RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA
RECHERCHE SCIENTIFIQUE
UNIVERSITÉ MENTOURI DE CONSTANTINE
FACULTÉ DES SCIENCES DE L'INGÉNIEUR
DÉPARTEMENT D'INFORMATIQUE

N° Ordre……
N° Série……

MÉMOIRE

POUR L'OBTENTION DU DIPLÔME DE MAGISTER EN INFORMATIQUE

THÈME :

RÉSOLUTION DE PROBLÈMES D'OPTIMISATION COMBINATOIRE PAR SYSTÈMES ARTIFICIELS AUTO-ORGANISÉS

PRÉSENTÉ PAR :

MOSTEPHA REDOUANE KHOUADJIA

DIRIGÉ PAR :

Dr. SALIM CHIKHI

DEVANT LE JURY COMPOSÉ DE :

Dr. M-K KHOLLADI	Université de Constantine	Président
Dr. S. CHIKHI	Université de Constantine	Rapporteur
Dr. A. CHAOUI	Université de Constantine	Examinateur
Dr. D-E. SAIDOUNI	Université de Constantine	Examinateur

Octobre 2008


بسم الله الرحمن الرحيم

A mes parents…

Remerciements

Je tiens à remercier Monsieur Salim Chikhi qui m'a encadré durant mon travail de recherche et qui a su m'aider avec les connaissances dont il m'a gratifié durant mon travail.
Les conseils judicieux qu'il m'a accordés ont été pour moi une source inépuisable d'inspiration. Travailler sous sa direction aura enrichi énormément ma formation.

Je remercie chaleureusement Monsieur Mohamed-Khireddine Kholladi de m'avoir fait l'honneur d'accepter d'être président du Jury.

De même, j'adresse des remerciements chaleureux à Monsieur Djamel-Eddine Saidouni et Monsieur Allaoua Chaoui, qui ont accepté d'être les examinateurs de ce travail. En particulier, je les remercie pour l'attention qu'ils ont portée à mon travail et pour le temps passé à l'étudier.

Que Monsieur Mohamed Batouche et Mme Souham Meshoul soient remerciés pour leur soutien tout le long de mon parcours universitaire. Ils m'ont appris le goût de la recherche scientifique, grâce à eux j'ai confirmé mon intérêt pour ce domaine.

Je remercie aussi les membres de l'équipe-projet Dolphin de l'INRIA Lille Nord-Europe, qui ont contribué pour une partie à achever mon travail, en m'octroyant un environnement de travail agréable, motivant, et très constructif.

Je tiens à exprimer ma profonde gratitude envers Monsieur El-Ghazali Talbi directeur de l'équipe Dolphin, pour ses encouragements le long de mes recherches. Je le remercie vivement pour ses conseils judicieux qui ont permis de l'enrichir.

Que Mme Laetitia Jourdan, chargée de recherche à L'INRIA Lille Nord-Europe trouve ici toute ma reconnaissance pour ses conseils précieux et son soutien.

Mes remerciements les plus vifs vont vers mes amis du quartier, ceux de l'université de Constantine, ainsi que ceux de l'université des sciences et technologies de Lille.

Enfin, ce travail n'aurait pu aboutir sans le soutien constant de mes parents. Je tiens à les remercier pour leur affection et leur soutien constant qu'ils m'ont accordés depuis le début de mes études, sans oublier mes sœurs, mon frère, et mes beaux frères, pour leurs encouragements et leur dévouement.

Résolution de problèmes d'optimisation combinatoire par systèmes artificiels auto-organisés

RÉSUMÉ

Dans ce mémoire nous présentons une nouvelle métaheuristique s'inspirant de l'auto-organisation chez l'espèce de fourmis*Pachycondyla apicalis*. Cette auto-organisation se résume en des interactions simples et parallèles de l'ensemble des fourmis de la colonie.

Par ailleurs, son élaboration s'appuie sur le concept de programmation à mémoire adaptative. Ce concept présente un cadre unifié de conception de métaheuristiques en se basant sur une mémoire, ainsi que sur des mécanismes d'intensification et de diversification, permettant de guide la recherche de solutions.

La métaheuristique proposée est implémentée sur la plate-forme dédiée à la conception et à l'implémentation de métaheuristiques ParadisEO. Elle est appliquée sur le problème du voyageur de commerce. Les résultats obtenus à la suite des expérimentations sont très satisfaisants, et mettent en évidence la théorie de l'auto-organisation comme nouveau paradigme pour la conception de métaheuristiques.

Mots-clés : Métaheuristiques bio-inspirées, algorithmes des colonies de fourmis, fourmis *Pachycondyla aplicalis*, auto-organisation, Programmation à Mémoire Adaptative, plate-forme ParadisEO.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

Les ingénieurs et les décideurs sont confrontés quotidiennement à des problèmes de complexité grandissante, qui surgissent dans des secteurs techniques très divers, comme dans la recherche opérationnelle, la conception de systèmes mécