robot devrait être considéré.

Une planification d'un chemin, est utilisée généralement pour résoudre des problèmes dans des différents domaines, de la simple planification d'un itinéraire spatial jusqu'à la sélection d'une séquence d'actions appropriée qui est nécessaire pour atteindre un certain objectif. Puisque l'environnement n'est pas toujours connu préalablement, ce type de planification est souvent limité aux environnements conçus à l'avance et des environnements que nous pouvons décrire avec précision acceptable avant le processus de planification.

En effet, l'action planification peut être utilisée dans des environnements entièrement connus/ou partiellement connus, ainsi que dans des environnements entièrement inconnus où les informations détectées définissent le mouvement souhaité du robot.

Le présent chapitre, présente un aperçu général sur quelques théories relatives à ce problème de robotique. Pour plus d'informations le lecteur peut consulter les références [3,5].

II.2. Environnement d'un robot mobile

Par définition, l'environnement dans lequel le robot se déplace il se compose d'un espace libre et autre occupé par des obstacles donnés (voir Figure II.1).

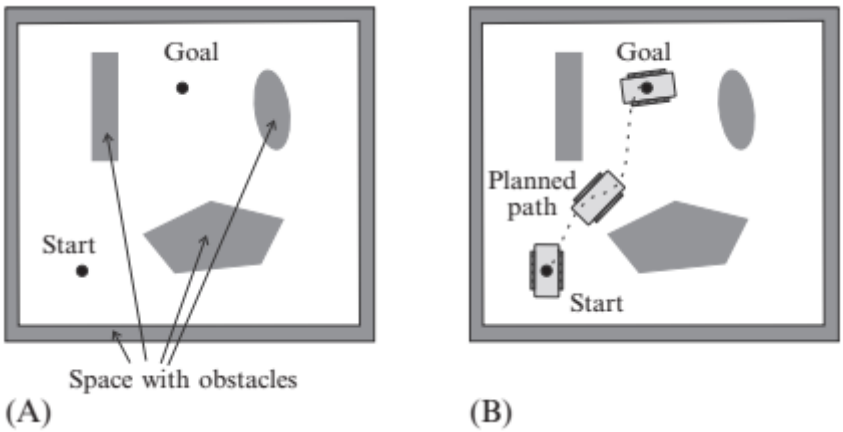


Figure II.1. Environnement avec des obstacles et des configurations de départ et d'objectif (d'arrivé), (B) Chemins possibles de départ et d'objectif (d'arrivé).

Comme la montre la Figure II.1, l'espace libre dans lequel existe le robot englobe généralement deux positions de départ et d'objective (d'arrivée).

Une configuration d'une trajectoire (chemin), est un ensemble complet de paramètres qui définisse le robot dans l'espace. Ces paramètres incluent généralement la position et l'orientation du robot ou également les angles de ses articulations. Le nombre de ces paramètres est égal au nombre de degrés de liberté (Degrees Of Freedom-DOF). Aussi, un environnement qui comprend des obstacles mobiles est dite dynamique, tandis que celle avec des obstacles fixes est dite statique. Également, l'identité de cette dernière (environnement) est dépondue de l'état de ces obstacles c.à.d. environnement connue a des obstacles connus est vice-versa.

II.3. Planification d'une trajectoire (chemin)

Une planification d'une trajectoire, consiste à trouver un chemin continu qui conduira le robot d'une position de départ à une autre d'arrivée. Le chemin entier doit se trouver dans l'espace libre (comme illustré sur la Figure II.1). Lors d'une planification, le robot mobile doit utiliser une carte environnementale qui est stockée généralement dans son propre mémoire.

Ordinairement, le robot mobile peut passer d'un état à un autre en mettant en œuvre des différentes actions. Un chemin approprié est donc décrit comme une séquence d'actions qui guident le robot de passer d'une configuration de départ (état), à travers certaines configurations intermédiaires jusqu'à

une dite de but ou d'arrivé. L'action choisie, dans l'état actuel et l'état suivant dépend de l'algorithme de planification utilisé, ainsi que quelques critères imposés. En réalité, le choix de l'état suivant (par l'algorithme) le plus approprié parmi tous les états possibles qui peuvent être visités, est dépond de l'état actuel. Cette décision, est prise en fonction de certains critères définies généralement avec l'une des mesures de distance, telles que la distance euclidienne la plus courte à l'état cible.

Parmi certains états de départ et d'objectif, il peut y avoir un ou plusieurs chemins (trajectoires), ou il n'y a aucun chemin qui relie ces états. Il existe généralement plusieurs chemins possibles (c.à.d. des chemins qui ne heurtent pas les obstacles). Pour restreindre le choix de sélection, des exigences ou des critères supplémentaires sont introduits pour atteindre le degré d'optimalité souhaité :

- La longueur du chemin doit être la plus courte possible,
- Le chemin approprié est celui dans lequel le robot peut prendre le plus petit temps,
- Le chemin doit être éloigné des obstacles,
- Le chemin doit être lisse sans virages serrés,
- La trajectoire doit prendre en compte les contraintes de mouvement (par exemple, les contraintes non holonomiques, où à l'heure actuelle, toutes les directions de conduite ne sont pas possibles).

II.4. Configuration et espace de configuration

Le processus de configuration est l'état du système dans l'espace d'environnement réparti en n dimensions. Le vecteur d'état est décrit par $q = [q_1, \dots., q_n]^T$. L'état q est un point dans un espace à n dimensions appelé espace de configuration (*configuration space*) Q qui présente toutes les configurations possibles du système mobile suivant son modèle cinématique.

Une partie de l'espace de configuration est occupée par les obstacles dite O_i est noté $O_{Obst} = \bigcup_i O_i$. Ainsi, la partie libre de l'environnement sans les obstacles est définie par :

$$Q_{free} = Q - O_{Obst}$$

La partie Q_{free} contient un espace où le système mobile peut planifier son mouvement.

Supposons qu'il existe un robot circulaire (circular robot), qui ne peut exécuter que des mouvements de translation dans un plan (c.-à-d. qu'il a deux DOF $q = [x, y]^T$), alors sa configuration peut être définie par un point (x, y) présentant le centre du robot.

Dans ce cas, l'espace de configuration Q est déterminé en déplaçant le robot autour d'obstacles tout en le gardant en contact avec l'un de ces derniers, comme le montre la Figure II.2. Aussi, le point centre du robot décrit la frontière entre Q_{free} et O_{Obst}.

Un exemple supplémentaire d'espace de configuration pour un robot triangulaire et un obstacle rectangulaire est illustré dans la Figure II.3. Le robot ne peut se déplacer que dans les directions x et y.

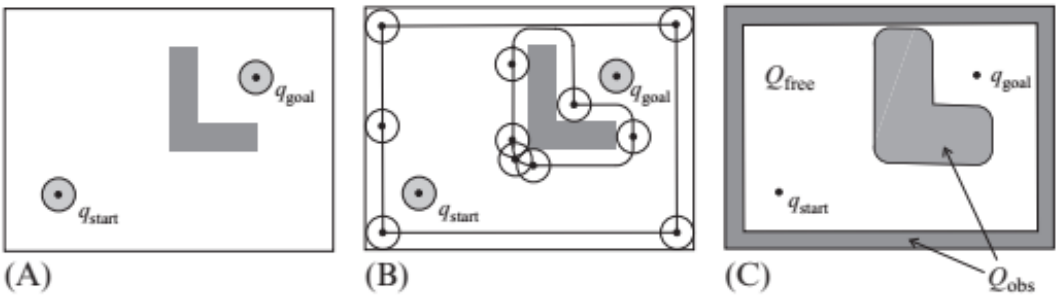


Figure II.2. (A) Robot de forme circulaire dans un espace contenant un obstacle avec des positions de débuts et d'arrivées, (B) définition de l'espace (C) environnement transformé de (A) à un espace de configuration où le robot est présenté simplement par son point centre.

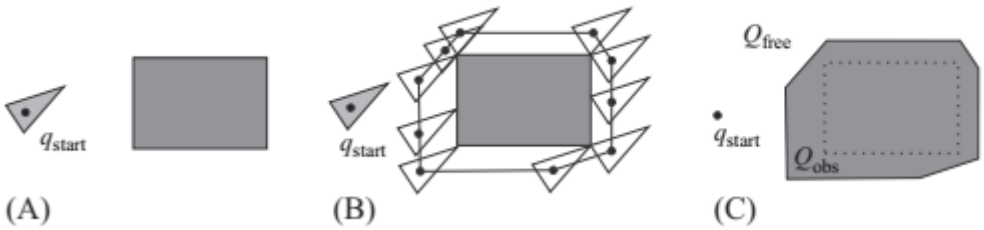


Figure II.3. (A) Obstacle rectangulaire et robot triangulaire avec un point q définissant sa configuration, (B) détermination de l'espace, (C) espace libre Q_{free}, et espace occupé par l'obstacle Q_{obst}.

Si le robot illustré sur la Figure II.3 pouvait aussi tourner, sa configuration aurait trois dimensions $q = [x, y, \phi]^T$ et l'espace de configuration serait plus complexe. Par simplification, nous pouvons déterminer l'espace de configuration en utilisant un cercle qui décrit le contour du robot, dont son centre est coïncidé avec celui du robot.

La configuration obtenue Q_{free} est alors plus petite que la vraie configuration libre car la zone du cercle délimité est plus grande que la zone du robot. Cependant, cela simplifie considérablement le problème de planification de chemin.

II.5. Description mathématique de la forme et de la position des obstacles dans un environnement

Une description mathématique de la forme et de la position d'un obstacle est indispensable pour l'estimation de l'espace de configuration du robot et des simplifications supplémentaires de

l'environnement. En effet, les deux approches les plus courantes pour décrire les obstacles sont la présentation des frontières en enregistrant les sommets et la présentation avec des demi-plans.

A. Description des obstacles basée sur les frontières

Le plan du sol de l'obstacle est un polygone à m côtés et a m sommets. La frontière d'obstacle (forme) peut être décrite par ses sommets répertoriés dans le sens anti-horaire. Les trous dans les obstacles et les murs environnants sont décrits par des sommets dans le sens des aiguilles d'une montre (Figure II.4). Une telle description d'obstacle est valable pour les polygones convexes et non convexes.

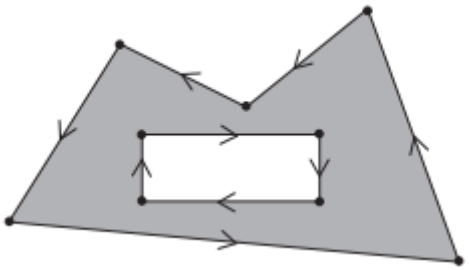


Figure II.4. Exemple de description d'un obstacle avec des sommets listés : sens antihoraire pour la limite d'obstacle externe et sens horaire pour ses trous. Le côté gauche de chaque segment de droite fléché appartient à l'obstacle (zone ombrée).

B. Description des obstacles à l'aide de demi-plans en intersection

Un polygone convexe avec m sommets peut être décrit par une l'union de m demi-plans, d'où le demi-plan est défini par $f(x, y) \leq 0$ pour les droites ou $f(x, y, z) \leq 0$ pour des plans (obstacles ont 3-dimensions). Un exemple, comment décrire un polygone à cinq sommets est illustré sur la Figure II.5. Il est à noter que, des formes non convexes ou des formes avec des trous peuvent être décrites en utilisant des opérations sur des ensembles telles que l'union, les intersections, la différence entre les ensembles, etc.

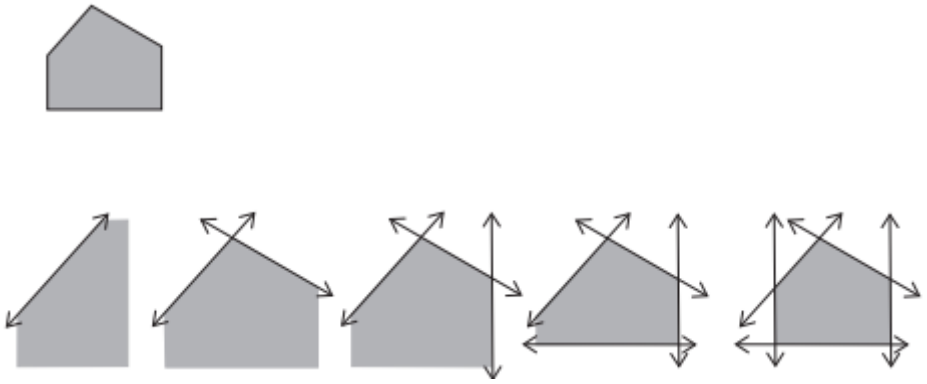


Figure II.5. Exemple de description d'un obstacle par des demi-plans.

II.6. Présentation d'un environnement exploité pour une planification d'une trajectoire

Avant de procéder à une planification d'une trajectoire, l'environnement doit être présenté d'une manière mathématique unifiée pour avoir une adaptation avec les algorithmes de recherche de chemin (trajectoire).

II.6.1. Représentation graphique

L'espace de configuration se compose généralement d'un espace libre, qui contient tous les états possibles du système mobile, et d'un espace occupé par les obstacles. Si l'espace libre est réduit et présenté par un sous-ensemble de configurations (p.ex., des centres de cellules) qui incluent des configurations de départ et d'objectif, le nombre souhaité de configurations intermédiaires et de transitions entre elles, alors le graphe de transition d'état est obtenu. Les états sont présentés par des cercles appelés nœuds, et les connexions entre eux sont données par des lignes. Les connexions (lignes) présentent donc les actions nécessaires pour que le système se déplace entre les différents états.

Dans un graphe pondéré (*weighted graph*), chaque connexion a un poids ou un coût prescrit qui est nécessaire pour l'exécution de l'action et la transition entre les différents états de cette connexion. Dans un graphe orienté, les connexions sont marquées avec des directions possibles et le coût dépend de la direction de la transition, tandis que dans un graphe non orienté, la transition est possible dans les deux sens. La Figure II.6 montre un exemple de graphe pondéré et dirigé. La recherche de chemin dans un graphe est possible avec différents algorithmes tels que le A^*, et l'algorithme dit de *Dijkstra*, etc.

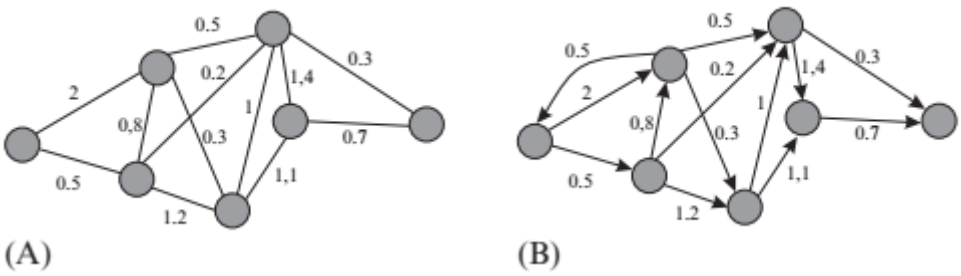


Figure II.6. Exemple de (A) graphe pondéré et (B) graphe orienté et pondéré.

II.6.2. Décomposition cellulaire

L'environnement d'un robot mobile peut être partitionné en un ensemble de cellules, qui sont généralement définies en utilisant de simples formes géométriques. Les cellules doivent être convexes puisque chaque segment de ligne droite reliant deux configurations à l'intérieur de la cellule doit se trouver entièrement dans cette cellule. Un environnement partitionné en ensemble de cellules peut

être présenté avec un graphe de transition d'état, dont ces états ont certains points à l'intérieur des cellules (p.ex., les centres) et les connexions entre les états (nœuds) ne sont possibles qu'entre les cellules voisines (p.ex., celles avec un bord commun).

A. Décomposition précise en cellules

Une décomposition en un ensemble de cellules est dite précise si toutes les cellules se trouvent entièrement dans l'espace libre ou entièrement dans l'espace occupé par les obstacles. Une décomposition précise (*Accurate decomposition*) est donc sans pertes puisque l'union de toutes les cellules libres est égale à l'espace de configuration libre Q_{free}.

Un exemple de ce genre est la décomposition verticale, elle est illustrée sur la Figure II.7. Cette décomposition peut être obtenue en utilisant une ligne verticale imaginaire allant de la bordure gauche de l'environnement à la bordure droite. Chaque fois qu'un coin d'obstacle apparaît (les sommets d'un polygone), une bordure verticale entre les cellules est créée, allant du coin uniquement vers le haut ou vers le bas, ou vers le haut et vers le bas. Il est à noter que, la complexité de cette approche dépend de la géométrie de l'environnement. Pour les environnements simples, le nombre de cellules et la connexion entre elles sont faibles ; tandis qu'avec le nombre croissant de polygones (obstacles) et de leurs sommets, le nombre de cellules augmente.

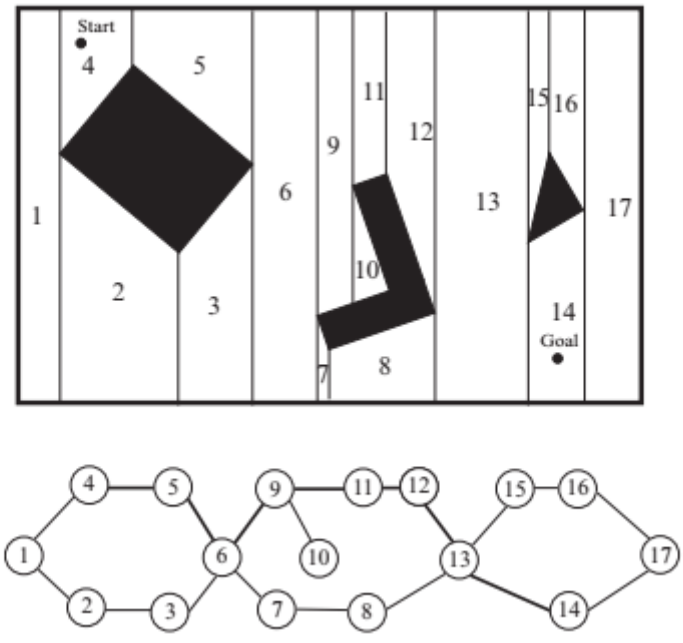


Figure II.7. Décomposition cellulaire verticale (en haut) et graphe associé (en bas) avec un chemin marqué entre le début et la configuration de l'objectif (point d'arrivé).

Une décomposition précise en un ensemble de cellules peut être présentée avec un graphe de transition d'état où les nœuds du graphe sont, p.ex. les centres des cellules et les transitions entre les cellules passent, p.ex. par les points centraux des frontières entre les cellules.

B. Décomposition approximative des cellules

Une décomposition de l'environnement en un ensemble de cellules, est dite approximative lorsqu'il est possible que, certaines cellules contiennent un espace de configuration libre ainsi qu'un obstacle ou seulement une partie d'un obstacle. Toutes les cellules contenant au moins une partie d'un obstacle sont donc normalement marquées comme occupées, tandis que les cellules restées, sont marquées comme libres. Si la décomposition en cellules est réalisée en utilisant une même taille, alors une grille d'occupation (voir Figure II.8) est obtenue.

Le centre de chaque cellule (sur la Figure II.8, la cellule libre est signée en blanc) est présenté comme un nœud dans un graphe. Les transitions entre les cellules se font dans quatre ou huit directions si les transitions diagonales sont autorisées. Cette approche est très simple pour l'application, mais en raison de la taille constante des cellules, certaines informations sur l'environnement peuvent être perdues (il ne s'agit pas d'une décomposition sans perte) ; par exemple, les obstacles sont agrandis et des passages étroits entre eux peuvent se perdre.

L'inconvénient majeur de cette approche est lié à l'utilisation de la mémoire, augmentée par agrandissement de la taille de l'environnement.

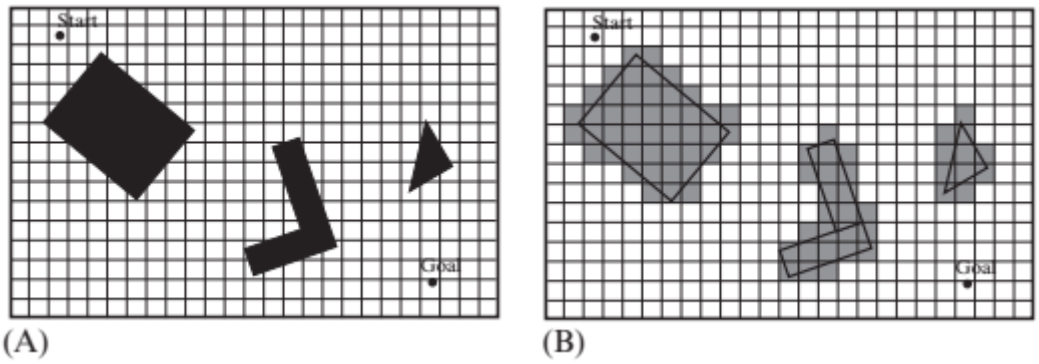


Figure II.8. Décomposition approximative en cellules obtenue en deux étapes : (A) pose d'une grille de cellules égales sur l'environnement et (B) marquage des cellules comme libres ou occupées (cellules ombrées).

En réalité, une petite perte d'informations d'environnement peut être rencontrée avec une utilisation d'une petite mémoire dans le cas où on utilise des cellules avec des tailles variables. Un exemple représentatif de ce dernier, sont les quadtrees où initialement une seule cellule est placée dans l'environnement.

Si l'entier de cette dernière est placé dans un espace libre ou si la cellule entière est occupée par un obstacle, cette dernière est restée telle quelle est. Autrement, si cette cellule n'est que partiellement occupée par un obstacle, elle est alors divisée en un ensemble de cellules plus petites. Cette procédure est répétée jusqu'à l'obtention d'une résolution souhaitée. Un exemple d'arbre quaternaire est illustré sur la Figure II.9, ainsi sa présentation peut être décrite avec un graphe d'état.

Il est à noter que, une décomposition approximative est plus simple qu'une décomposition précise ; cependant, il peut arriver qu'en raison d'une perte d'informations, un algorithme de planification de chemin (trajectoire) n'arrive pas à trouver une solution bien qu'elle est existée dans l'environnement.

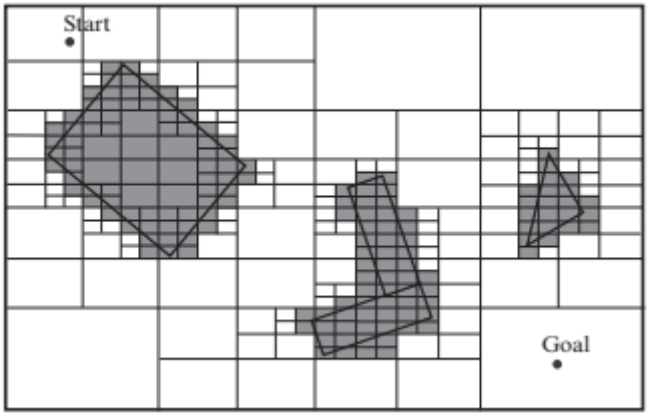


Figure II.9. Décomposition approximative en cellules de tailles variables : arbre quaternaire. Les cellules libres sont marquées en blanc et les cellules occupées sont grisées.

II.6.3. Feuille de route (*Roadmap*)

Une feuille de route (*Roadmap*), est composée généralement de lignes, de courbes et de leurs points d'intersection, elle offre des connexions possibles entre des points dans un espace libre. En effet, la tâche de planification d'un chemin (trajectoire) consiste à connecter un point de départ et un point de destination (d'arrivé) à une liaison routière existée déjà pour trouver une séquence de routes de connexion. Principalement, une feuille de route dépend de la géométrie de l'environnement. En effet, l'enjeu est de trouver un nombre minimal de routes qui permettent à un robot mobile d'accéder à n'importe quelle partie libre de l'environnement. Dans ce qui suit, deux algorithmes différents de construction de feuille de route sont présentés : la carte de visibilité (*visibility map*) et le graphe de Voronoi (*Voronoi graph*).

A. Carte de visibilité [2]

Un graphe de visibilité comprend toutes les connexions possibles entre deux sommets quelconques qui se trouvent entièrement dans un espace libre de l'environnement. Cela signifie que pour chaque

sommet, des connexions sont établies avec tous les autres sommets visibles depuis celui-ci. Le point de départ et le point d'arrivé sont considérés comme des sommets.

Des connexions sont également établies entre les sommets voisins du même polygone. Un exemple de graphe de visibilité est illustré sur la Figure II.10-A. le chemin obtenu à l'aide d'un graphe de visibilité a tendance à être le plus proche possible des obstacles ; c'est donc le chemin le plus court possible.

Les graphiques de visibilité sont simples à utiliser mais le nombre de liaisons routières augmente avec le nombre d'obstacles, ce qui peut entraîner une complexité plus élevée et donc une moindre efficacité. Un graphe de visibilité peut être simplifié en supprimant les connexions redondantes (*redundant connections*) qui peuvent être remplacées par des connexions plus courtes déjà existées.

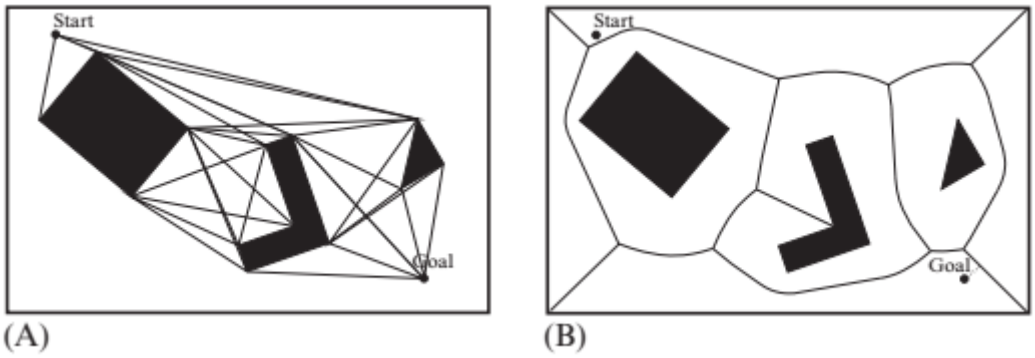


Figure II.10. Feuille de route : (A) graphe de visibilité et (B) graphe de Voronoi.

B. Graphe de Voronoi

Un graphe de Voronoi (voir Figure II.10-B), se compose d'un ensemble de sections de route qui ont la plus grande distance par rapport aux obstacles. En effet, ce graphe est défini pour un plan avec n points (p.ex., obstacle ponctuel), Il partitionne le plan en n régions dont les frontières définissent la feuille de route. Chaque région a exactement un point de génération et tout point dans une certaine région est plus proche de son point de génération que de tout autre point de génération. Un exemple de graphe général de Voronoi pour le plan d'un environnement est constitué d'obstacles de toute forme (carré, ligne droite, etc.) est illustré à sur Figure II.10. Ici, l'espace est divisé en un ensemble de régions où chaque région a exactement un obstacle générateur. Tout point dans une certaine région est plus proche de l'obstacle générateur que de tout autre obstacle. Ainsi, nous devons noter que les frontières entre les régions définissent la feuille de route.

Le déplacement sur une telle route (trajectoire), minimise énormément le risque d'une collision avec des obstacles. Dans les environnements construits à partir de polygones, la feuille de route se compose de trois courbes de Voronoi typiques, comme le montre la Figure II.11. Une courbe de Voronoi qui a une distance égale entre :

-Deux sommets est une ligne droite,

-Deux arêtes est également une ligne droite, et

-Un sommet et un segment de ligne droite est une parabole.

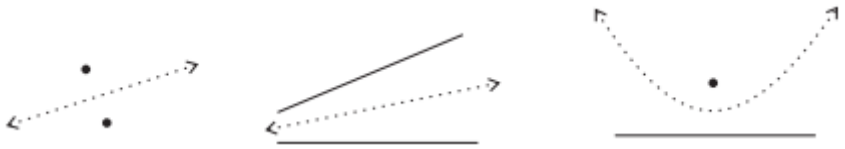


Figure II.11. Courbes typiques de Voronoï.

Comme il est déjà mentionné, cette approche maximise la distance du robot au obstacles. Cependant, la longueur de chemin obtenue est loin d'être optimale (la plus courte). Un robot équipé de capteurs de distance peut appliquer une commande pour s'éloigner de manière égale de tous les obstacles, ce qui signifie qu'il suit les routes du graphique de Voronoi. Bien que, les robots qui utilisent uniquement des capteurs tactiles ou de proximité ne puissent pas suivre les routes de Voronoi, puisqu'ils peuvent avoir des problèmes de localisation, ainsi ils peuvent facilement suivre des routes définies en utilisant le graphe de visibilité.

C. Triangulation

La triangulation est une forme de décomposition où l'environnement est divisé en un ensemble de cellules triangulaires. Bien qu'il existe de divers algorithmes de triangulation, trouver une bonne approche de triangulation qui évite les triangles étroits est un problème de recherche ouvert [3]. Un algorithme possible à utiliser est celui dit la triangulation de *Delaunay*, qui est en réalité une double présentation du graphe de Voronoi. Dans un graphe de *Delaunay*, le centre de chaque triangle (centre du cercle circonscrit) coïncide avec chaque sommet du polygone de Voronoi. Un exemple de ce dernier est illustré sur la Figure II.12.

Basant sur les approches de décomposition d'environnement précises, des algorithmes de recherche de chemin (trajectoire) peuvent être utilisés, afin de déterminer si ce dernier existe ou non. Par conséquent, ces approches sont donc complètes.

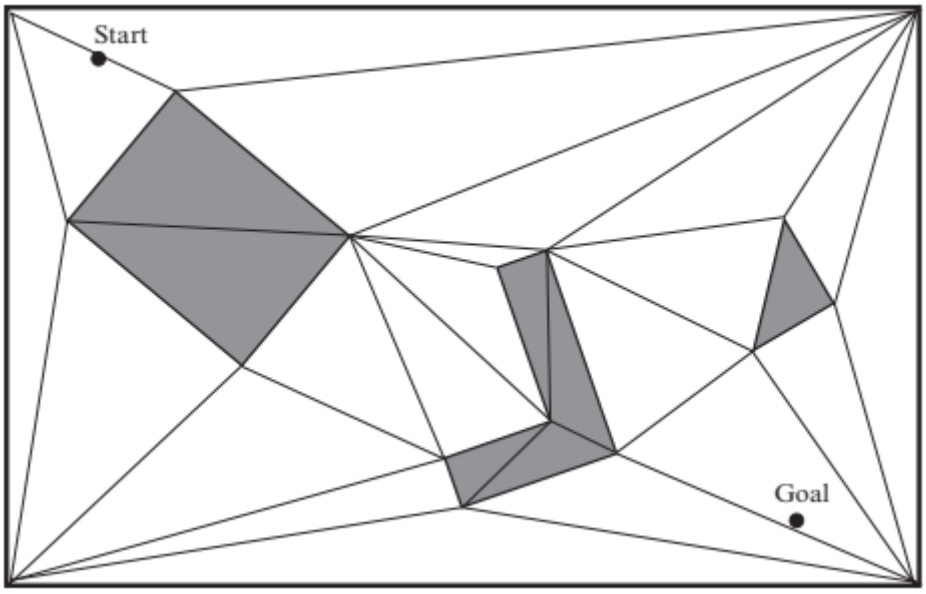


Figure II.12. Triangulation de *Delaunay* où certains bords de triangle coïncident avec les bords des obstacles.

II.6.4. Le champ potentiel [1,3]

Cette méthode permet de décrire un environnement avec un champ dit potentiel, qui peut être considéré comme une hauteur imaginaire. Notamment, le point d'arrivé est placé en bas, ainsi la hauteur accroît avec la distance vers ce dernier point et même plus élevée aux obstacles.

Cette procédure de planification, peut être vu comme un mouvement d'une balle qui se déplace vers un point de but, comme il est montré sur la Figure II.13.

Un champ potentiel est exprimé comme la somme du champ attractif dû au point d'arrivé $U_{attr}(\boldsymbol{q})$ et à un champ répulsif $U_{rep}(q)$ dû aux obstacles :

$$U(q) = U_{attr}(q) + U_{rep}(q) \tag{II.1}$$

Le point d'arrivé est le minimum global du champ potentiel. Le potentiel attractif $U_{attr}(q)$ exprimé dans (II.1) peut être défini mathématiquement comme la montre l'équation (II.2), sachant que $D(q, q_{goal})$ à la racine carrée de la distance euclidienne vers le point d'arrivé.

$$D(q, q_{goal}) = \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2}$$

$$U_{attr}(q) = k_{attr}\frac{1}{2}D^2(q, q_{goal}) \tag{II.2}$$

D'où k_{attr} est une constante positive.

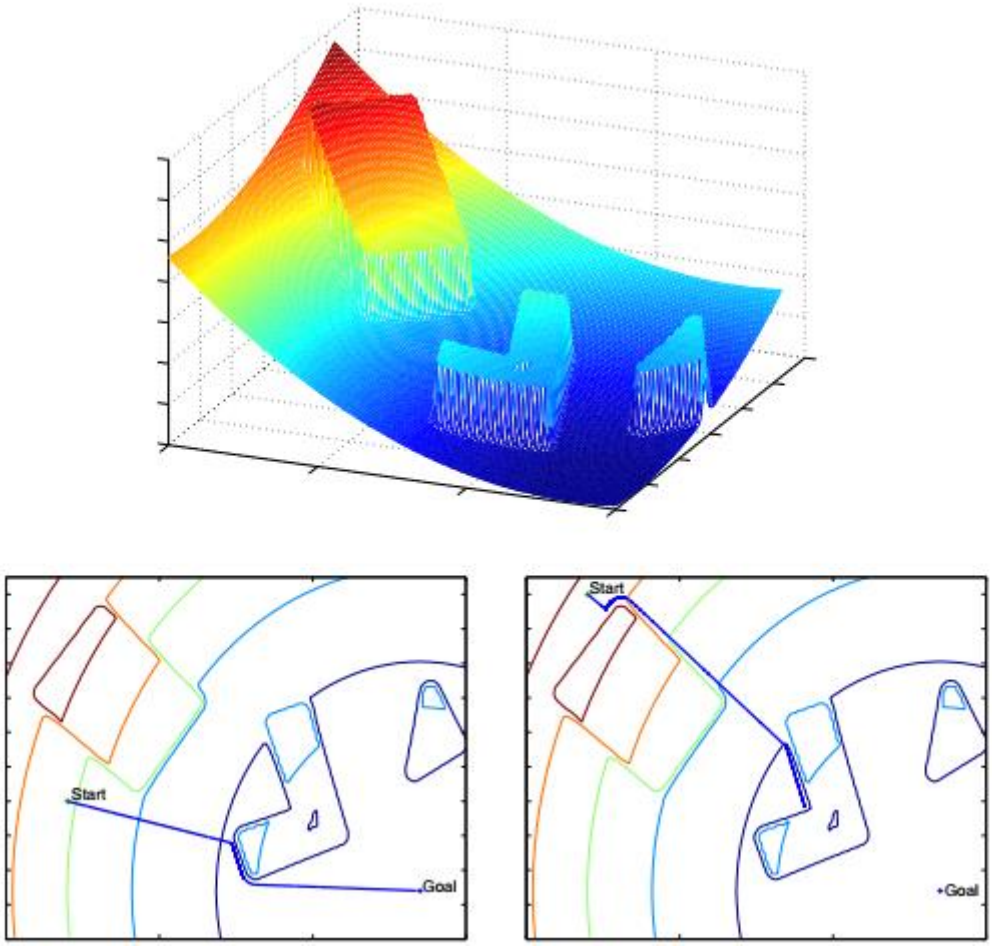


Figure II.12. Champ potentiel pour un point d'arrivé connu (en haut), des contours avec un potentiel égal et un chemin calculé pour deux points de départ (en bas), chemin calculé avec un accès réussi au but (figure gauche) et un accès non réussi (figure droite).

Le potentiel répulsif $U_{rep}(q)$ doit être très élevé à proximité des obstacles et avec un potentiel décroissant lorsque la distance des obstacles augmente $D(q, q_{obst})$. Il doit être nul lorsque $D(q, q_{obst})$ est plus élevé qu'une certaine valeur seuil D_0. Le potentiel répulsif peut être exprimé comme suit :

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{D(q,q_{obst})} - \frac{1}{D_0} \right)^2 & D(q) \leq D_0 \\ 0 & D(q) > D_0 \end{cases} \quad \text{(II.3)}$$

Où k_{rep} est une constante positive et $D(q, q_{obst})$ est la distance au point le plus proche de l'obstacle le plus proche. Pour obtenir le chemin du début au point de d'arrivé, nous devons suivre le gradient négatif du champ potentiel $(-\nabla U(q))$.

En utilisant un champ potentiel pour la présentation de l'environnement, le robot peut atteindre le but en suivant simplement le gradient négatif du champ potentiel. Le gradient négatif du champ potentiel est explicitement calculé à partir de la position connue du robot. Il est à noter que, l'inconvénient principal de cette approche est que le robot peut être piégé dans un minimum local, chose qui peut être produite si l'environnement contient un obstacle de forme concave (voir Figure II.12) ou lorsque le robot commence à osciller entre plusieurs points distancés similairement d'un l'obstacle quelconque. En effet, une tache de planification d'une trajectoire à l'aide d'un champ potentiel peut être utilisée pour calculer la trajectoire de référence que le robot doit la suivre, ou elle peut également être utilisée en ligne dans la commande de mouvement pour diriger le robot dans la direction actuelle du gradient négatif.

II.6.5. Planification d'un parcours basée sur un processus d'échantillonnage

Jusqu'à présent, les méthodes de planification présentées nécessitaient une présentation explicite de l'espace de configuration d'environnement libre. En augmentant la dimension de l'espace de configuration, ces méthodes ont tendance à prendre du temps.

Dans de tels cas, des méthodes basées sur l'échantillonnage pourraient être utilisées pour la présentation de l'environnement et la planification d'un trajet.

Avec ce genre de planification (basée sur l'échantillonnage), des points aléatoires (configurations de robot) sont échantillonnés ; puis des méthodes de détection de collision sont appliquées afin de vérifier si ces points appartiennent à l'espace libre ou non [3]. A partir d'un ensemble de ces points échantillonnés et des connexions entre eux (les connexions doivent être également dans l'espace libre), un chemin entre un point de départ connu et un point de d'arrivé souhaité est recherché.

Les méthodes basées sur l'échantillonnage ne nécessitent pas de calcul de l'espace de configuration libre Q_{free}, qui est une opération longue pour des d'obstacles de formes complexes et pour de nombreux *DOF*. Au lieu de cela, les échantillons aléatoires sont utilisés pour présenter l'espace de configuration, et indépendamment de la géométrie de l'environnement, une solution de planification de chemin peut être obtenue pour une grande variété de problèmes.

Par rapport à la proche de décomposition de l'environnement en un ensemble de cellules, les approches basées sur l'échantillonnage ne nécessitent pas un nombre élevé de cellules et des calculs gourmandes en temps, qui sont nécessaires pour obtenir une décomposition précise. En raison de l'inclusion des mécanismes stochastiques (marche aléatoire), tels que dans une planification de chemin aléatoire, la possibilité d'être piégé dans un minimum local (p.ex., comme dans la méthode du champ potentiel) est probablement faible, car le mouvement est défini entre des échantillons choisis au hasard dans l'espace libre.

Pour minimiser le temps de traitement, la détection de collision n'est vérifiée que pour des obstacles suffisamment proches et présentant un risque potentiel de collision avec le robot. De plus, le robot et les obstacles peuvent être encerclés par des formes simples de sorte qu'une détection de collision complexe (entre la vraie forme du robot et les formes d'obstacles) est effectuée uniquement lorsque ces formes englobantes avoir un chevauchement. En outre, la détection de collision peut être effectuée de manière hiérarchique en échangeant les formes les plus grandes qui entourent l'entier du robot par deux formes plus petites entourent deux moitiés du robot, comme le montre la Figure II.13.

Si l'une des deux formes chevauche l'obstacle, elle est à nouveau divisée en deux formes plus petites appropriées. Cela se repoursuit, jusqu'à ce que la collision soit confirmée ou rejetée ou jusqu'à ce que la résolution souhaitée soit atteinte.

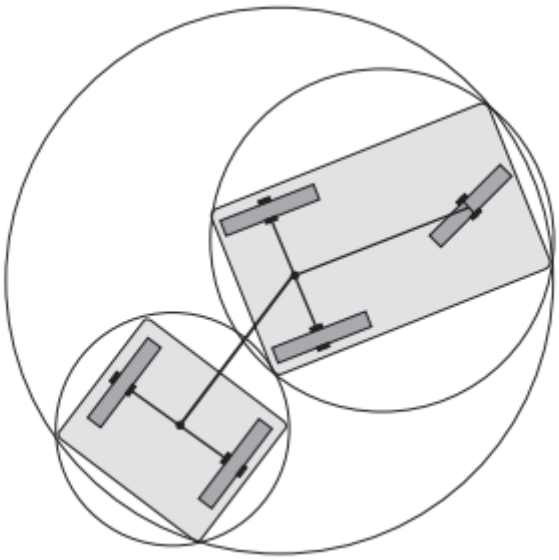


Figure II.13. Présentation d'une forme complexe d'un robot avec une forme simple englobante (cercle), qui peut être divisée en deux formes simples plus petites tout en effectuant une détection hiérarchique de la collision du robot.

Les approches basées sur l'échantillonnage, peuvent être divisées en deux classes (i) ceux qui sont appropriées pour une recherche de chemin unique et ceux qui peuvent être utilisés pour de nombreuses requêtes de recherche de chemin.

Concernant la première classe, le chemin à partir d'un point de départ unique à un seul point objectif ou d'arrivé, doit être trouvé aussi rapidement que possible ; par conséquent, les algorithmes concentrent uniquement sur les parties de l'environnement les plus prometteuses afin de trouver la solution recherchée. Les nouveaux points et connexions au graphe, sont inclus au moment de l'exécution jusqu'à ce que la solution soit trouvée.

Dans ces dernières approches, la procédure en entier de présentation de tout l'espace d'environnement libre par un graphe non orienté ou une feuille de route est effectuée en premier lieu. La solution de planification de chemin, peut être trouvée dans le graphe obtenu pour une paire arbitraire de points de départ et d'arrivé. Dans ce qui suit, nous présentons un exemple pour chaque approche.

A. La méthode RRT

Cette méthode présentée sous le nom *Rapidly exploring Random Tree* (*RRT*), est une technique de recherche de chemin à partir d'un certain point connu (de départ) à un autre point de d'arrivé [9]. A chaque itération, l'algorithme exécuté permet d'ajouter une nouvelle connexion à partir du point placé (positionné) aléatoirement au point le plus proche du graphe existant. Dès la première itération, la configuration de départ q_i présente l'arbre graphique. À chaque itération suivante, une configuration q_{rand} est choisie aléatoirement et le nœud le plus proche q_{near} par rapport au graphe existant est recherché.

Tout le long de chemin, entre q_{near} et q_{rand} avec une distance prédéfinie égale à ε, un candidat pour un nouveau nœud q_{new} est calculé. Si q_{new} ainsi que la connexion de q_{near} à q_{new} sont dans l'espace libre, alors q_{new} est sélectionné comme nouveau nœud et sa connexion à q_{near} est ajoutée au graphe. La Figure II.14, donne un aperçu graphique sur cette méthode.

Il est à noter que, le processus de recherche est terminé après un certain nombre d'itérations (p.ex. 100 itérations) ou lorsqu'une certaine probabilité est atteinte (p.ex. 10%).

Une fois les conditions d'arrêts de l'algorithme sont vérifiées, le point d'arrivé est sélectionné au lieu de prendre un nouvel échantillon aléatoire, ainsi une vérification est réalisée afin de déterminer si ce dernier (le point d'arrivé) a la possibilité d'être connecté au graphe [25]. Un tel arbre graphique s'étend rapidement à des zones inexplorées comme le montre la Figure II.15. Cette méthode n'a que deux paramètres : la taille du pas ε et le nombre d'itérations.

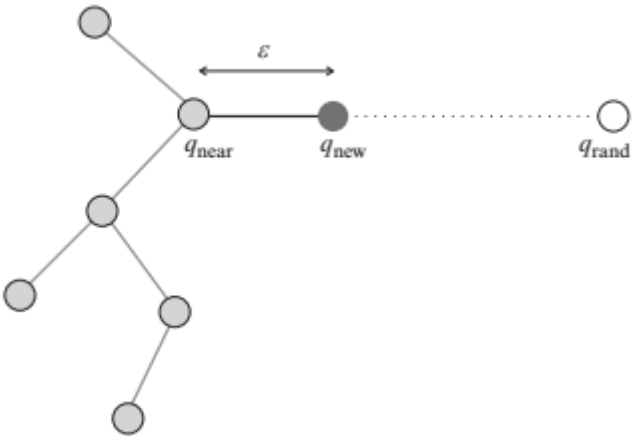


Figure II.14. Principe de la méthode *RRT*.

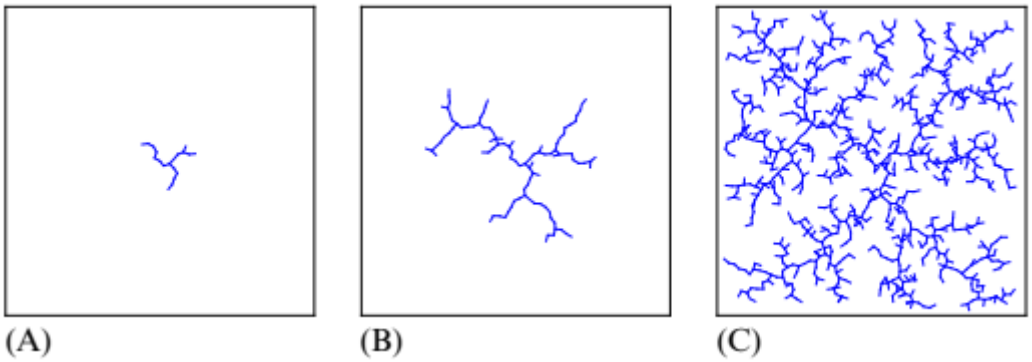


Figure II.15. L'arbre construit par la méthode *RRT* progresse rapidement vers l'espace libre inexploré. De gauche à droite, les arbres construits contiennent respectivement (A) 20, (B) 100 et (C) 1000 nœuds.

B. La méthode PRT

Cette méthode, qui porte le nom *Probabilistic Roadmap* (PRM), est plus globale que la précédente, elle offre la possibilité de faire une recherche d'un chemin entre plusieurs points de départ et plusieurs points d'arrivés [11]. En effet, l'algorithme *PRM* est basé sur deux étapes principales, la première étape concerne la phase d'apprentissage où une feuille de route ou un graphe non orienté dans l'espace libre est construit comme le montre la Figure II.16A. Concernant la seconde étape, le point de départ et le point de d'arrivé actuels sont connectés au graphe, par la suite un algorithme de recherche de chemin est utilisé pour trouver le chemin le plus optimal, comme le montre la Figure II.16B.

Dans la phase d'apprentissage, une feuille de route est construite. Cette dernière, est initialement un ensemble vide et elle est ensuite remplie de nœuds en répétant les étapes suivantes : Une configuration q_{rand} choisie au hasard dans l'espace libre est inclue dans la carte, ainsi les nœuds Q_n nécessaires pour étendre la carte sont trouvés.

Ces nœuds peuvent être trouvés en choisissant K nœuds voisins (Q_n), ou tous les nœuds voisins dont la distance à q_{rand} est plus petite que certains paramètres prédéfinis D, comme la montre la Figure II.17.

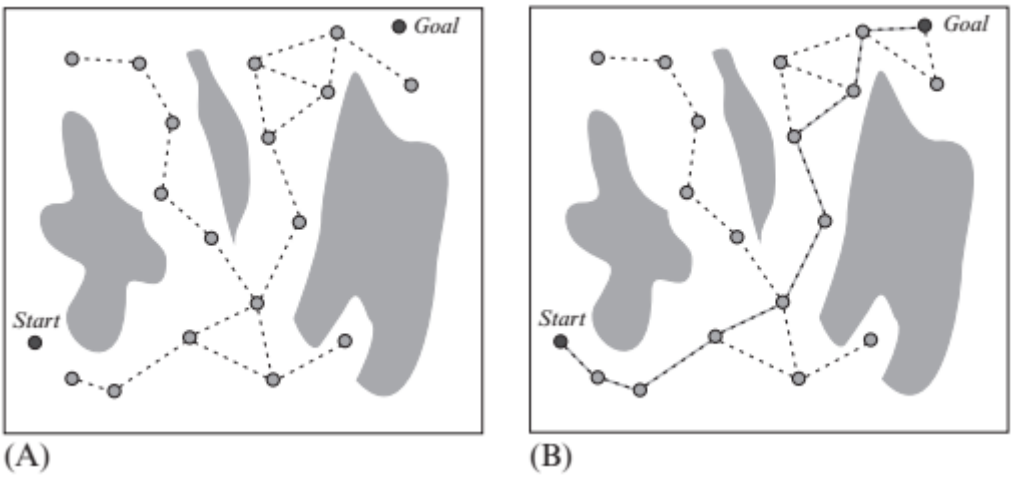


Figure II.16. Méthode *PRM* : (A) phase d'apprentissage et (B) phase de recherche de chemin.

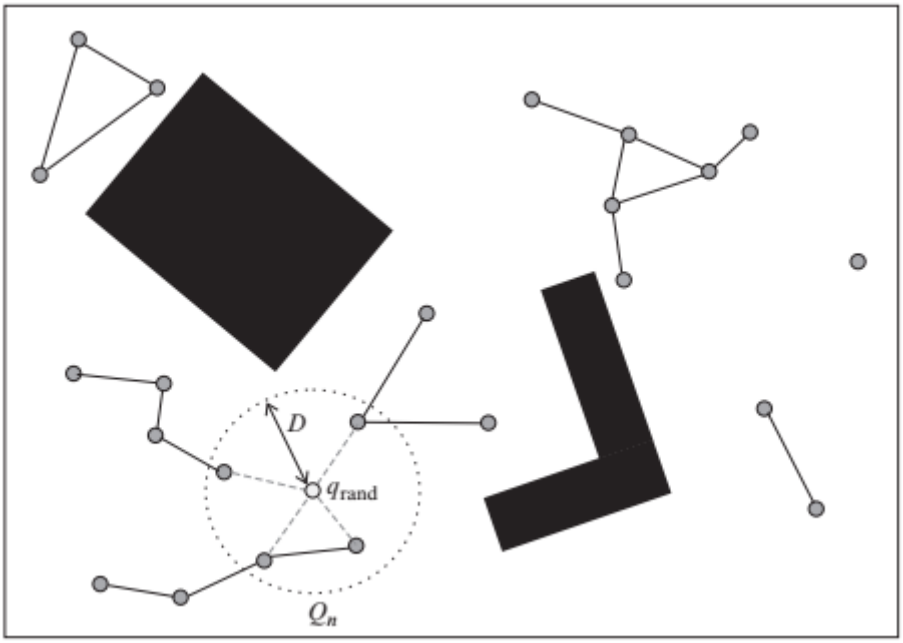


Figure II.17. Dans la méthode *PRM*, toutes les connexions possibles du point aléatoire q_{rand} aux nœuds voisins Q_n sont incluses dans la carte.

.

Il est à noter que, dans la première /ou les premières itération(s), les nœuds voisins sont probablement introuvables. Toutes les connexions simples de q_{rand} aux nœuds Q_n qui se trouvent entièrement dans l'espace libre sont inclues dans la carte. Cette procédure est répétée jusqu'à ce que la carte soit rempli avec N nœuds (nombre souhaité).

Dans la phase de recherche de chemin, les points de départ et d'arrivé souhaités sont connecté à travers l'espace libre avec les nœuds possibles les plus proches existant dans la carte. Ensuite, un

algorithme de recherche de chemin est appliqué afin de trouver le chemin connectant les deux points (début et fin).

Ces deux phases n'ont pas besoin d'être exécutées séparément, car elles peuvent être répétées de manière itérative jusqu'à ce que le nombre actuel de nœuds soit suffisamment élevé pour trouver la solution recherchée. Si la solution ne peut pas être trouvée, la carte est encore étendue avec de nouveaux nœuds et connexions jusqu'à ce que la solution soit réalisable. De cette manière, nous nous approchons de manière itérative vers la présentation la plus appropriée de l'espace libre.

Cette méthode est très efficace pour les robots avec de nombreux *DOF*. Cependant, il peut y avoir des problèmes pour trouver le chemin entre deux régions qui sont connectées à un passage étroit. Ce problème peut être résolu en ajoutant des nœuds supplémentaires à l'aide du test de pont, où trois points aléatoires proches sur une ligne droite sont sélectionnés, comme le montre la Figure II.18. Si les points les plus extérieurs sont en collision avec l'obstacle et que celui du milieu ne l'est pas, alors ce dernier est inclus en tant que nœud dans la carte.

Nous essayons de connecter ce nœud aux nœuds voisins de la même manière que les autres connexions et ajoutons ces connexions à la carte. En combinant un test de pont avec un échantillonnage uniforme dans une stratégie d'échantillonnage hybride, des cartes routières plus petites qui couvrent efficacement l'espace libre et ont toujours une bonne connectivité dans les passages étroits peuvent être obtenues.

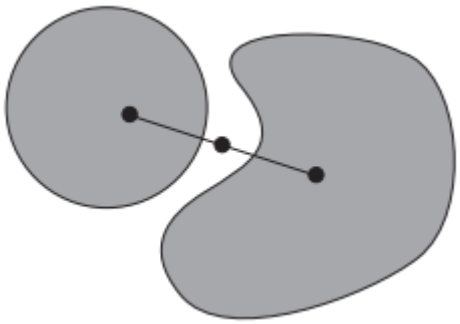


Figure II.18. Test de pont.

II.7. Conclusion

Dans ce chapitre, nous avons présenté des notions théoriques très importantes relatives à la planification d'un chemin pour un robot mobile au sens général. En effet, plonger dans ce problème (planification), implique la connaissance de quelques fondements (théories ou études), qui aident l'ingénieur ou le concepteur à prendre des outils clés afin de surmonter des difficultés ou proposer des solutions.

Comme nous avons vu dans ce chapitre, chaque méthode appliquée doit être adaptée avec un algorithme de recherche de chemin. Par conséquent, la connaissance des principes de quelques algorithmes (les plus connais) est indispensable. Dans le chapitre suivant, nous parlerons sur les plus célèbres de ces algorithmes.

Chapitre III

Algorithmes de planification d'une trajectoire pour un robot mobile (les plus célèbres)

III.1. Introduction

Ce chapitre, est consacré à la présentation de quelques algorithmes relatifs au problème de planification d'une trajectoire pour un robot mobile. Plus précisément, une illustration préliminaire d'une famille de ces derniers dite ; *Bug Algorithms* est initiée ce chapitre. Une grande partie de ce dernier (chapitre) est réservée par la suite aux autres types d'algorithmes (les plus célèbres) basé sur l'aspect graphique.

III.2. Algorithmes de Bug (*Bug Algorithms*)

Les algorithmes dits - *Bug Algorithms* - sont des algorithmes de planification de chemin, ont un caractère algorithmique simple, ils supposent uniquement des connaissances locales, ils n'ont pas besoin d'une carte d'environnement. Par conséquent, ils s'adaptent avec les situations où une carte d'environnement est inconnue ou cette dernière est évoluée (changée) rapidement et également lorsque la plate-forme mobile a une puissance de calcul très limitée.

Ces algorithmes utilisent des informations locales obtenues à partir de leurs capteurs (p. ex., un capteur de distance) et des informations d'objectifs globaux. Leur exécution consiste en deux comportements simples : (i) le mouvement en ligne droite vers le but et (ii) limites des obstacles qui doivent les suivre. Les robots mobiles basés sur ces algorithmes, ont la possibilité d'éviter tout genre d'obstacle et de se diriger vers l'objectif (point d'arrivé). Ces algorithmes nécessitent une faible utilisation d'espace mémoire, ainsi le chemin obtenu est généralement loin d'être optimal. Ce genre d'algorithmes ont été mis en œuvre la première fois par [12], plusieurs améliorations sur eux ont été proposées par la suite, voir p.ex. [13-15].

Dans ce qui suit, nous présentons trois versions de base de ce genre d'algorithmes :

III.2.1. Algorithmes de Bug0 (*Bug*0 *Algorithm*)

Un algorithme *Bug0* a deux comportements de base :

1. Déplacez vers le but jusqu'à ce qu'un obstacle soit détecté ou que l'objectif soit atteint,

2. Si un obstacle est détecté, tournez à gauche (ou à droite, mais toujours même direction) et suivez le contour de l'obstacle, jusqu'au l'accès à un mouvement sur une ligne droite vers le but est à nouveau possible.

Un exemple qui illustre graphiquement le principe de cet algorithme, est indiqué sur la Figure III.1.

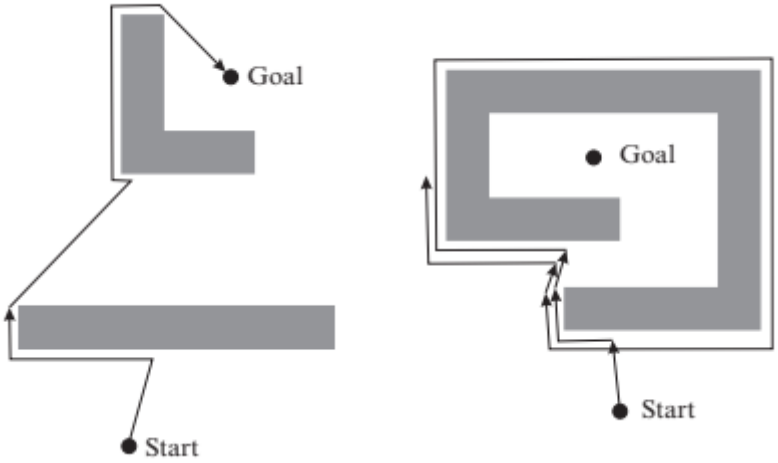


Figure III.1. L'algorithme *Bug0* trouve avec succès un chemin vers l'objectif (point d'arrivé) dans l'environnement présenté sur la figure gauche, alors qu'il échoue dans l'environnement présenté sur la figure droite.

III.2.2. Algorithmes de *Bug1* (*Bug*1 *Algorithm*)

Le *Bug1* est une autre version d'algorithme de *Bug*, par rapport à *Bug0*, il utilise plus de mémoire et il nécessite aussi des calculs supplémentaires. A chaque itération, cet algorithme doit calculer la distance euclidienne du but (point d'arrivé) et doit mémoriser le point le plus proche de la circonférence de l'obstacle par rapport au but.

Son exécution se déroule en suivant les étapes suivantes :

1. Déplacez sur une ligne droite vers le but jusqu'à ce qu'un obstacle soit atteint ou que le but soit atteint,

2. Si un obstacle est détecté, tournez à gauche et suivez tout le contour de l'obstacle et mesurez la distance euclidienne au but. Lorsque le point où l'obstacle a été initialement détecté est à nouveau atteint, suivez le contour de l'obstacle dans la direction la plus courte jusqu'au point du contour le plus proche de l'objectif. Puis reprenez le mouvement vers le but suivant une

ligne droite.

Une illustration graphique d'un algorithme *Bug1* est donnée sur la Figure III.2.

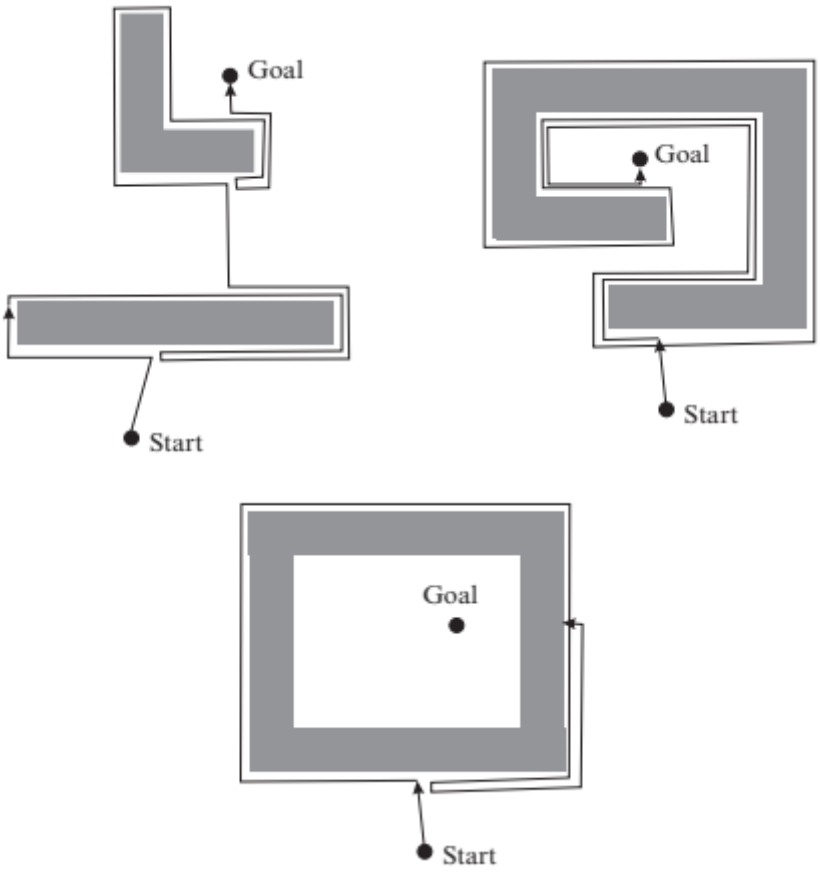


Figure III.2. Comme indiqué sur les deux figures ci-dessus, l'algorithme *Bug1* découvre le chemin vers le but s'il existe. Dans le pire des cas, sa trajectoire est égale à $\frac{3}{2}$ de la longueur de tous les contours d'obstacles, elle est plus longue que la distance euclidienne mesurée du début jusqu'au but. L'algorithme peut identifier un objectif inaccessible.

Il est à noter, en se référant à la Figure III.2, que le chemin obtenu n'est pas optimal, et il est dans le pire des cas, égale à $\frac{3}{2}$ de la longueur de tous les contours d'obstacles plus longue que la distance euclidienne du début au but. Pour chaque obstacle détecté sur la trajectoire du début à la fin, l'algorithme ne trouve qu'un seul point d'entrée et un point de départ du contour d'obstacle. Il ne détecte donc jamais l'obstacle plus d'une fois et ne tourne donc jamais sur les mêmes obstacles. Lorsque l'algorithme détecte le même obstacle plusieurs fois, il sait que le point de départ ou le point de but est capturé à l'intérieur de l'obstacle ainsi la recherche de chemin peut être terminée, car aucun chemin

possible vers le but est existé (situation montrée sur la troisième partie de la Figure III.2).

III.2.3. Algorithmes de *Bug2* (*Bug2 Algorithm*)

L'algorithme *Bug2,* exécute un déplacement sur une ligne principale définie comme une ligne droite reliant le point de départ et le point de but. Il marche (il s'exécute) en répétant les étapes suivantes :

1. Déplacez sur la ligne principale jusqu'à ce qu'un obstacle soit touché ou que le point de but soit atteint. Dans ce dernier cas, la recherche de chemin est terminée,

2. Si un obstacle est détecté, suivez le contour de l'obstacle jusqu'à ce que la ligne principale soit atteinte où la distance euclidienne au point de but est plus courte que la distance euclidienne du point dont l'obstacle a été détecté pour la première fois.

Bien que, l'algorithme *Bug2* semble en général beaucoup plus efficace que le Bug1 (voir partie gauche de la Figure III.3), il ne garantit pas que le robot détecte certains obstacles une seule fois. Dans certaines configurations d'obstacles, le robot avec le *Bug2* pourrait faire des encerclements répétés considérés inutiles jusqu'à ce qu'il atteigne le point de but, comme il est illustré sur la partie droite de la Figure III.3.

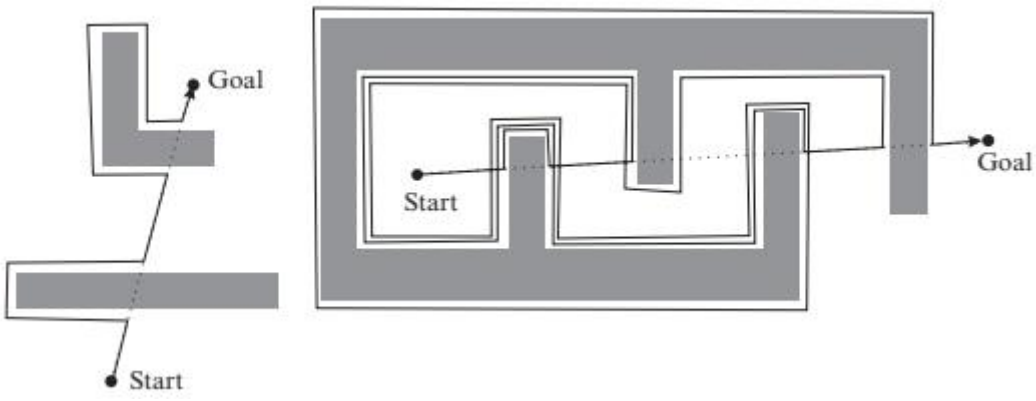


Figure III.3. L'algorithme *Bug2* suit la ligne principale jusqu'au point de but. Il peut encercler le même obstacle plusieurs fois ; par conséquent, des cercles inutiles peuvent se produire. L'algorithme *Bug2* peut identifier un objectif inaccessible.

L'algorithme peut identifier que l'objectif est inaccessible, s'il rencontre (détecte) le même obstacle au même point plus d'une fois. Une comparaison directe entre le Bug1 et le Bug2, nous permettre de conclure que :

1. Le *Bug1* est plus approfondi, puisqu'il évalue toutes les possibilités avant de prendre une décision,

2. Le *Bug2* est l'algorithme *glouton (Greedy Algorithm)*, puisqu'il sélectionne la première option qui semble prometteuse.

3. Dans la plupart des cas, l'algorithme *Bug2* est plus efficace que *Bug1*. Cependant, le fonctionnement de l'algorithme *Bug1* est plus facile à prévoir.

III.3. Méthodes de planification basées sur des graphes (*Graph-based path planning methods*)

En général, tout algorithme de recherche de chemin (trajectoire) doit passer par les étapes suivantes ; Au début, le nœud de départ doit être vérifié pour déterminer s'il s'agit d'un nœud d'arrivé (nœud objectif) ou non. Généralement ce n'est pas le cas, par conséquent, la recherche est étendue à d'autres nœuds dans le voisinage du nœud courant. Par la suite, l'un des nœuds voisins est sélectionné (la manière dont les nœuds sont sélectionnés dépend de l'algorithme utilisé et de sa fonction de coût), s'il ne s'agit pas du nœud objectif, la recherche est également étendue aux nœuds voisins de ce nouveau nœud. Cette procédure se poursuit, jusqu'à ce que la solution soit trouvée ou jusqu'à ce que tous les nœuds du graphe ont fait l'objet d'une enquête.

Lors du processus de recherche (dans un graphe), une liste de nœuds qui ont déjà été visité doit être réalisée afin d'éviter de visiter le même nœud plus d'une fois. Les nœuds qui peuvent encore étendre la zone de recherche à d'autres nœuds voisins (également appelés nœuds vivants) sont conservés dans une liste dite *ouverte* (*open list*). Les nœuds qui n'ont pas de successeurs ou qui ont déjà été vérifiés (également appelés nœuds morts) sont conservés dans une liste dite *fermée* (*closed list*).

L'évolution de la recherche dépend de la stratégie de sélection de nouveaux nœuds pour étendre la zone de recherche. Une liste de nœuds ouverts est triée selon certains critères et lorsque la recherche est étendue, le premier nœud de cette liste triée est devenu pris ; ce nœud correspond le mieux aux critères de tri utilisés (à la plus petite valeur des critères).

Au début, la liste ouverte Q ne contient que le nœud de départ. Les nœuds dans le voisinage du nœud de départ sont découverts puis sont mis dans une liste ouverte, le nœud de départ est mis par la suite dans une liste fermée. La recherche est ensuite étendue avec le premier nœud de la liste ouverte ainsi les successeurs (du nœud choisi dans la liste ouverte) sont déterminés. La liste ouverte contient alors les successeurs précédents restants (sauf le premier qui a déjà été étendu) et les successeurs actuellement déterminés du nœud choisi.

Cette procédure est illustrée à la Figure III.4, où les nœuds de la liste ouverte sont colorés en gris clair, les nœuds de la liste fermée sont colorés en noir et les nœuds non visités sont blancs. Sur la Figure III.4, on peut voir pourquoi les nœuds de la liste ouverte sont appelés nœuds feuilles (*leaf nodes*).

Les algorithmes de recherche de graphes peuvent être classés comme *informés* ou *non-informés*. Les algorithmes *non-informés* n'utilisent aucune information supplémentaire à côté de la définition du problème. Ces algorithmes cherchent systématiquement le graphe et ne distinguent pas les nœuds les

plus prometteurs des moins prometteurs. La recherche informée, également appelée recherche *heuristique*, contient des informations supplémentaires sur les nœuds. Par conséquent, ils offrent la possibilité de faire la sélection des nœuds plus prometteurs, chose qui améliore significativement la recherche de la solution finale.

III.3.1. Recherche en largeur (Breadth-First Search) [3]

La méthode de recherche dite *Breadth-First Search* [3], est appartient à la classe d'algorithmes de recherche de graphes non-informés. Il explore d'abord les nœuds peu profonds, c'est-à-dire les nœuds les plus proches du nœud de départ. Tous les nœuds accessibles en k étapes sont visités avant les nœuds qui nécessitent $k+1$ étapes (Figure III.4).

Figure III.4. L'algorithme *Breadth-First Search* explore d'abord les nœuds les plus proches. Le nœud actuellement exploré est marqué d'une flèche, les nœuds de la liste ouverte ont une couleur grise, les nœuds de la liste fermée ont une couleur noir et les nœuds non visités sont colorés en blancs.

Dans la liste ouverte Q, les nœuds sont triés selon une approche FIFO. Les nœuds nouvellement ouverts sont ajoutés à la fin de la liste Q et les nœuds pour étendre la recherche sont pris à partir du début de la liste Q.

L'algorithme est considéré complet, puisqu'il découvre une solution s'elle existe. Autrement, dans le cas d'existence de plusieurs solutions, ce dernier permet de trouver celle qui a le moins pas par rapport au nœud de départ.

En général, la solution trouvée n'est pas optimale, puisqu'il n'est pas nécessaire que toutes les transitions entre les nœuds aient le même coût. Il est à noter que, cet algorithme nécessite l'utilisation d'un espace mémoire volumineux et un long temps de calcul. Aussi, il existe une proportionnalité entre ces derniers facteurs (mémoire+ temps de calcul) et la taille du graphe.

III.3.2. Recherche en profondeur (Depth-First Search)

La technique dite *Depth-First Search* [3], est un algorithme de recherche du graphe de type non-informé, dont l'examen est d'abord étendu en profondeur. En effet, le nœud le plus éloigné du nœud

de départ est utilisé pour étendre la recherche. Le mécanisme d'examen (recherche) se poursuit en profondeur jusqu'à ce que le nœud actuel n'ait plus de successeurs.

La recherche est ensuite, poursuivie avec le nœud le plus profond suivant, dont les successeurs n'ont pas encore été explorés. Une description du principe de cet algorithme est illustrée sur la Figure III.5.

La liste des nœuds ouverts Q est une pile triée par la méthode LIFO (dernier entré, premier sorti). Les nœuds nouvellement ouverts sont ajoutés au début de la liste Q et à partir de la même fin ; également, les nœuds pour étendre la zone de recherche sont aussi pris.

Notons que, cette méthode n'est pas complète. En effet, dans le cas d'un graphe infini (avec une branche infinie), il peut être piégé dans une branche du graphe ; ou dans le cas d'une branche de boucle (boucle en profondeur de graphe finie), elle peut rester bloquée dans des cycles donnés. Pour éviter ce problème, la recherche peut être limitée à une certaine profondeur uniquement, mais la solution peut alors avoir une profondeur supérieure à la limite de profondeur maximale. Dans tous les cas, l'algorithme n'est pas optimal puisque le chemin trouvé n'est pas nécessairement le plus court.

Cette méthode utilise peu de mémoire, puisqu'elle ne stocke que le chemin du nœud de départ au nœud actuel et aux nœuds intermédiaires qui n'ont pas encore été explorés. Lorsque certains nœuds et tous leurs successeurs sont explorés, ce nœud n'a plus besoin d'être stocké dans la mémoire.

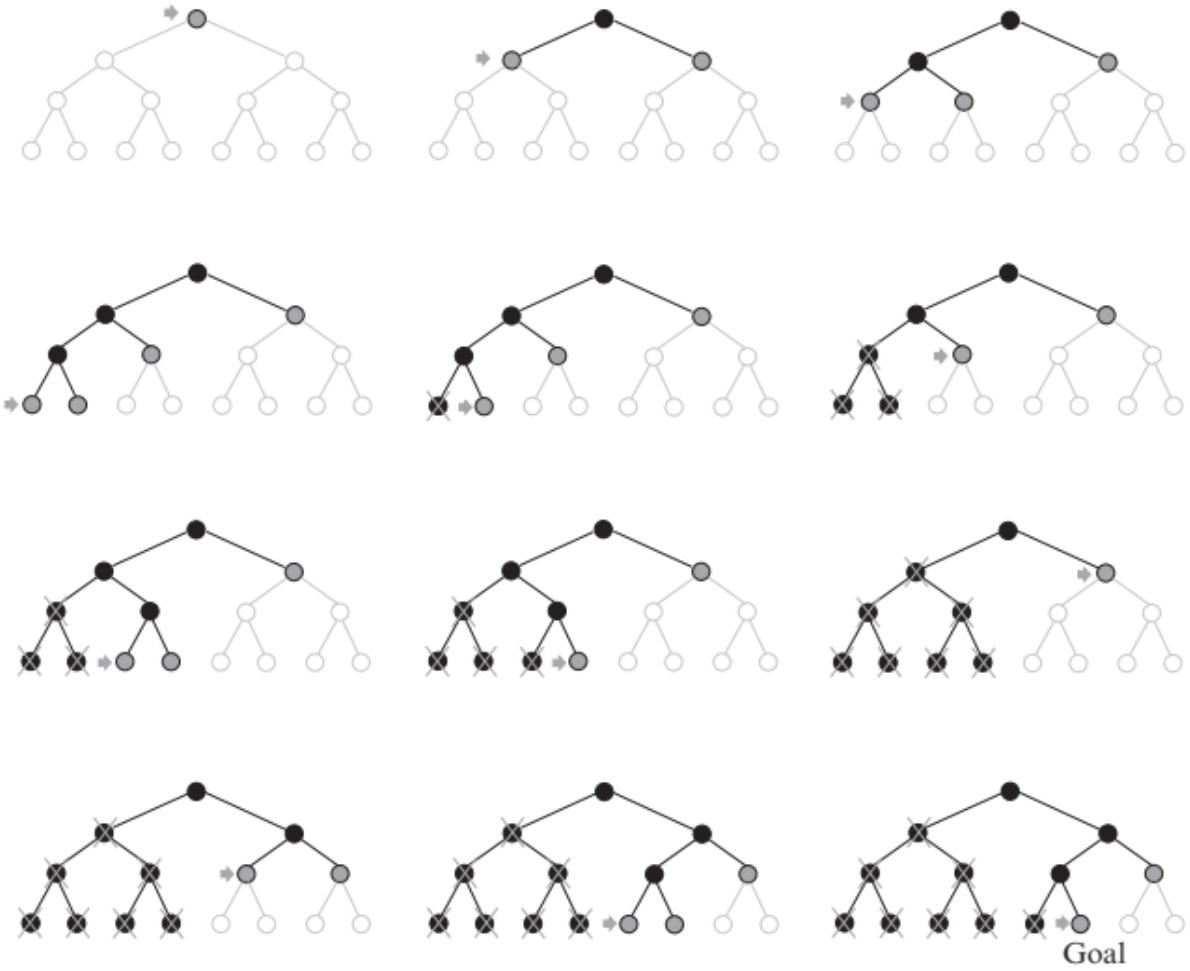


Figure III.5. L'algorithme *Depth-First Search* utilise peu de mémoire, puisqu'il n'enregistre que les nœuds feuilles (marqués en gris clair) et les nœuds étendus (gris). Les nœuds supprimés de la mémoire sont indiqués par des croix.

.

III.3.3. Recherche itérative d'approfondissement en profondeur (Iterative Deepening Depth-First Search)

Cet algorithme combine les avantages des deux algorithmes précédents *Breadth-First Search* et *Depth-First Search*. Il augmente itérativement la limite de profondeur de recherche et explore les nœuds à l'aide de la méthode *Depth-First Search*, jusqu'à ce que la solution soit trouvée (voir Figure III.6). L'exécution de cet algorithme commence par la méthode *Depth-First Search* pour les nœuds qui sont à zéro pas du nœud de départ. Si la solution n'est pas trouvée, la recherche en profondeur est répétée pour les nœuds qui sont jusqu'à un pas de celui de départ et ainsi de suite.

Cet algorithme est considéré complet (la solution est trouvée s'elle existe), et il utilise peu de mémoire, il est aussi optimal si le coût de toutes les transitions est égal ou si les coûts de transition augmentent avec la profondeur du nœud.

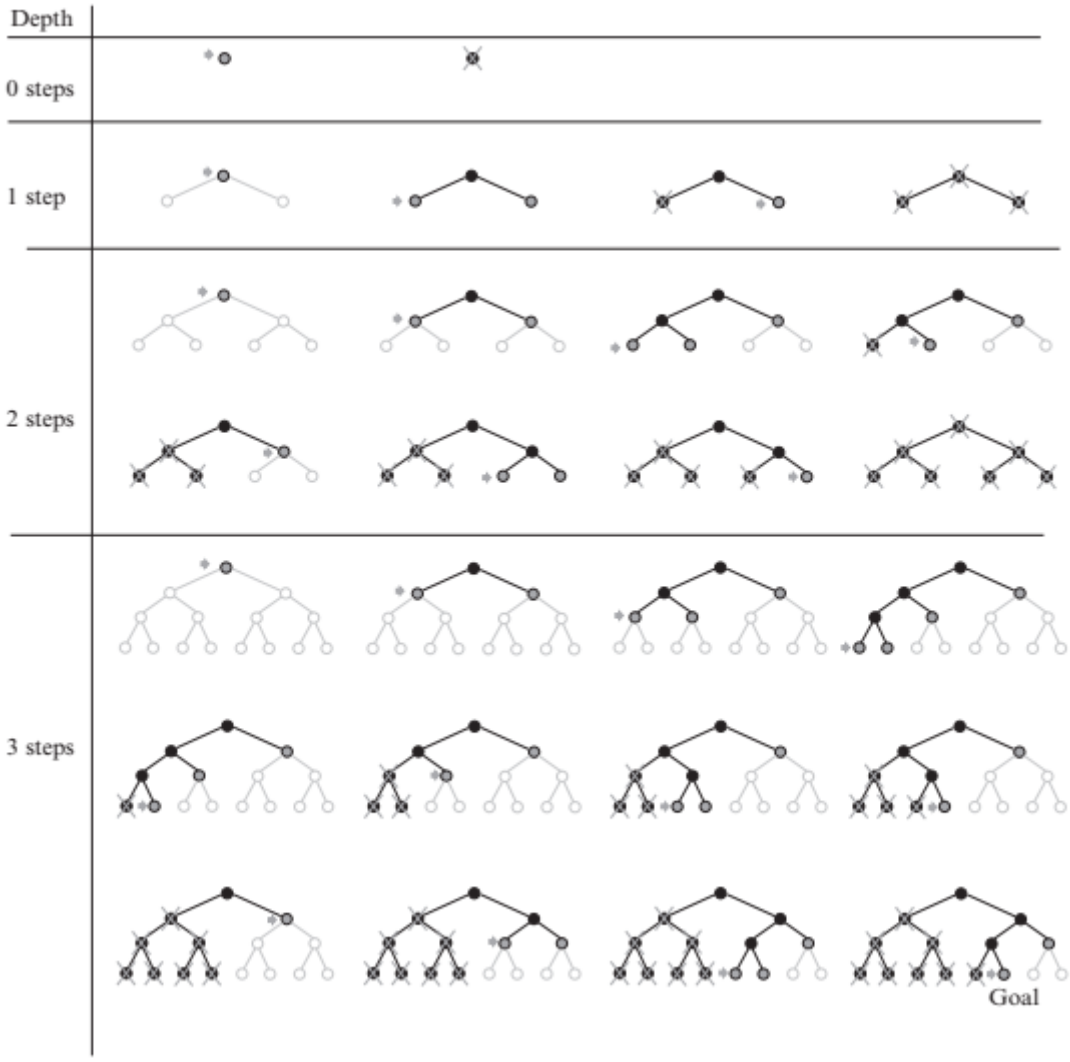


Figure III.6. Démonstration d'un algorithme ***Iterative Deepening Depth-First Search***, où la recherche est itérée à la profondeur des étapes de l'arbre. Les nœuds supprimés de la mémoire sont indiqués par des croix. L'algorithme se termine lorsque le nœud de but (d'arrivé) est trouvé (troisième étape).

III.3.4. Algorithme de Dijkstra

L'algorithme de *Dijkstra* est un algorithme non-informé, il offre la possibilité de chercher les chemins les plus courts chemins d'un nœud (nœud source) à tous les autres nœuds du graphe [3]. Le résultat est donc l'arbre de chemin le plus court du nœud source à tout autre nœud. Cet algorithme est proposé en 1959 par Edgser Wybe Dijkstra (1930-2002) [16], il a connu par la suite de nombreuses modifications [17,18]. On note que, avec les problèmes où le chemin le plus court entre un nœud de départ et un nœud d'objectif (d'arrivé) doit être trouvé, cet algorithme peut sembler inefficace en raison du calcul des chemins optimaux vers tous les autres nœuds. Dans ce cas, l'algorithme peut être arrêté dès que le chemin le plus court souhaité est trouvé.

L'algorithme découvre le chemin le plus court entre le nœud de départ (source) et le nœud de but, puisqu'il calcule la fonction de coût du chemin entre le nœud de départ et le nœud actuel, nommé en anglais *Cost-to-Here (CTH)*.

Le coût du chemin vers le nœud actuel *CTH* est calculé comme la somme du coût du chemin vers le nœud précédent (à partir duquel nous sommes arrivés au nœud actuel) et le coût de la transition entre le nœud précédent et le nœud actuel. Dans le cas de plusieurs chemins les plus courts (chemins avec le même coût), l'algorithme ne renvoie qu'un seul.

Pour exécuter l'algorithme, les transitions entre les nœuds et leurs coûts doivent être définis. Pour chaque nœud visité, l'algorithme enregistre à lui, le coût du chemin le plus court actuel (le *CTH*) et la connexion d'où nous sommes arrivés au nœud actuel. Lors de la recherche, les listes de nœuds ouverts et de nœuds fermés sont adaptées.

Dans ce qui suit, on présente les différentes étapes de l'exécution de cet algorithme : Au début, la liste ouverte ne contient que le nœud de départ, qui n'a aucun *CTH* et il est sans connexion à un nœud précédent. La liste des nœuds fermés est vide. Ensuite, les étapes suivantes (illustrées sur la Figure III.7) sont exécutées.

1. A partir de la liste ouverte, prenez le premier nœud ; appelez-le nœud actuel. La liste ouverte est triée en fonction du *CTH* dans l'ordre croissant, où le premier nœud est celui avec le plus petit coût,

2. Pour tous les nœuds accessibles à partir du nœud actuel, calculez-le *CTH* en tant que la somme du *CTH* du nœud actuel et le coût de transition,

3. Calculer et stocker le *CTH* et la connexion appropriée au nœud actuel pour tous les nœuds qui ne disposent pas déjà de ces informations stockées.

4. Si, à l'étape précédente, certains nœuds ont déjà un *CTH* et une connexion appropriée d'une itération précédente, alors comparez à la fois ces deux coûts et enregistrer le coût le plus bas ainsi son lien (connexion),

5. Les nœuds sont ajoutés à la liste ouverte et triés par ordre croissant selon le *CTH*. Une telle liste est appelée une file d'attente prioritaire et permettre au nœud avec le plus bas (price-to-here) d'être trouvé plus rapidement que dans une liste non triée. Le nœud actuel est déplacé vers la liste fermée.

À l'origine, l'algorithme de *Dijkstra* se termine lorsque la liste des nœuds ouverts est vide et le résultat sont les chemins les plus courts entre le nœud de départ et tous les autres nœuds. Si seulement le chemin le plus court depuis le nœud de départ et un nœud de but est nécessaire, alors l'algorithme se termine lorsque le nœud de but (d'arrivé) est ajouté à la liste fermée.

Le chemin résultant peut être obtenu en suivant les connexions qui nous ont amenés au nœud de but. Au nœud de but (d'arrivé), la connexion au nœud précédent est obtenue, puis la connexion à son nœud précédent est lue et ainsi de suite jusqu'à ce que le nœud de départ soit atteint. Enfin, la liste des

connexions obtenue est inversée.

Il est à noter que, l'algorithme de *Dijkstra* est un algorithme complet et optimal si tous les connexions ont des coûts supérieurs à zéro.

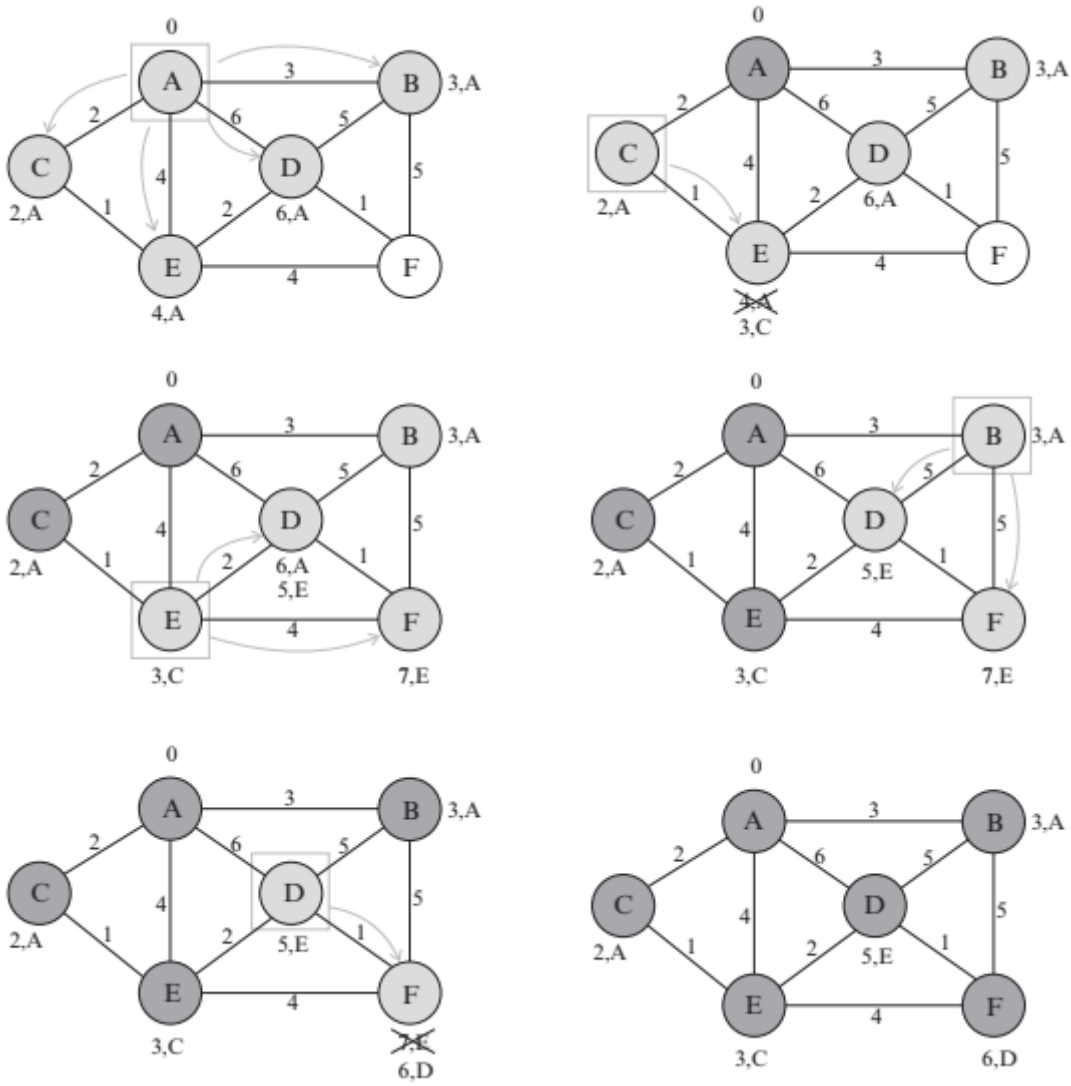


Figure III.7. L'algorithme de *Dijkstra* pour chercher les chemins les plus courts du nœud de départ *A* à tous les autres nœuds. Le nœud actuel est entouré d'un carré gris et ses successeurs sont indiqués par des flèches. Les coûts des transitions entre les nœuds sont indiqués au niveau des connexions. Le long des nœuds, le *CTH* et la connexion au nœud précédent sont donné. Les nœuds de la liste ouverte sont colorés en gris clair tandis que les nœuds de la liste fermée sont colorés en gris foncé. Exemple : si nous devons obtenir le chemin le plus court entre les nœuds *A* et *F*, son coût se lit $Cost_{F-D-E-C-A} = 6$, tandis que le chemin passe par des nœuds $A \rightarrow C \longrightarrow E \longrightarrow D \longrightarrow F$.

.

*III.3.5. Algorithme A**

Cet algorithme a été proposé pour la première fois par Peter E. Hart, Nils John Nilsson et Bertram Raphael en 1968 [19] à l'institut de recherche de Standford. Il s'agit d'une extension de l'algorithme de *Dijkstra* de 1959. L'idée de cet algorithme est d'utilisée l'heuristique pour guider la recherche. L'étoile signifié que sa recherche dans un graphe au sommet désirer fait sous forme un arbre guidée, qui a la forme d'une étoile.

Le *A** est un algorithme informé, puisqu'il inclut des informations supplémentaires ou heuristique. L'heuristique est l'estimation du coût du chemin entre le nœud actuel et l'objectif correspondant à la partie d'arbre graphique qui n'a pas encore été explorée.

Cela permet à l'algorithme de faire la distinction entre plusieurs ou moins de nœuds prometteurs, et par conséquent, il peut trouver la solution plus efficacement. Pour chaque nœud, l'algorithme calcule la fonction heuristique qui est l'estimation du coût du chemin de ce nœud à l'objectif, appelé en anglais *Cost-to-Goal (CTG)*. Cette fonction heuristique peut être la distance euclidienne ou la distance de Manhattan (somme des déplacements verticaux et horizontaux) du nœud actuel au nœud de but. L'heuristique peut également être calculée par une autre fonction appropriée.

Pendant l'exécution de l'algorithme pour chaque nœud, le coût du chemin complet (coût-du-chemin-entier/*cost-of-the-whole-path*) est calculé comme étant les deux coûts *CTH* et *CTG*. Lors de la recherche de chemin, la liste des nœuds ouverts et la liste des nœuds fermés sont également utilisés.

Dans ce qui suit, on présente aussi les différentes étapes de l'exécution de cet algorithme : Au début, la liste ouverte ne contient que le nœud de départ, qui n'a aucun *CTH* et il est sans connexion à un nœud précédent. Par la suite, les étapes suivantes (illustrées également sur la Figure III.8) sont répétées.

1. A partir de la liste ouverte, prenez le premier nœud ; appelez-le nœud actuel. La liste ouverte est triée en fonction du *(cost-of-the-whole-path)* dans un ordre croissant, où le premier nœud est celui avec le plus petit coût,

2. Pour tous les nœuds accessibles à partir du nœud actuel, calculez-le *CTG*, le *CTH* en tant que la somme du *CTH* du nœud actuel et le coût de transition, ainsi le *(cost-of-the-whole-path)* en tant que la somme de *CTH et CTG*.

3. Pour chacun de ces nœuds qui n'ont pas déjà stocké le *CTH*, les valeurs de la connexion à partir du nœud actuel, le *CTG* et le *(cost-of-the-whole-path)* sont enregistrées.

4. Si, à l'étape précédente un nœud a déjà stocké les valeurs de coût d'une itération précédente. Les deux coûts *CTH* (coût courant et coût précédent) sont comparés par la suite, ainsi celui avec une moindre valeur sera stocker, et cela avec la connexion correspondante et le *(cost-of-the-whole-path)*.

5. Les nœuds dont les valeurs de coût ont été calculées pour la première fois, sont ajoutés à la liste ouverte. Les nœuds qui ont déjà été dans la liste ouverte, et qui ont été mis-à-jour sont conservés dans

la même liste (ouverte). Les nœuds qui ont été mis-à-jour, et qui étaient dans la liste fermée sont déplacés vers la liste ouverte. La liste ouverte est triée en fonction du coût *(cost-of-the-whole-path)* dans un ordre croissant. Le nœud actuel est transféré vers la liste fermée.

Dans la première sous-figure (Figure III.8), le nœud actuel est le nœud de départ et ses nœuds successeurs (accessibles dans quatre directions : gauche, droite, haut et bas à partir du nœud actuel) sont sélectionnés. Pour tous les nœuds successeurs, le coût *CTH* est un comme ils sont à seulement un pas du nœud étoile ainsi le coût *CTG* est en réalité la distance de Manhattan (heuristique) entre le nœud successeur et le nœud objectif, qui peut également être mesurée à travers l'obstacle. La somme des deux coûts est égale au coût *(cost-of-the-whole-path)*. Les nœuds successeurs ont également une connexion marquée par une flèche (direction) à son nœud parent (nœud actuel marqué par un cercle). La liste ouverte contient maintenant ces quatre nœuds successeurs et la liste fermée contient le nœud de départ.

Concernant la deuxième sous-figure (Figure III.8), le nœud avec le coût *(cost-of-the-whole-path)* le plus bas (avec un coût égal à 6) est choisi dans la liste ouverte comme un nœud actuel. Le nœud courant n'a qu'un seul successeur (les autres cellules sont bloquées par l'obstacle ou dans la liste fermée). Le coût *CTH* pour ce nœud successeur est égal à 2 puisqu'il est à deux pas (distance de Manhattan) du nœud de départ ainsi le coût *CTG* est égal à 8. Le nœud actuel est placé dans la liste fermée et le nœud successeur dans la liste ouverte. Ainsi, regardant la troisième sous-figure, le nœud actuel devient le nœud de la liste ouverte avec le coût *(cost-of-the-whole-path)* le plus bas, qui égal à 6. Ensuite, l'algorithme continue de la même manière que dans les deux premières étapes.

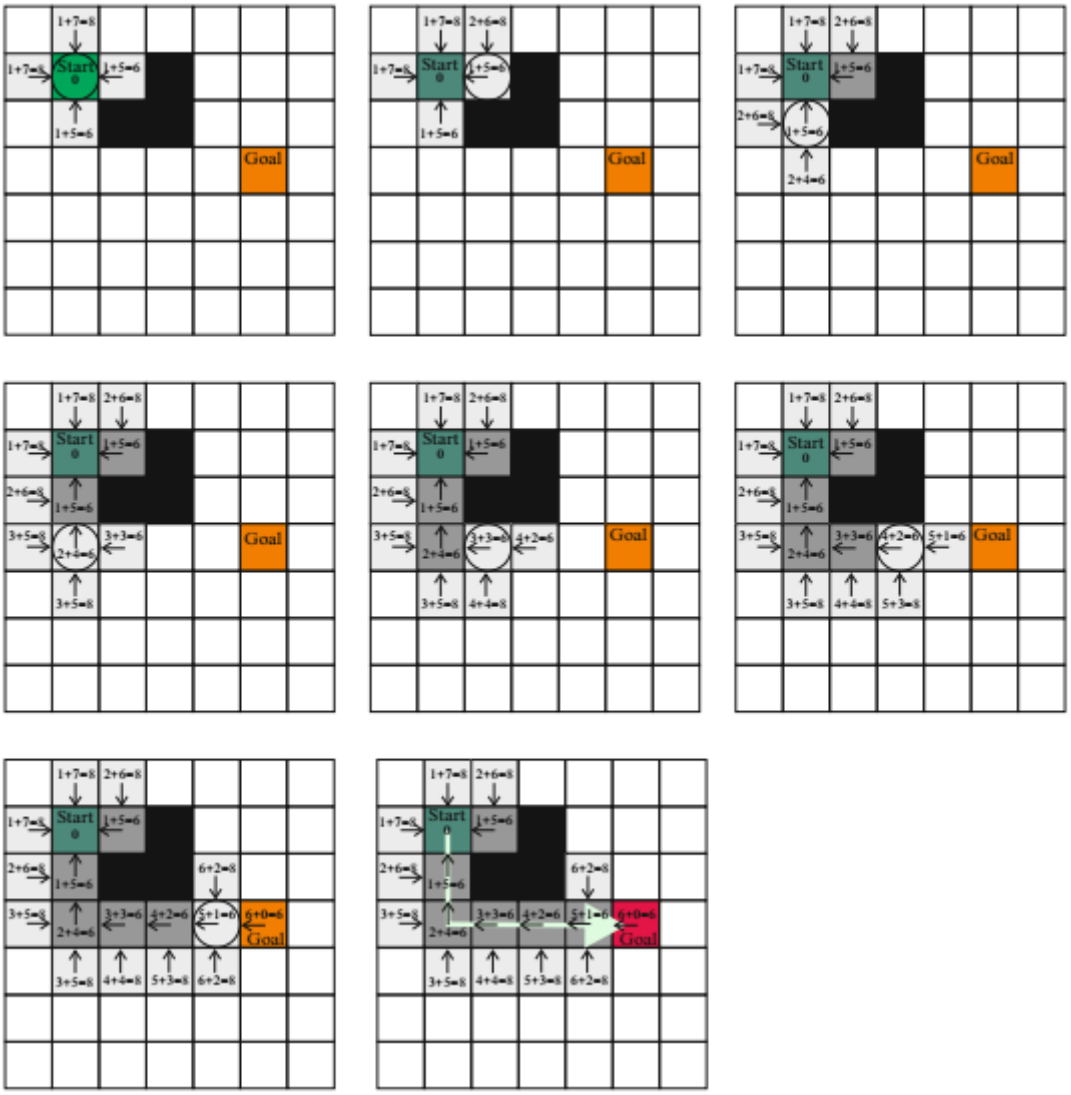


Figure III.8. Les nœuds de départ et d'objectif (d'arrivé) sont étiquetés. Le nœud actuel est marqué d'un cercle, les nœuds de la liste ouverte sont indiqués en gris clair, les nœuds de la liste fermée sont en gris foncé, tandis que les obstacles sont en noirs. Les transitions sont possibles dans quatre directions (gauche, droite, haut et bas). Dans chaque cellule visitée (nœud), la direction vers le nœud parent est indiquée par une flèche. Chaque cellule visitée contient le coût du chemin, qui est la somme du coût *CTH* et du *CTG*. Pour la fonction de coût, la distance de Manhattan est utilisée. Le chemin trouvé peut être utilisé en suivant les connexions marquées par des flèches [3].

L'algorithme A^* permet, de garantir la découverte du chemin optimal dans un graphe donné à condition si l'heuristique de calcul du coût *CTG* est admissible (ou optimisée), ce qui signifie que le coût *CTG* estimé est inférieur ou égal au coût *CTG* réel. L'algorithme se termine lorsque le nœud de but (d'arrivé) est ajouté à la liste fermée.

Il est à noter aussi que, cet algorithme est complet puisqu'il offre la possibilité de découvrir le chemin cherché s'il existe, et comme déjà mentionné, il est optimal si l'heuristique est admissible (optimisée).

Néanmoins, comme toutes les autres méthodes, ce dernier a un inconvénient, qui est sa nécessité à l'utilisation d'une grande mémoire. Si tous les coût *CTG* sont mis à zéro, l'algorithme A^* est devient équivalent à celui de *Dijkstra*.

La Figure III.9, nous montre une comparaison des performances des deux algorithmes de *Dijkstra* et A^*.

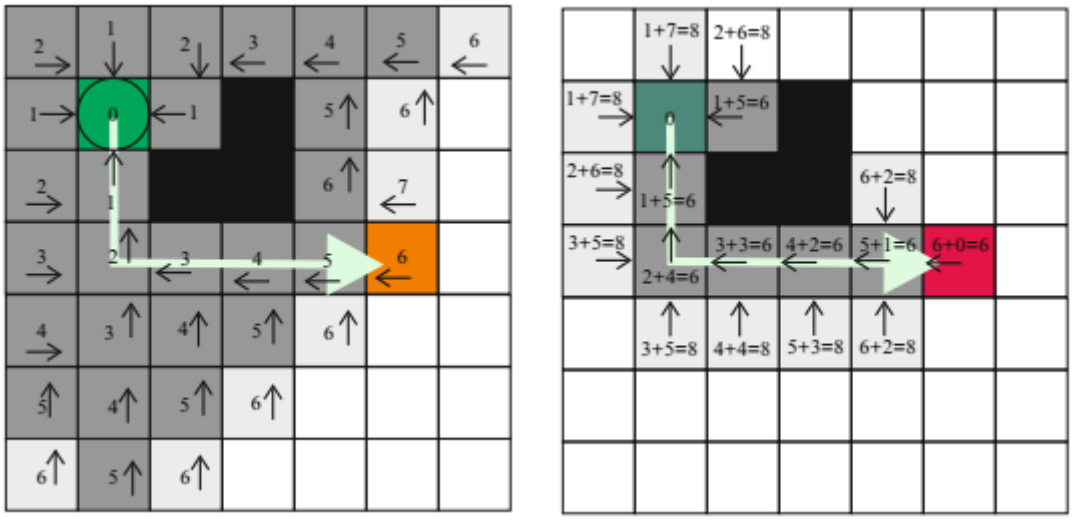


Figure III.9. Comparaison des performances des deux algorithmes de *Dijkstra* (à gauche) et celui de A^* (à droite). Les deux algorithmes trouvent le chemin le plus court vers l'objectif envisagé ; cependant, l'algorithme A^* nécessite moins d'itérations en raison de l'heuristique utilisée lors de la recherche du graphe [3].

III.3.6. Algorithme de Greedy Best-First Search

L'algorithme dit *Greedy Best-First Search* est une méthode de recherche de type informé. La liste ouverte est triée selon le coût *CTG* croissant. Par conséquent, la recherche dans chaque itération est étendue au nœud ouvert, qui est le plus proche de l'objectif (a le plus petit coût par rapport à l'objectif) en supposant qu'il atteindra l'objectif rapidement. Le chemin trouvé n'est pas une optimalité suffisante, comme le montre la Figure III.10 [3]. L'algorithme ne prend en compte que le coût du nœud actuel au but et il ignore le coût le coût nécessaire pour arriver au nœud actuel ; par conséquent, le chemin global peut devenir plus long que celui optimal.

Comme l'algorithme n'est pas optimal, il n'est pas nécessaire que l'heuristique soit admissible, comme dans le cas de l'algorithme A^*.

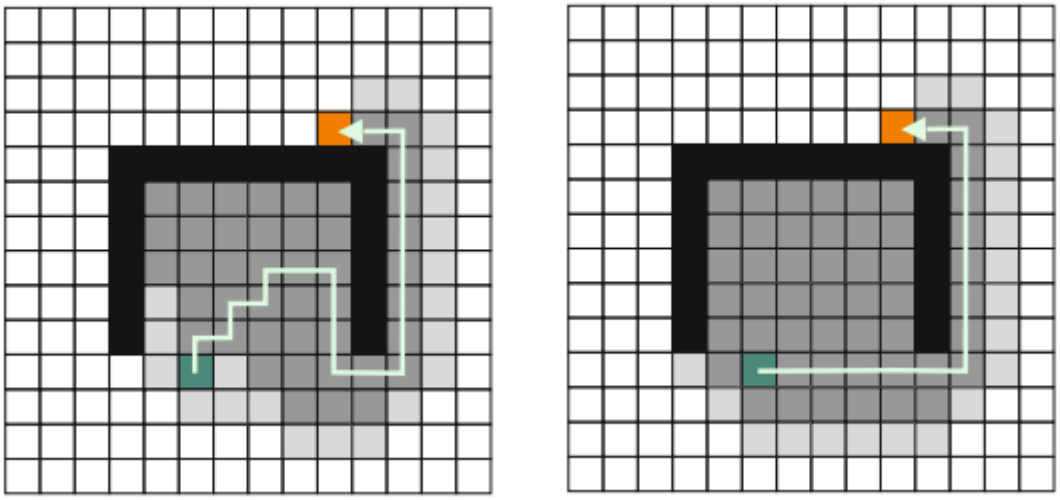


Figure III.10. Comparaison de L'algorithme dit Greedy Best-First Search (à gauche) et le A^* (à droite). Le chemin trouvé par le premier est n'est pas optimal. Dans la plupart des cas, il peut trouver la solution plus rapidement mais ce n'est pas toujours optimal. Les nœuds gris foncé sont ceux de la liste fermée, tandis que les nœuds gris clair sont ceux de la liste ouverte.

III.4. Conclusion

Un algorithme de recherche de chemin est un outil (logiciel) extrêmement important. Son intérêt réside dans le fait, qu'il présente l'esprit de la machine devant un problème de poursuite d'une trajectoire (chemin).

L'univers de ce genre de méthodes et techniques relatives surtout à la robotique mobile, est toujours en évolution, comme le montre la quantité des recherches réalisées.

Dans le chapitre suivant, nous présentons en détaille l'utilisation (toujours pour le problème de poursuite d'une trajectoire) d'un autre type d'algorithme plus sophistiqué que ceux présentés, il s'agit d'un algorithme évolutionnaire dit ; Algorithme d'Optimisation par Colonies de Fourmis (OEP) et en anglais Ant Colony Optimization (ACO) [4].

Chapitre IV :

Planification d'une trajectoire d'un robot mobile via un Algorithme d'optimisation de de colonies de fourmis (ACO)

IV.1. Introduction

Ce chapitre est consacré à la résolution du problème de planification d'une trajectoire (chemin) optimale (le plus court) relative à un robot mobile, en utilisant un algorithme dit algorithmes de colonies de fourmis (Ant Colony Optimization ACO) [4]. La méthode proposée tente d'optimiser le chemin (le plus court) suivi par un robot mobile à partir de sa position initiale à une position (finale) de destination dans un espace de mouvement modilisé sous la forme d'un environnement quadrillé a des obstacles de type statiques. Ce chapitre est doté, d'un ensemble d'essai par simulation de la méthode présentée. Ainsi, tous les résultats obtenus seront présentés et commentés.

IV.2. Les Algorithmes de Colonies de fourmis (Ant Colony Optimization ACO)

Les algorithmes de colonies de fourmis, dits en anglais Ant Colony Optimization ACO, sont une métaheuristique appartenant aux méthodes d'intelligence en essaim : elles mettent en œuvre un mécanisme de mémoire collective permettant de guider la recherche en mutualisant sur les bonnes solutions déjà trouvées. Mise au point au début des années 1990 [20,21] avec l'algorithme Ant System (AS) [22], cette méthode est issue de l'observation des fourmis réelles qui communiquent par des signaux chimiques avec les autres membres de la colonie, afin de signaler un parcours à suivre. Elles utilisent pour cela des phéromones, une substance attractive naturellement sécrétée par les fourmis, qu'elles dispersent à leur passage, notamment lorsqu'une source de nourriture a été trouvée. Ce processus d'échange d'information utilisant l'environnement ambiant est appelé *stigmergie*. La sécrétion de phéromones permet, dans le cas des fourmis, d'aboutir à des itinéraires de distance quasi-

optimale au cours du temps (entre le nid et la source de nourriture), bien que les capacités cognitives de ces insectes soient très limitées.

Comparé aux autres métaheuristiques de la même famille, C-ACO (versions classiques) présentent plusieurs caractéristiques relativement intéressantes, telles que : p.ex. leur processus de forage est basé sur une rétroaction positive (*positive feedback*) d'où la probabilité qu'une fourmi choisit une piste dans le parcoure est proportionnelle au nombre de fourmis qui ont déjà choisi la même piste, cette caractéristique permet l'accélérer la découverte de bonnes solutions ; la procédure de recherche d'une heuristique gloutonne qui est exploitée pour aider l'algorithme à trouver de bonnes solutions dans un nombre minimal d'itération. En outre, leur robustesse vis-à-vis la défiance de certain (ou un petit ensemble) des individus de la colonie et la décentralisation (les fourmis sont indépendantes dans la prise de leurs décisions).

IV.2. Optimisation par colonies de fourmis (ACO)

Une partie importante des informations présentées ci-après sont tirées du chapitre 4 : les algorithmes de colonies de fourmis, du livre : Métaheuristiques pour l'optimisation difficile [23].

IV.2.1. Origines de l'approche

Les algorithmes de colonies de fourmis forment une classe des métaheuristiques proposée pour des problèmes d'optimisation difficile. Ces algorithmes s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les fourmis) communiquent indirectement via des modifications dynamiques de leur environnement (les pistes de phéromones) et construisent ainsi une solution à un problème en s'appuyant sur leur expérience collective. Inspiré de ce phénomène naturel, Dorigo et ses collègues ont conçu une heuristique, baptisée *Ant System* (AS) [22], pour résoudre le problème dit, Problème du voyageur de commerce, en anglais (Travelling Salesman Problem TSP), mais n'a pas permis de produire des résultats compétitifs.

Cependant, l'intérêt pour la métaphore était lancé et de nombreux algorithmes s'en inspirant ont depuis été proposés, certains atteignant des résultats très convaincants.

IV.2.2. Optimisation naturelle : pistes de phéromones

Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation : les insectes sociaux en général, et les colonies de fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix

lors de l'exploitation de sources de nourritures.

Les fourmis ont la particularité d'employer pour communiquer des substances volatiles appelées phéromones. Elles sont très sensibles à ces substances, qu'elles perçoivent grâce à des récepteurs situés dans leurs antennes. Ces substances sont nombreuses et varient selon les espèces. Les fourmis peuvent déposer des phéromones au sol, grâce à une glande située dans leur abdomen, et former ainsi des pistes odorantes, qui pourront être suivies par leurs congénères (Figure IV.1).

Figure IV.1. Des fourmis suivent une piste de phéromone [http : //www.alexanderwild. com/].

Les fourmis utilisent les pistes de phéromone pour marquer leur trajet, par exemple entre le nid et une source de nourriture. Une colonie est ainsi capable de choisir (sous certaines conditions) le plus court chemin vers une source à exploiter, sans que les individus aient une vision globale du trajet.

L'expérience de l'obstacle a montré comment le dépôt de phéromone mène progressivement les fourmis à emprunter le chemin le plus court. La principale illustration de cette expérience est donnée par la Figure IV.2 : un obstacle est placé sur le trajet des fourmis qui, après une étape d'exploration, finiront par emprunter le plus court chemin entre le nid et la source de nourriture. Les fourmis qui sont retournées le plus rapidement au nid en passant par la source de nourriture sont celles qui ont emprunté le chemin le plus court. Il en découle que la quantité de phéromone déposée par unité de temps sur ce trajet est plus importante que sur les autres. Par ailleurs, une fourmi est d'autant plus attirée à un certain endroit que le taux de phéromones y est important. De ce fait, le plus court chemin a une probabilité plus importante d'être emprunté par les fourmis que les autres chemins et sera donc, à terme, emprunté par toutes les fourmis.

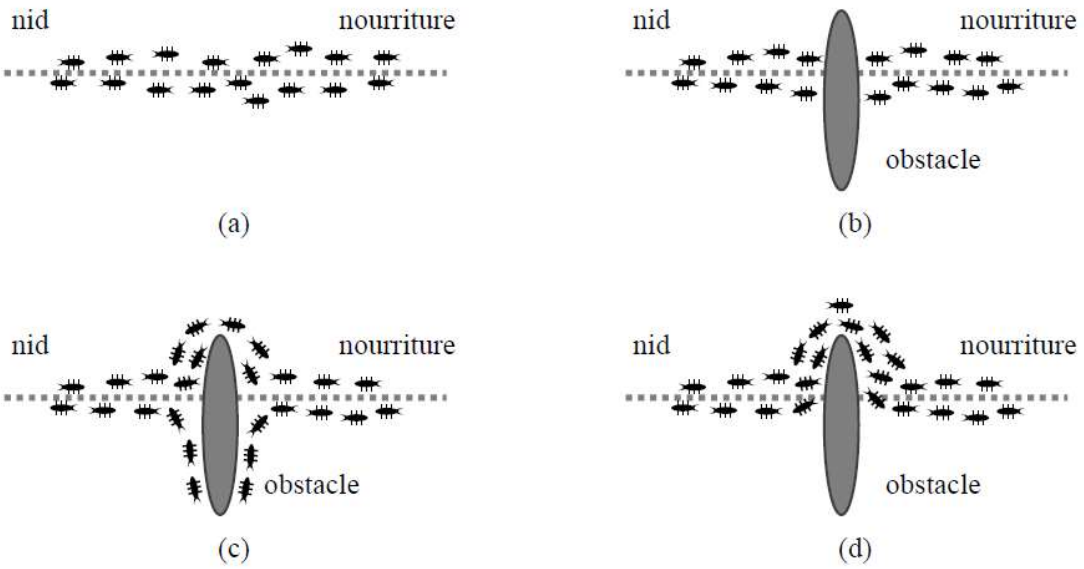


Figure IV.2. Illustration de la capacité des fourmis à chercher de la nourriture en minimisant leur parcours. (a) Recherche sans obstacle, (b) Apparition d'un obstacle, (c) Recherche du chemin optimal, (d) Chemin optimal trouvé [25].

On constate qu'ici le choix s'opère par un mécanisme d'*amplification* d'une fluctuation initiale. Cependant, il est possible qu'en cas d'une plus grande quantité de phéromone déposée sur les grandes branches, au début de l'exploitation, la colonie choisisse le plus long parcours. D'autres expérience, avec une une autre espèce de fourmis, ont montré que si les fourmis sont capables d'effectuer des demi-tours, alors la colonie est plus flixsible et le risque d'être piégé sur le chemin long est plus faible.

Il est très difficile de connaitre avec précision les propriétés physico-chimiques des pistes de phéromones, qui varient en fonction des espéces et d'un grand nombre de pramètres. Cependent, les métaheuristiques d'optimisation de colonies de fourmis s'appuient en grande partie sur le phénomène d'évaporation des pistes de phérmone. Or, on constate dans la nature que les pistes s'évaporent plus lentement que ne le prévoient les modèles. Les fourmis réelles disposent en effet 'd'heuristques' leur apportant un peu plus d'information sur le problème (par exemple une information sur la direction). En effet, il faut garder à l'esprit que l'intérêt immédiat de la colonie (trouver le plus court chemin vers une source de nourriture) peut être en concurrence avec l'intérêt *adaptatif* de tels comportements. Si l'on prend en compte l'ensemble des containtes que subit une colonie de fourmis (prédation, compétition avec d'autre colonies, etc.) un choix rapide et stable peut être meilleur, et un changement de site exploité peut entraîner des coûts trop forts pour permettre la sélection naturelle d'une option.

IV.2.3. Optimisation par colonies de fourmis et problème du voyageur de commerce

Le problème TSP a fait l'objet de la première implémentation d'un algorithme ACO : le AS [?]. Le passage de la métaphore à l'algorithme est ici relativement facile et le problème du TSP est bien connu et étudié. Il est très intéressant d'approfondir le principe de ce premier algorithme pour bien comprendre le mode de fonctionnement des algorithmes ACO. Il y a deux façons d'aborder ces algorithmes. La première, la plus évidente au premier abord, est celle qui a historiquement mené à l'algorithme AS original ; nous avons choisi de la décrire dans cette section. La seconde est une description plus formelle des mécanismes communs aux algorithmes ACO, elle sera décrite dans la section IV.2.3.2.

Le TSP consiste à trouver le trajet le plus court (désigné par "tournée" ou plus loin par "tour") reliant n villes données, chaque ville ne devant être visitée qu'une seule fois. Le problème est plus généralement défini comme un graphe complètement connecté (N, A), où les villes sont les nœuds N et les trajets entre ces villes, les arêtes A.

IV.2.3.1. Algorithme de colonie de fourmis de base, le AS

Dans l'algorithme AS, à chaque itération $(1 \leq t \leq t_{max})$, chaque fourmi k $(k = 1, \dots, m)$ parcourt le graphe et construit un trajet complet de $n = |N|$ étapes (on note $|N|$ le cardinal de l'ensemble N). Pour chaque fourmi, le trajet entre une ville i et une ville j dépend de :

1- La liste des villes déjà visitées, qui définit les mouvements possibles à chaque pas, quand la fourmi k est sur la ville i : J_i^k,

2- L'inverse de la distance entre les villes : $\eta_{ij} = \frac{1}{d_{ij}}$, appelée visibilité. Cette information "statique" est utilisée pour diriger le choix des fourmis vers des villes proches, et éviter les villes trop lointaines,

3- La quantité de phéromone déposée sur l'arête reliant les deux villes, appelée l'intensité de la piste. Ce paramètre définit l'attractivité d'une partie du trajet global et change à chaque passage d'une fourmi. C'est, en quelque sorte, une mémoire globale du système, qui évolue par apprentissage.

La règle de déplacement (appelée "règle aléatoire de transition proportionnelle" par les auteurs [?]) est la suivante :

$$p_{ij}^k(t) = \begin{cases} \frac{\left(\tau_{ij}(t)\right)^{\alpha}(\eta_{ij})^{B}}{\sum_{l \in J_i^k}(\tau_{il}(t))^{\alpha}(\eta_{il})^{B}} & si \quad j \in J_i^k \\ 0 & si \quad j \notin J_i^k \end{cases} \qquad \text{(IV.1)}$$

Où α et β sont deux paramètres contrôlant l'importance relative de l'intensité de la piste $\tau_{ij}(t)$, et de la visibilité η_{ij}.

Dans le cas où $\alpha = 0$, seule la visibilité de la ville est prise en compte ; la ville la plus proche est donc choisie à chaque pas. Au contraire, avec $\beta = 0$, seules les pistes de phéromone jouent. Pour éviter une sélection trop rapide d'un trajet, un compromis entre ces deux paramètres, jouant sur les comportements de *diversification* et d'*intensification* (voir section IV.2.3.2 de ce chapitre), est nécessaire.

Après un tour complet, chaque fourmi laisse une certaine quantité de phéromone $\Delta\tau_{ij}^{k}(t)$ sur l'ensemble de son parcours, quantité qui dépend de la qualité de la solution trouvée :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L^{k}(t)} & si \quad (i,j) \in T^{k}(t) \\ 0 & si \quad (i,j) \notin T^{k}(t) \end{cases} \tag{IV.2}$$

Où $T^{k}(t)$ est le trajet effectué par la fourmi k à l'itération t, avec $L^{k}(t)$ est la longueur de la tournée et Q un paramètre fixé.

L'algorithme ne serait pas complet sans le processus d'évaporation des pistes de phéromone. En effet, pour éviter d'être piégé dans des solutions sous-optimales, il est nécessaire de permettre au système "d'oublier" les mauvaises solutions. On contrebalance donc l'additivité des pistes par une décroissance constante des valeurs des arêtes à chaque itération. La règle de mise-à-jour des pistes est donc :

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t+1) + \sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t) \tag{IV.3}$$

Où m est le nombre de fourmis et ρ le taux d'évaporation. La quantité initiale de phéromone sur les arêtes est une distribution uniforme d'une petite quantité $\tau_0 \geq 0$.

La Figure IV.3 présente un exemple simplifie de problème TSP optimisé par un algorithme AS dont le pseudo-code est présenté sur l'Algorithme IV.1.

Algorithme IV.1 : Algorithme AS

Pour $\boldsymbol{t = 1, \dots, t_{max}}$
  Pour chaque fourmi $\boldsymbol{k = 1, \dots, m}$
    Choisir une ville au hasard
    Pour chaque ville non visité $\boldsymbol{i}$
      Choisir une ville $\boldsymbol{j}$, dans la liste $\boldsymbol{J_i^k}$ des villes restantes, selon la formule (IV.1)
    Fin Pour
    Déposer une piste $\boldsymbol{\Delta\tau_{ij}^k(t)}$ sur le trajet $\boldsymbol{T^k(t)}$ conformément à l'équation (IV.2)
  Fin Pour
  Evaporer les pistes selon la formule (IV.3)
Fin Pour

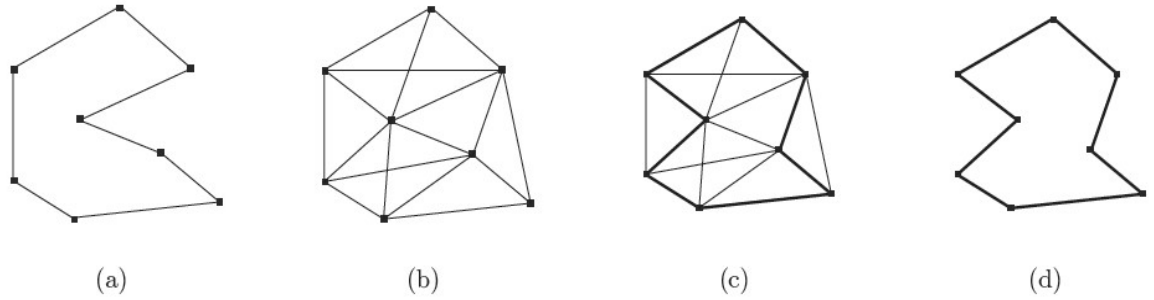


Figure IV.3. Le problème TSP optimisé par l'algorithme AS, les points représentent les **villes** et l'épaisseur des **arêtes** la quantité de phéromone déposée. (a) exemple de trajet construit par une fourmi, (b) au début du calcul, tous les chemins sont explorés, (c) le chemin le plus court est plus renforcé que les autres, (d) l'évaporation permet d'éliminer les autres chemins les moins renforcé [?].

IV.2.3.2. Principaux schémas ACO pour le TSP

– **Ant System & élitisme (*Elitist Ant*)** Une première variante du "Système de Fourmis" à été proposée dans [4] : elle est caractérisée par l'introduction de fourmis "élitistes". Dans cette version, la meilleure fourmi (celle qui a effectué le trajet le plus court) dépose une quantité de phéromone plus grande, dans l'optique d'accroitre la probabilité des autres fourmis d'explorer la solution la plus prometteuse.

– ***Rank-Based Ant System* (*RBAS*)** [26], qui s'appuie sur *Elitist Ant* et propose pour sa part de classer les solutions de chaque cycle par ordre de qualité, pour ne renforcer que les ω premières (ω étant une variable paramétrable) avec une pondération relative au rang de la solution,

– ***Best-Worst Ant System* (*BWAS*)** [27], où seules les plus mauvaises solutions sont évaporées, tandis que les meilleures solutions locales sont renforcées.

– *MAX-MIN Ant System* (*MMAS*)

Cet algorithme est l'un des successeurs de AS les plus performant. Proposé dans [28,29], il introduit quatre innovations par rapport à AS qui sont les suivantes :

1. L'exploitation intense des meilleures solutions trouvées. Après chaque itération, les traces de phéromones sont mises-à-jour uniquement en utilisant soit la meilleure solution trouvée lors de la dernière itération soit la meilleure solution trouvée depuis le début de l'algorithme,
2. Les valeurs de phéromone possibles sur les arêtes sont restreintes à être comprises dans l'intervalle $[\tau_{min}, \tau_{max}]$ avec τ_{min} et τ_{max} sont deux paramètres de l'algorithme. Le but de restreindre les valeurs des phéromones est d'éviter la stagnation prématurée de la recherche,
3. Les valeurs initiales des phéromones τ_0 est fixée à τ_{max},
4. À chaque fois que l'algorithme approche à un état de stagnation (les fourmis construisent les mêmes solutions) ou si la meilleure affectation trouvée depuis le début de l'algorithme n'a pas été améliorée depuis un certain nombre d'itérations, l'algorithme réinitialise toutes les traces de phéromones à τ_{max} (c.-à-d. : il fait un "restart"),

Les expérimentations de cet algorithme sur le TSP ont montrées que pour les petites instances, il vaudrait mieux utiliser uniquement la meilleure solution de la dernière itération pour la mise-à-jour des phéromones, tandis que pour de plus grandes instances, il serait intéressant d'altérer entre la meilleure solution de la dernière itération et la meilleure solution trouvée depuis le début de l'algorithme.

– *Ant Colony* System (*ACS*)

Dans le but d'améliorer les performances de AS sur des problèmes de grandes tailles, Dorigo et Gambardella ont proposé dans [30] une variante appelée Ant Colony System (ACS). Appliquée sur le TSP, cette nouvelle variante est essentiellement différente de AS sur les trois points suivants :

1. Une fourmi k se situant sur le nœud i, choisit le prochain noeud j en utilisant une loi proportionnelle pseudo-aléatoire formulée comme suit :

$$j = \begin{cases} argmax_{u \in J_j^k} \left[\tau_{iu}(t) \cdot \left(\eta_{iJ}\right)^B \right] & si \quad q \leq q_0 \\ J & si \quad q > q_0 \end{cases} \qquad \text{(IV.4)}$$

où q est une variable aléatoire uniformément distribuée dans l'intervalle $[0,1]$, q_0 est un paramètre tel que $0 \leq q_0 \leq 1$, J est une variable aléatoire sélectionnée selon la probabilité :

$$p_{ij}^k(t) = \frac{\left(\tau_{ij}(t)\right)^\alpha (\eta_{ij})^B}{\sum_{l \in J_i^k} (\tau_{il}(t))^\alpha (\eta_{il})^B} \qquad si \quad j \in J_i^k \qquad \text{(IV.5)}$$

La valeur de paramètre q_0 est très importante, car lorsque $q \leq q_0$, l'algorithme exploite au maximum les informations apprises (c.-à-d. : l'algorithme fait de l'exploitation en cherchant des solutions autour des meilleures solutions déjà trouvées) et lorsque $q > q_0$, l'algorithme utilise la règle standard (2.1) lui permet de faire un choix probabiliste et donc il explore l'espace de recherche.

2. La gestion des pistes est séparée en deux niveaux : une mise-à-jour locale et une mise-à-jour globale. Chaque fourmi dépose une piste lors de la mise-à-jour locale :

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0 \qquad \text{(IV.6)}$$

Où τ_0 est la valeur initiale de la piste. À chaque passage, les arêtes visitées voient leur quantité de phéromone diminuer, ce qui favorise la diversification par la prise en compte des trajets non explorés. À chaque itération, la mise-à-jour globale s'effectue comme suit :

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}^{k}(t) \qquad \text{(IV.7)}$$

Où, les arêtes (i,j) appartiennent au meilleur tour T^{best} de longueur L^{best} avec $\Delta\tau_{ij}^{k}(t) = \frac{1}{L^{best}}$. Ici, seule la meilleure piste est donc mise-à-jour, ce qui participe à une intensification par sélection de la meilleure solution.

3. Le système utilise une liste de candidats. Cette liste stocke pour chaque ville les v plus proches voisines, classées par distances croissantes. Une fourmi ne prendra en compte une arête vers une ville en dehors de la liste que si celle-ci a déjà été explorée. Concrètement, si toutes les arêtes ont déjà été visitées dans la liste de candidats, le choix se fera en fonction de la règle (IV.5), sinon c'est la plus proche des villes non visitées qui sera choisie.

IV.2.3.3. Choix des paramètres

Le choix des paramètres de réglage des différentes variantes ACO citées préalablement, a une importance capitale sur les résultats obtenus. D'après [23], pour l'algorithme AS, les auteurs préconisent que, bien que la valeur de Q ait peu d'influence sur le résultat final, cette valeur soit du même ordre de grandeur qu'une estimation de la longueur du meilleur trajet trouvé. D'autre part, la ville de départ de chaque fourmi est typiquement choisie par un tirage aléatoire uniforme, aucune influence significative du placement de départ n'ayant pu être démontrée.

En ce qui concerne l'algorithme ACS, les auteurs conseillent d'utiliser $\tau_0 = (n \cdot L_{nn})^{-1}$, où n est le nombre de villes et L_{nn} la longueur d'un tour trouvé par la méthode du plus proche voisin. Le nombre

de fourmis m est un paramètre important ; les auteurs suggèrent d'utiliser autant de fourmis que de villes $(i.e. m = n)$ pour de bonnes performances sur le problème TSP. Il est possible de n'utiliser qu'une seule fourmi, mais l'effet d'amplification des longueurs différentes est alors perdu, de même que le parallélisme naturel de l'algorithme, ce qui peut s'avérer néfaste pour certains problèmes. En règle générale, les algorithmes ACO semblent assez peu sensibles à un réglage fin du nombre de fourmis.

IV.2.4. Formalisation et propriétés d'un algorithme ACO

Un extrait très clair sur les ACO, a été écrit par [31], il présente une petite introduction descriptive aux apports originaux des métaheuristiques ACO, son contenu est le suivant :

> *Une métaheuristique de colonie de fourmis est un processus stochastique construisant une solution, en ajoutant des composants aux solutions partielles. Ce processus prend en compte (i) une heuristique sur l'instance du problème (ii) des pistes de phéromone changeant dynamiquement pour refléter l'expérience acquise par les agents.*

Une formalisation très précise à été présentée par [23]. Elle passe par une représentation du problème, un comportement de base des fourmis et une organisation générale de la métaheuristique. Plusieurs concepts sont également à mettre en valeur pour comprendre les principes de ces algorithmes, notamment la définition des pistes de phéromone en tant que mémoire adaptative, la nécessité d'un réglage *intensification/diversification* et enfin l'utilisation d'une *recherche locale*. En se référant à [?], nous traitons ci-après ces différents sujets.

IV.2.4.1. Formalisation

Représentation du problème Selon [23], un problème d'optimisation traité par un algorithme ACO est représenté généralement par un *jeu de solutions*, une fonction objectif assignant une valeur à chaque solution et un *jeu de contraintes*. L'objectif est de trouver l'optimum global de la fonction objectif satisfaisant les contraintes. Les différents états du problème sont caractérisés comme une séquence de composants. On peut noter que, dans certains cas, un coût peut être associé à des états autres que des solutions. Dans cette représentation, les fourmis construisent des solutions en se déplaçant sur un graphe $G = (C, L)$, où les nœuds sont les composants de C et où l'ensemble L connecte les composants de C. 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).

Comportement des fourmis Les fourmis artificielles peuvent être considérées comme une procédure de construction stochastique *construisant* des solutions sur le graphe $G = (C, L)$. En général, les fourmis tentent d'élaborer des solutions faisables mais, si nécessaire, elles peuvent produire des solutions infaisables. Les composants et les connexions peuvent être associés à des pistes de phéromone τ (mettant en place une mémoire adaptative décrivant l'état du système) et à une valeur heuristique (représentant une information à priori sur le problème, ou venant d'une source autre que celle des fourmis ; c'est bien souvent le coût de l'état en cours). Les pistes de phéromone et la valeur de l'heuristique peuvent être associées soit aux composants, soit aux connexions (Figure IV.4).

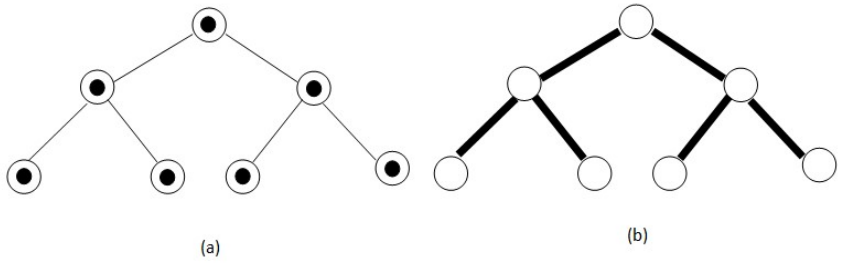


Figure IV.4. Dans un algorithme ACO, les pistes de phéromone peuvent être associées aux composants (a) ou aux connexions (b) du graphe représentant le problème à résoudre [23].

Chaque fourmi dispose d'une mémoire utilisée pour stocker le trajet effectué, d'un état initial et de conditions d'arrêt. Les fourmis se déplacent en utilisant une règle de décision probabiliste fonction des pistes de phéromone locales, de l'état de la fourmi et des contraintes du problème. Lors de l'ajout d'un composant à la solution en cours, les fourmis peuvent mettre à jour la piste associée au composant ou à la connexion correspondante. Une fois la solution construite, elles peuvent mettre à jour la piste de phéromone des composants ou des connexions utilisées. Enfin, une fourmi dispose au minimum de la capacité de construire une solution du problème.

Organisation de la métaheuristique En plus des règles régissant le comportement des fourmis, un autre processus majeur a cours : l'*évaporation* des pistes de phéromone. En effet, à chaque itération, la valeur des pistes de phéromone est *diminuée*. Le but de cette diminution est d'éviter une convergence rapide et le piégeage de l'algorithme dans des minimums locaux, par une forme d'oubli favorisant l'exploration de nouvelles régions.

Selon les auteurs du formalisme ACO, il est possible d'implémenter d'autres processus nécessitant un contrôle centralisé (et donc ne pouvant être directement pris en charge par des fourmis), sous la forme de processus annexes. Ce n'est, à notre sens, que peu souhaitable ; en effet, on perd alors la caractéristique décentralisée du système.

De plus, l'implémentation de processus annexes entre difficilement dans une formalisation rigoureuse.

IV.2.4.2. Phéromones et mémoire

La *stigmergie* est l'un des mécanismes fondamentaux caractérisant les algorithmes ACO, Elle est précisément définie comme une forme de communication passant par le biais de modifications de l'environnement [23]. La grande importance de ce phénomène est que les individus échangent des informations par le biais du travail en cours, de l'état d'avancement de la tache globale à accomplir. Dans les algorithmes ACO ce mécanisme est apparu clairement dans le processus de construction des pistes de phéromone, ce dernier à un grand impact sur la qualité des résultats obtenus.

Également, le choix de la méthode d'implémentation des pistes de phéromone est très important. Ce choix est en grande partie lié aux possibilités de *représentation* de l'espace de recherche, chaque représentation pouvant apporter une façon différente d'implémenter les pistes. Par exemple, pour le problème TSP, une implémentation efficace consiste à utiliser une piste τ_{ij} entre deux villes i et j comme une représentation de l'intérêt de visiter la ville j après la ville i. Une autre représentation possible, moins efficace en pratique, consiste à considérer τ_{ij} comme une représentation de l'intérêt de visiter i en tant que j*ème* ville.

En effet, les pistes de phéromone décrivent à chaque pas l'état de la recherche de la solution par le système, les agents (fourmis) modifient la façon dont le problème va être représenté et perçu par les autres agents. Cette information est partagée par le biais des modifications de l'environnement des fourmis, grâce au stigmergie. L'information est donc stockée un certain temps dans le système, ce qui a amené certains auteurs à considérer ce processus comme une forme de *mémoire adaptative* [Tail 01], où la dynamique de stockage et de partage de l'information va être cruciale pour le système.

IV.2.4.3. Intensification/diversification

Un point critique, lors de l'application d'un algorithme ACO, est de trouver un équilibre entre l'exploitation (*intensification*) de l'expérience de recherche acquise par les fourmis et l'exploration (*diversification*) de nouvelles zones de l'espace de recherche non encore visitées. Les algorithmes ACO possèdent 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 relatives des traces de phéromone, essentiellement les deux paramètres α et ρ.

Le paramètre 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 de 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.

IV.2.4.4. ACO et la recherche locale

Dans plusieurs applications à des problèmes d'optimisation combinatoire difficiles, les algorithmes ACO réalisent de meilleures performances lorsqu'ils sont hybridés 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 ACO, les fourmis génèrent aléatoirement, en exploitant les traces de phéromone, des solutions initiales prometteuses pour la recherche locale.

IV.2.4.5. Parallélisme

Un algorithme ACO est caractérisé principalement par une structure équipé d'un parallélisme *intrinsèque*. D'une manière générale, les solutions de bonnes qualités émergentes du résultat des *interactions* indirectes ayant cours dans le système, pas d'un codage explicite d'échanges. En effet, chaque fourmi ne prend en compte que des informations locales de son environnement (les pistes de phéromone) ; il est donc facile de paralléliser un tel algorithme. Il est intéressant de noter que les différents processus en cours dans la métaheuristique (c.-à-d. le comportement des fourmis, l'évaporation et les processus annexes) peuvent également être implémentés de manière indépendante, l'utilisateur étant libre de décider de la manière dont ils vont interagir.

IV.2.4.6. Convergence

Les métaheuristiques peuvent être vues comme des modifications d'un algorithme de base : une recherche aléatoire. Cet algorithme possède une propriété très intéressante, il permet de garantir que la solution optimale sera trouvée tôt ou tard, on parle alors de convergence. Cependant, puisque cet algorithme de base est biaisé, la garantie de convergence n'existe plus.

Si, dans certains cas, on peut facilement être certain de la convergence d'un algorithme de colonies de fourmis (MMAS p. ex.), le problème reste entier en ce qui concerne la convergence d'un algorithme ACO quelconque. Cependant, il existe plusieurs variantes dont la convergence à été prouvée : Le "Graph-Based Ant System" (GBAS) [32]. La différence entre GBAS et l'algorithme AS se situe au niveau de la mise-à-jour des pistes de phéromone, qui n'est permise que si une meilleure solution est trouvée. Pour certaines valeurs de paramètres, et étant donné $\epsilon > 0$ une "faible" valeur, l'algorithme trouvera la solution optimale avec une probabilité $P_t \geq 1 - \epsilon$, après un temps $t \geq t_0$(où t_0 est fonction de ϵ).

Dans le même contexte, une nouvelle variante de ACO dite *ACO with stench pheromone* (ACO-SP) [33] à été proposée récemment avec une garantie de la convergence. Cet algorithme est créé dont le but d'optimiser la circulation dans un réseau de trafic de type dynamique. ACO-SP a deux différences fondamentales, résident principalement dans le modèle de phéromone utilisé (02 modèles de

phéromone) : le premier étant une phéromone régulière produit afin d'attirer les fourmis caractérisées par un faible coût, tandis que le second étant une phéromone de mauvaise odeur (*stench pheromone*) utilisé afin d'aliéner les fourmis lors de la détection d'une stagnation quelconque.

IV.3. Planification d'une trajectoire via un algorithme ACO, principe de la méthode

IV.3.1. Présentation préliminaire

Dans ce qui suit, nous présentons l'application d'un algorithme ACO sous sa version primitive dans la résolution d'un problème de planification d'un chemin à coût minimal relatif à un robot mobile, en imitant le comportement de recherche de nourriture d'une colonie de fourmis réelles. En effet, lors de la recherche d'une source de nourriture, les fourmis fixent leur direction du mouvement basant sur des informations relatives à la visibilité locale et d'autres (informations) relatives à l'intensité des phéromones environnementales. Comme nous l'avons indiqué, après avoir trouvé une source de nourriture (le plus souvent fixe), les fourmis laisseraient des phéromones sur leurs parcours afin d'attirer et guider l'ensemble de la colonie vers la source de nourriture objective de fourrage, basant sur ses caractéristiques biologiques est inespéré la méthode de planification de chemin pour un robot mobile.

IV.3.2. Adaptation du ACO au problème traité

Il est très clair que, au niveau d'un processus de recherche effectué par une colonie de fourmis (sous sa forme la plus simple), le facteur phéromone joue un rôle vital dans la réussite d'opération de recherche. Plus précisément, à chaque étape de déplacement, chaque fourmi dans la colonie, doit trouver une grille voisine possible vers laquelle, il pourrait se déplacer. En effet, comme il est montré sur Figure IV.5, qui présente ici un simple exemple environnemental, une fourmi quelconque situé dans une grille dite g, dont elle est entourée par huit nœuds voisins, dont les cinq nœuds qui ont une couleur bleue sont des nœuds accessibles. Fondamentalement, la première application d'un algorithme de type ACO dans le problème en question est basée principalement sur l'emploi des deux formations suivantes (i), la visibilité locale et (ii) L'intensité des pistes de phéromones.

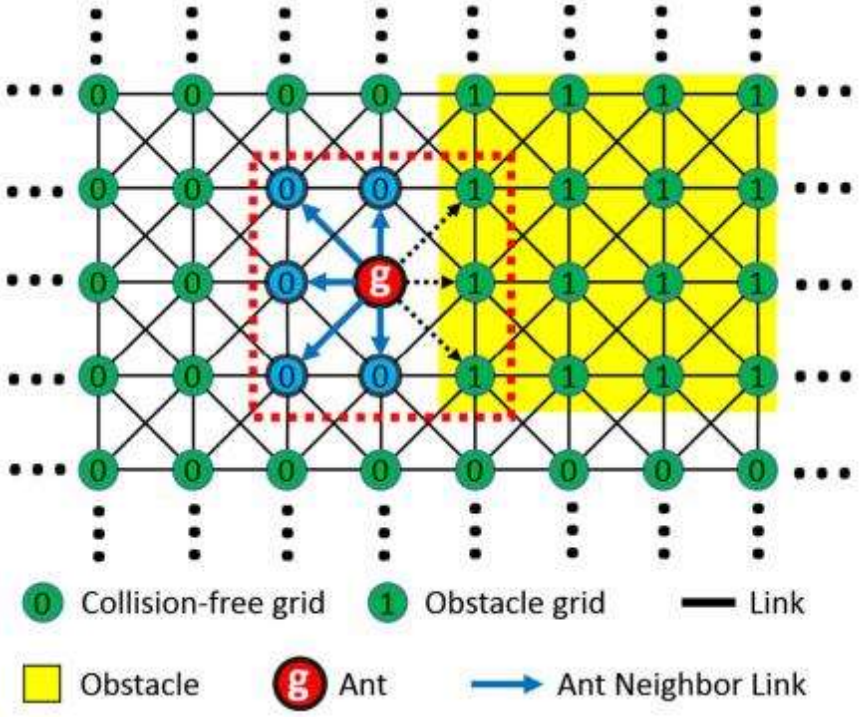


Figure IV.5. L'environnement d'une fourmi est divisé en $(N_x \times N_y)$ grilles connectées. La région jaune représente un espace obstacle, dont les grilles appartenant à cet endroit sont notées par **"1"**. Les autres sont situés dans un espace sans collision, ils sont désignés par un **"0"**. La fourmi est située dans la grille g, et elle a huit grille voisines, dans lesquelles les cinq grilles bleues sont des grilles accessibles, et l'autre trois grilles sont occupés par des obstacles.

Dans ce qui suit, on présente on détaille toutes les étapes (démarches) suivies afin de conduire le robot mobile vers une destination finale (grille finale), en parcourant un chemin (*Path*) optimal et en utilisant un algorithme ACO classique.

Il est à noter que, la présentation en question a accompagné les données utilisées aux tests de simulations.

Au début, le mécanisme de recherche ou de conduite, composé principalement d'un environnement adapté au problème traité (recherche d'un chemin optimal), dans lequel se déplace le robot ainsi l'organe de recherche, responsable d'opération de conduite du robot, qui est dans notre cas l'algorithme ACO.

Au début du mécanisme de recherche, on fixe les éléments suivants :

1- L'enivrement dans lequel, se déplace le robot :

Ce dernier dit *grid map* schématisé sur la Figure IV.5, est la plus utilisé lors d'une résolution d'un problème de recherche de chemin optimal relatif à robot mobile en utilisant un algorithme ACO (voir

p.ex. [?], [?]). En effet, dans nos essais de simulation, nous avons utilisé l'environnement quadrillé illustré sur la Figure IV.6, dont il composé de vingt (20) lignes (rows) indiquées par N_x et vingt (20) colonnes (columns) indiquées par N_y, en somme nous avons $N_x \times N_y = 400$ grilles g.

De plus, dans ce champ de déplacement les grilles libres blancs sont symbolisées par des '0', ainsi les grilles occupées rouges (des obstacles) sont symbolisées par des '1' (voir la Figure IV.7).

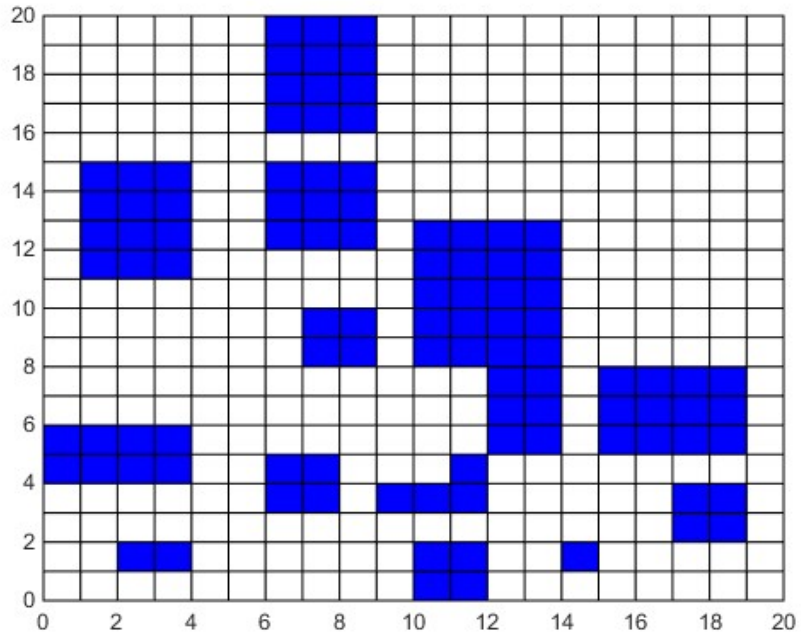


Figure IV.6. Champ de déplacement utilisé en simulation.

En effet, ce champ est mobilisé mathématique sous le MATLAB (logiciel utilisé), par une matrice dite G, illustrée dans ce qui suit :

```
% Map matrix
|
G=[0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
   0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0;
   0 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0;
   0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0;
   0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0;
   0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0;
   0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1;
   0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1;
   1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1;
   1 1 1 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0;
   0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 0 1 1 0;
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0;
   0 0 1 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0;
   0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0;];
```

Figure IV.7. Matrice G qui modélise le champ de déplacement sous le MATLAB.

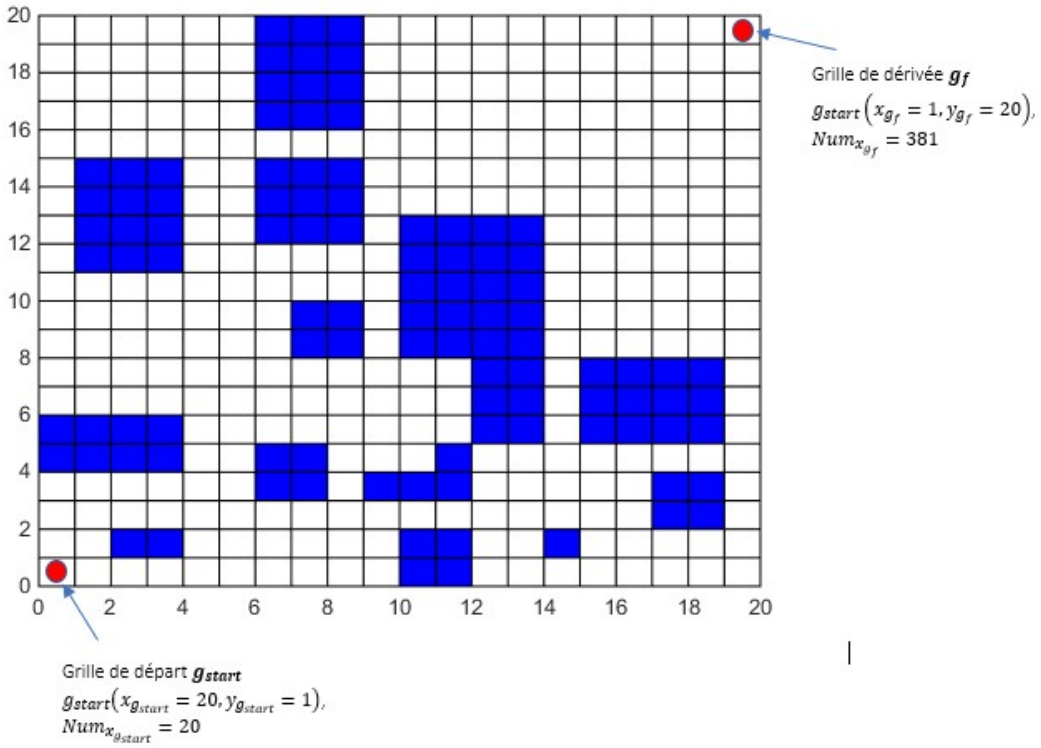


Figure IV.8. Grilles de départ g_{start} et celle d'arrivée g_f avec leurs coordonnées et leurs numéros.

2- Paramètres de l'algorithme ACO :

Le tableau ci-dessous, donne les différents paramètres de contrôle du ACO utilisé :

Paramètres	Valeurs
Nombre de fourmis m	50
α, β Facteurs de la fonction de transition	$2,6$
ρ, Q Paramètres de mise-à-jour des pistes de phéromones	$0.1,1$
$Iter\_max$ Nombre maximal d'itération	200
$\eta_{g_j\_Initialle}$ Matrice de transition aléatoire relative à chaque grille g_j	$1./D_{g_j g_f}$ (1)
$\tau_{g_j\_Initialle}$ Matrice de concentration intaille de phéromone relatif à chaque grille g_j	$10.*\eta_{g_j\_Initialle}$

Tableau IV.1 : Paramètres du ACO.

(1), $D_{g_j g_f}$ est une matrice de dimension contient les distances entre toutes les grilles g_j avec $j = 1, \dots (N_x \times N_y) = 400$ et la grille finale g_f. En effet, cette distance est calculée en utilisant l'équation (IV.8) indiquée ci-dessous.

$$D_{g_j g_f} = \sqrt{\left(x_{g_j} - x_{g_f}\right)^2 + \left(y_{g_j} - y_{g_f}\right)^2} \quad \text{(IV.8)}$$

L'étape suivante, concerne le calcul d'une matrice dite de mouvement $D_{move\ (n\times n)\times 8}$, cette matrice contient $(N_x \times N_y) = 400$ lignes (l'exemple actuel) et 8 colonnes. Plus précisément, chaque ligne j avec $J = 1, \dots, (N_x \times N_y)$, contient 8 colonnes, dont chacune contient les huit (même moins) déplacement (grilles) possible (ou non), qui peut faire le robot à partir de sa position actuelle (grille g_j).

Au lancement du processus de recherche (boucle d'itérations), toutes les fourmis sont intégrées afin de réaliser dans un premier temps les tâches suivantes :

1. La génération d'un vecteur $D_{Work_{1\times 8}} = D_{move\ (i_{Strart}\times 8)}$, $i_{Strart} = 1$ correspond à la grille de départ,
2. La génération d'un vecteur dit $Tabau_{1\times(n\times n)} = \mathbf{1}_{1\times(n\times n)}$ a l'exception que ce dernier est soumis à un changement comme suit :

$$Tabau_{1\times(n\times n)} = \begin{cases} 1 & grille\ non\ visitée \\ 0 & grille\ visitée \end{cases} \quad \text{(IV.9)}$$

3. Le vecteur $D_{Work_{1\times 8}}$ sera l'objet à un changement vu le vecteur $Tabau_{1\times(n\times n)}$ comme suit :

$$D_{Work_{1\times 8}} = \begin{cases} \left[D_{Work_{1\times 8}} \quad zeros(1,8 - length(D_{Work_{1\times 8}}))\right] & Tabau_{1\times(n\times n)}\left(D_{Work_{1\times 8}}(j_0)\right) = 0 \\ D_{Work_{1\times 8}} & si\ non \end{cases} \quad \text{(IV.10)}$$

Avec j_0 c'est l'indice qui signifié à une grille dans $D_{Work_{1\times 8}}$ déjà visitée. Il est à noter que, le premier changement effectué sur $D_{Work_{1\times 8}}$ dans (IV.10), est suit d'une vidange (vecteur vide) de la case dans $D_{Work_{1\times 8}}$ corresponde à la grille déjà visitée.

Tout le processus de recherche est représenté par l'organigramme illustrée sur la Figure IV.9. En somme à la fin de ce dernier on obtient :

1. Le chemin optimal relatif à chaque itération $Iter$ pour toutes les fourmis k qui ont réalisé un chemin complet :

$$Min\_global\_Dist_{(Iter,k)} = Min\left(Dist_{(Iter,k)}\right) ; iter = 1,.,Iter\_max \text{ et } k = 1,.,m.$$

2. La fourmi k symbolisée par (Min_k) qui réalise le chemin le plus cours durant tout le processus de recherche c.à.d. de ; $iter = 1$ à $iter = Iter\_max$, ainsi l'itération symbolisée par (Min_{Iter}) dans laquelle cette dernière à réaliser ce chemin.

Il est à noter que, une opération de mise-à-jour des pistes de phéromones est réalisée à la chaque itération par toutes les fourmis k qui ont réalisé un chemin complet, comme suit :

$$\tau_{g_j}(iter+1) = (1-\rho)\tau_{g_j}(iter) + \sum_{k=1}^{m} \Delta\tau_{g_j}^{k} \quad \text{(IV.12)}$$

D'où ρ est le taux d'évaporation des pistes de phéromones, m est le nombre de fourmis, $\Delta\tau_{g_j}^{k}$ est l'intensité de phéromone d'une fourmi k parcourant toutes les g_j (avec $j = 1, \ldots., Nombre\ de\ toutes\ les\ grilles\ dans\ le\ chemin\ complet\ réalisé$) jusqu'à la grille finale g_f :

$$\Delta\tau_{g_j}^{k} = \begin{cases} \frac{Q}{langeur\_Path_k}, & si\ k\ trouve\ un\ chemin\ complet \\ 0, & Autrement \end{cases} \quad \text{(IV.14)}$$

Aussi, le passage entre les grilles appartenant à un vecteur $D_{Work_{1\times 8}}$ par une fourmi k est réalisé en utilisant la règle de transition suivante :

$$P_{g_j}^k(iter) = \begin{cases} \tau_{g_j}^{\alpha}(iter)\eta_{g_j}^{\beta} & si\ g_j \in D_{Work_{1\times 8}} \\ 0 & Autrement \end{cases} \quad \text{(IV.15)}$$

Par la suite, le vecteur $P_{g_j}^k(iter)$ est mis à une normalisation comme suit :

$$P_{g_j}^k = P_{g_j}^k + \sum_{j=1}^{Nombre\ de\ gj\ dans\ D_{Work_{1\times 8}}} P_{g_j}^k \quad \text{(IV.16)}$$

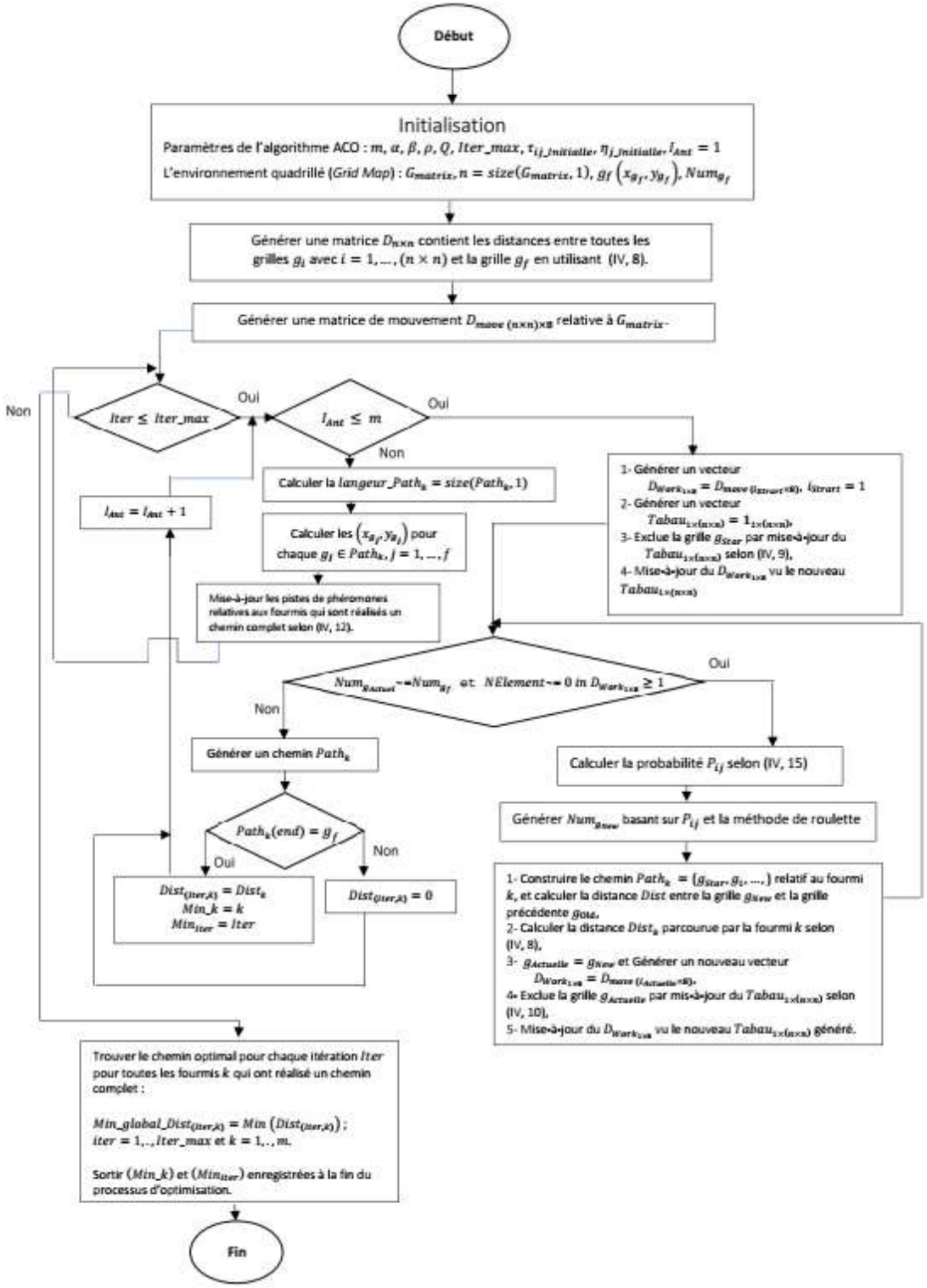


Figure IV.9. Organigramme d'une recherche d'un chemin optimal en utilisant un algorithme ACO.

$$Select_i = P_{g_j}^k \geq rand()$$
$$g_{Next} = Select(1) \quad \text{(IV.17)}$$

La grille suivante g_{Next} est choisi comme l'élément numéro 1 dans le vecteur $Select_i$.

Dans ce qui suit, on présente les résultats de simulation relatifs à trois scénarios différents dépend des positions des grilles de départs et d'arrivées.

Toutes les données relatives à ces derniers, sont illustrées dans le tableau ci-dessous :

Scénario	Numéro de la grille g_{Start} de départ	Cordonnées de la grille de départ g_{Start}	Numéro de la grille d'arrivée g_f	Cordonnées de la grille d'arrivée g_f
1	20	(20,1)	381	(1,20)
2	2	(2,1)	399	(19,20)
3	125	(5,7)	190	(10,15)

Tableau IV.2 : Trois scénarios de scénarios de planification de trajectoire.

Ci-dessous sont illustrées, les mouvements du robot (chemin optimal tracé) relatifs aux trois scénarios indiqués sur le tableau IV.2, ainsi les courbes d'évaluation des distances des chemins optimaux réalisés pour chaque itération jusqu'à la convergence vers le chemin optimal global atteint par l'ensemble de la colonie.

Scénario 1

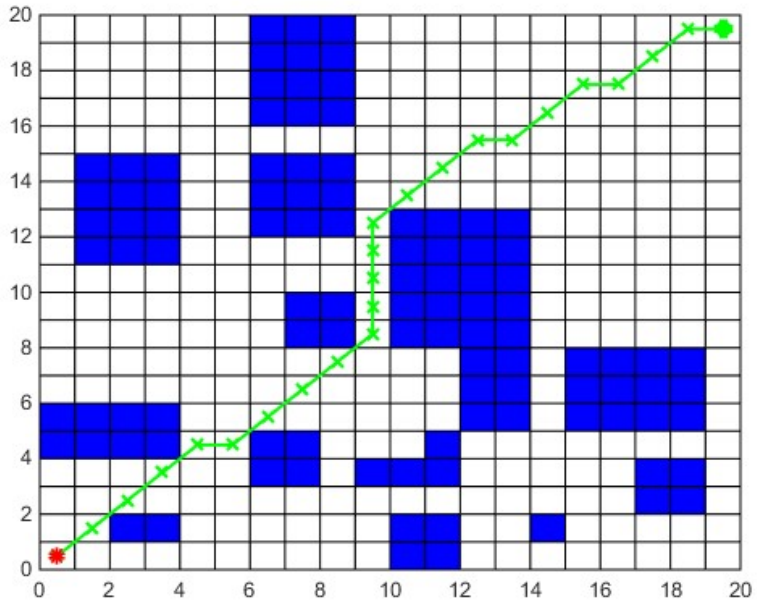


Figure IV.10. Chemin optimal trouvé relatif au scénario 1.

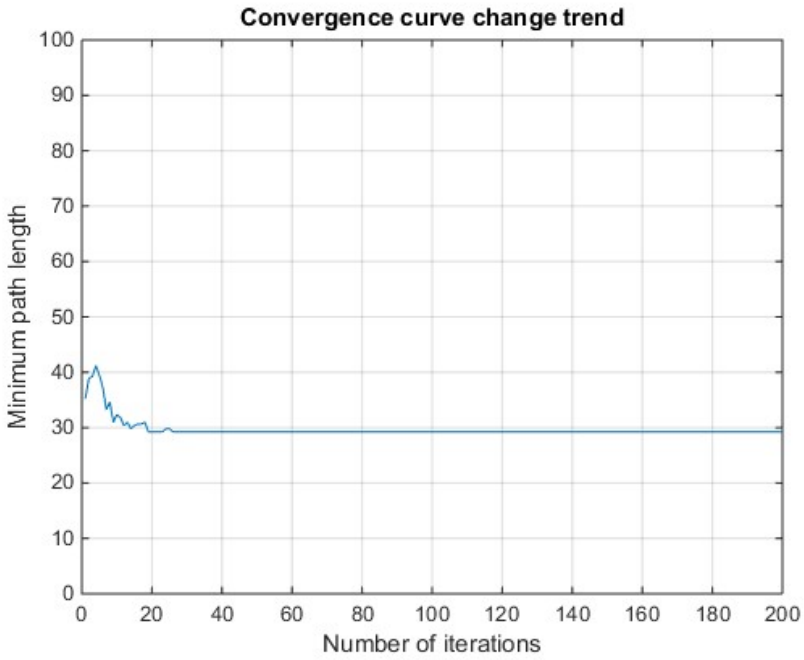


Figure IV.11. Evaluation des distances des chemins optimaux réalisés pour chaque itération relative au scénario 1.

Scénario 2

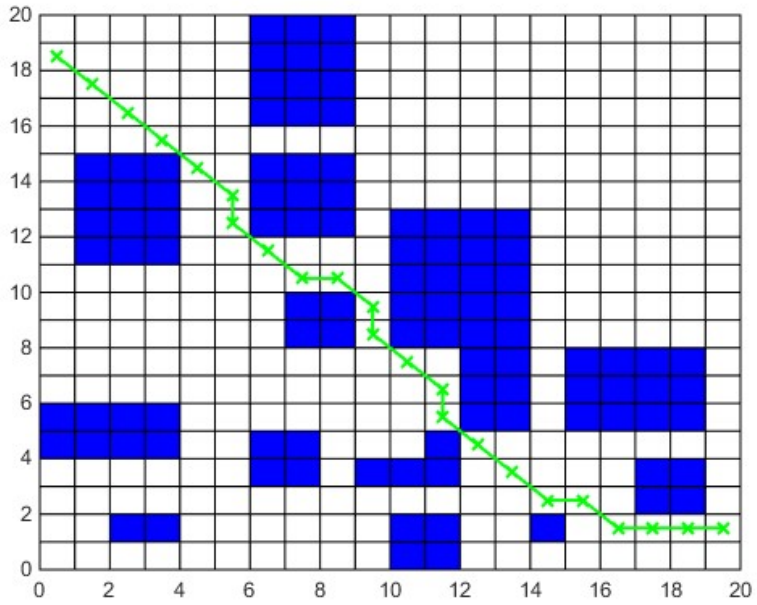


Figure IV.12. Chemin optimal trouvé relatif au scénario 2.

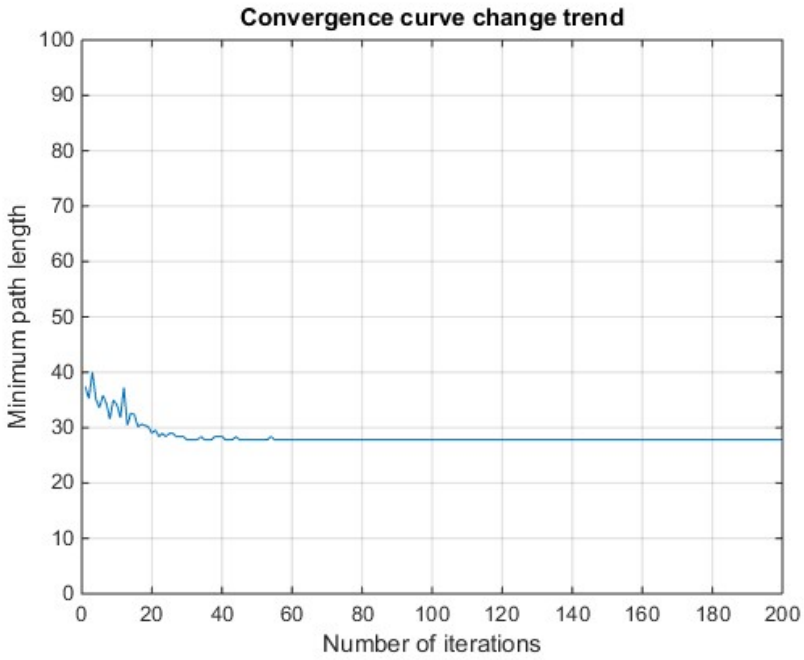


Figure IV.13. Evaluation des distances des chemins optimaux réalisés pour chaque itération relative au scénario 2.

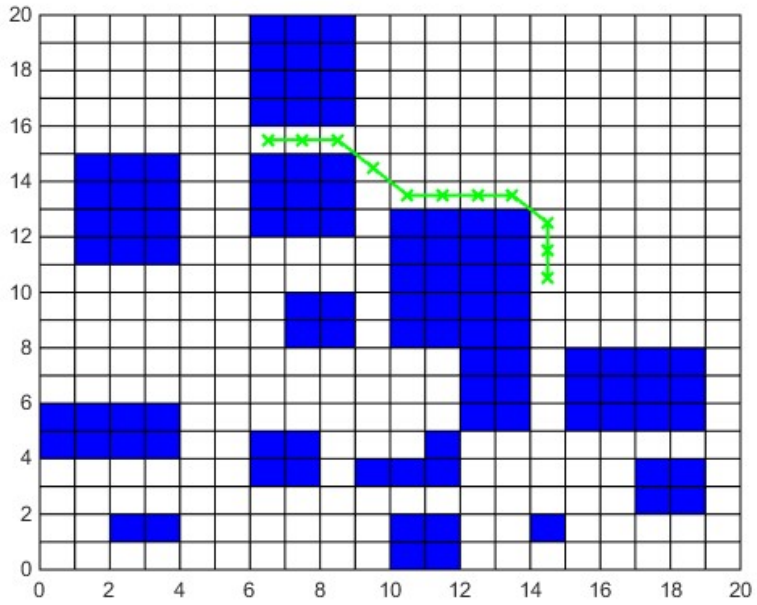


Figure IV.14. Chemin optimal trouvé relatif au scénario 3.

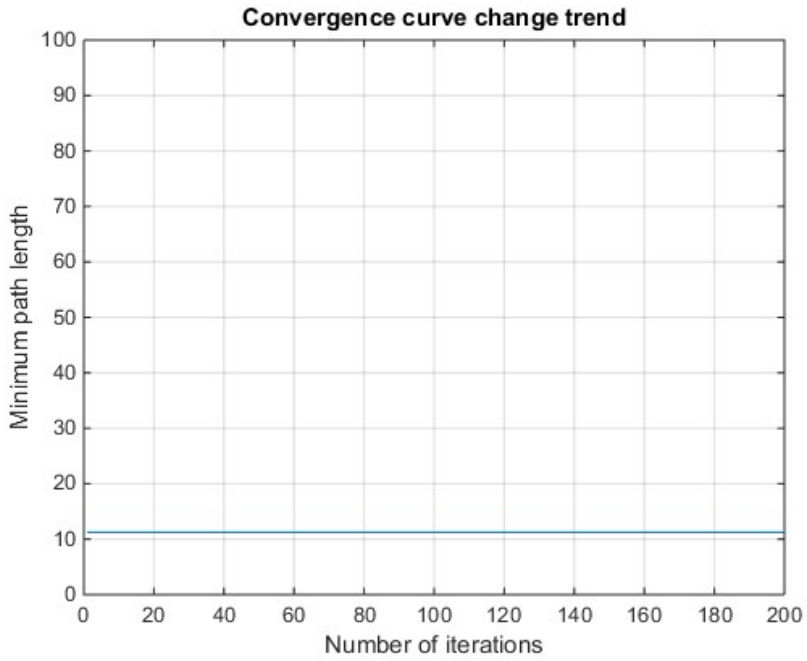


Figure IV.15. Evaluation des distances des chemins optimaux réalisés pour chaque itération relative au scénario 3.

Scénario	Longueur du chemin optimal global trouvé	Fourmi optimale	Itération optimale
1	29.2132	14	19
2	27.7990	20	30
3	11.2426	3	1

Tableau IV.3 : Longueur du chemin optimal global trouvé, fourmi optimale et itération optimale pour chaque scénario.

.

De tous ces résultats, on constate que la méthode utilisée est très efficace dans la résolution de ce genre de problème. On observe aussi, que l'algorithme ACO a montré une robustesse remarquable vis-à-vis le changement dans deux conditions environnementales du robot (trois scénarios différents), ses positions de départ et d'arrivée. Aussi, en termes de rapidité de convergence vers une solution optimale (le chemin le plus court), on constate que l'algorithme utilisé, permet de conduire le robot vers sa position finale après un nombre relativement minimal d'itérations (voir Figure 11,13 et 15).

Il est à noter que, pour avoir une crédibilité de plus pour ce travail, il reste à lui comparer avec d'autres méthodes (citées dans le chapitre III), chose à faire dans nos prochains travaux.

IV.4. Conclusion

Dans ce chapitre, nous avons présenté la résolution du problème de planification d'une trajectoire (chemin) relative à un robot mobile, en utilisant un algorithme ACO sous forme primitive. En effet, une description de ce dernier a été donnée, par la suite nous avons montré comment adapter cet algorithme à ce genre de problème.

Les résultats de simulations obtenus, sont très prometteuses, ils montrent réellement l'efficacité de la méthode utilisée.

Conclusion générale

Le travail présenté dans ce mémoire, a pour but de montrer la résolution d'un problème très connu en robotique mobile ; le problème de planification d'une trajectoire, en utilisant une méthode basée sur l'emploi d'une classe d'algorithmes évolutionnaires dits ; Algorithmes de Colonies de Fourmis et en anglais Ant Colony Optimization (ACO) [4].

En effet, le présent manuscrit porte au début (les chapitre I, II, et III) quelques informations de bases relatives à/aux : (i) la robotique mobile et ses fondements, présentées sous forme simple et résumée, (ii) notions théoriques très importantes relatives à la planification d'un chemin pour un robot mobile au sens général, (iii) quelques algorithmes (les plus connus) de planification d'une trajectoire.

Les algorithmes évolutionnaires, quelque soit leur type ont connus une grande utilisation presque dans tous les domaines (l'ingénierie, la médicine, l'économie,….) avec des objectifs variés (selon le but envisagé). La robotique, est plus précisément celle de type mobile, est connue l'utilisation de ce genre de méthodes.

Dans ce mémoire de Master, nous avons présenté l'utilisation d'un algorithme ACO sous sa forme primitive, dans la résolution du problème de planification d'une trajectoire d'un robot mobile intelligent. La méthode exposée, à permettre au robot mobile d'éviter un ensemble d'obstacles et de suivre des chemins optimaux dans son environnement. Les trajectoires générées, sont basées sur l'utilisation des deux règles principales caractérisant l'algorithme ACO, (i) la fonction (ou la règle) de transition et (ii) la règle de mise-à-jour des pistes de phéromones.

Les résultats de la simulation montrent la capacité de cette méthode, de conduire le robot de suivre des trajectoires de qualité, en évitant des obstacles distribués aléatoirement dans son environnement, définies avec des tailles et des positions différentes.

Malgré que, la méthode exposée à montrer son efficacité en plusieurs termes (robustesse, flexibilité, rapidité,…..), il reste de lui comparer avec d'autres méthodes et de le faire implémenter si possible sur un prototype réel.

Références Bibliographiques

[1] Y. F. Wang and G. S. Chirikjian, “A new potential field method for robot path planning,” in *Proc. IEEE Int. Robot. Automat. Conf.*, San Francisco, CA, 2000, pp. 977–982.

[2] H.-P. Huang and S.-Y. Chung, “Dynamic visibility graph for path planning,” in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004, pp. 2813–2818.

[3] G. Klancar, A. Zdešar, S. Blaži ˇ c, and I. Škrjanc, ˇ *Wheeled mobile robotics - From fundamentals towards autonomous systems*. Elsevier, Butterworth-Heinemann, 2017.

[4] M. Dorigo, V. Maniezzo, and A. Colorni. “Ant System: optimization by a colony of cooperating agents”. *IEEE Transactions on Systems, Man, and Cybernetics, Part B : Cybernetics*, Vol. 26, No. 1, pp. 29–41, 1996.

[5] Siegwart, R., Nourbakhsh, I.: Introduction to Autonomous Mobile Robots. MIT Press (2004).

[6] A. De Luca, G. Oriolo & P. R. Giordano, "Kinematic modeling and redundancy resolution for nonholonomic Mobile manipulators", Proc.s of the 2006 IEEE Inter. Conf. on Robotics and Automation, Orlando, Florida, pp. 1867-1873, May 2006.

[7] D. Filliat, "Cartographie et estimation globale de la position pour un robot mobile autonome", Thèse de doctorat de l’université Paris 6 Pierre et Marie Curie, France, 2001.

[8] N. Morette, "Contribution à la navigation de robots mobiles : approche par modèle direct et commande prédictive", Thèse de doctorat de l’Institut Prisme, Equipe Systèmes Robotiques interactifs, Université d’Orléans, France, 2009.

[9] S.M. LaValle, Rapidly-exploring random trees: a new tool for path planning, Technical Report TR 98-11, Computer Science Department, Iowa State University, Iowa, 2013.

[10] J.J. Kuffner, S.M. LaValle, RRT-connect: an efficient approach to single-query path planning, in: IEEE International Conference on Robotics and Automation (ICRA 2000), IEEE, 2000, pp. 1–7.

[11] L.E. Kavraki, P. Svestka, J.-C. Latombe, M.H. Overmars, Probabilistic roadmaps fo path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. 12 (4) (1996) 566–580.

[12] V. Lumelsky, P. Stepanov, Dynamic path planning for a mobile automaton with limited information on the environment, IEEE Trans. Autom. Control 31 (11) (1986) 1058–1063.

[13] A. Sankaranarayanan, M. Vidyasagar, A new path planning algorithm for moving a point object amidst unknown obstacles in a plane, in: IEEE Conference on Robotics and Automation, IEEE, 1990, pp. 1930 1936.

[14] I. Kamon, E. Rivlin, Sensory-based motion planning with global proofs, IEEE Trans. Robot. 13 (6) (1997) 814–821.

[15] S. Laubach, J. Burdick, An autonomous sensor-based path-planner for planetary microrovers, in: IEEE Conference on Robotics and Automation, IEEE, 1999, pp. 347–354.

[16] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1 :269–271, 1959.

[17] H. Kang, B. Lee, and K. Kim. Path planning algorithm using the particle swarm optimization and the improved dijkstra algorithm. In Proceedings of the IEEE Workshop on Computational Intelligence and Industrial Application, volume 2, pages 1002–1004, Wuhan, China, Dec 19-20, 2008.

[18] G. Qing, Z. Zheng and X. Yue, “Path-planning of automated guided vehicle based on improved Dijkstra algorithm,” 2017 29th Chinese Control And Decision Conference (CCDC), Chongqing, 2017, pp. 7138-7143.

[19] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael, A formal basis for the heuristic determination of minimum cost paths, Systems Science and Cybernetics, IEEE Transactions on 4.2 (1968): 100-107.

[20] M. Dorigo, V. Maniezzo, and A. Colorni. “Positive feedback as a search strategy “. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.

[21] M. Dorigo. “Optimization, Learning and Natural Algorithms “. PhD thesis, Politecnico di Milano, Italy, 1992.

[22] A. Colorni, M. Dorigo, and V. Maniezzo. “Distributed Optimization by Ant Colonies”. In: Proceedings of the First European Conference on Artificial Life, pp. 134–142, MIT Press, Paris, France, December 1991.

[23] J. Dréo, A. Pétrowski, P. Siarry, and É. Taillard. “Métaheuristiques pour l'optimisation difficile“. Éditions Eyrolles, 2003.

[25] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels. “Self-organized shortcuts in the Argentine ant”. Naturwissenschaften, Vol. 76, No. 12, pp. 579–581, 1989.

[26] B. Bullnheimer, R. Hartl, and C. Strauss. “A New Rank-based Version of the Ant System: A Computational Study”. Central European Journal for Operations Research and Economics, Vol. 7, No. 1, pp. 25–38, 1999.
[27] O. Cordon, I. D. Viana, F. Herrera, and L. Moreno. “A New ACO Model Integrating Evolutionally Computation Concepts: The Best-Worst Ant System”. In: Proceedings of the Second International Workshop on Ant Algorithms, IEEE Transaction on Evolutionary Computation, pp. 22–29, Brussels, Belgium, September 2000.
[28] T. Stützle and H. Hoos. “MAX-MIN Ant System”. Future Generation Computer Systems, Vol. 16, No. 8, pp. 889–914, 2000.
[29] T. Stützle and H. Hoos. “Improvements on Ant System: Introducing MAX-MIN Ant System”. In : Proceedings of the Third International Conference of Artificial Neural Networks and Genetic Algorithms, Springer-Verlag, Norwich, UK, April 1997.
[30] M. Dorigo and L. Gambardella. “Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem”. IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 53–66, 1997.
[31] J. Dréo. “Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical“. Thèse de doctorat en sciences, Spécialité : Génie Biologique et Médical, Option : Optimisation, Université Paris 12, Val de Marne, UFR de Sciences, 2004
[32] W. J. Gutjahr. “A Graph-based Ant System and its convergence “. Future Generation Computer Systems, Vol. 16, No. 8, pp. 873–888, 2000
[33] Z. Cong, B. De Schutter, and R. Babuska. “On the convergence of Ant Colony Optimization with stench pheromone “. IEEE Congress on Evolutionary, pp. 1876–1883, Cancun, 20–23 June 2013.
[34] O. Montiel-Ross, R. Sepúlveda, O. Castillo, and P. Melin, “Ant colony test center for planning autonomous mobile navigation,” *Computer Applications in Engineering Education*, vol. 21, no. 2, pp. 214–229, 2013.
[35] X. M. You, S. Liu, and J. Q. Lv, “Ant colony algorithm based on dynamic search strategy and its application on path planning of robot,” *Control and Decision*, vol. 32, pp. 552–556, 2017, in Chinese.
[36] Gao, W., Tang, Q., Ye, B. *et al.* An enhanced heuristic ant colony optimization for mobile robot path planning. *Soft Comput* 24**,** 6139–6150 (2020).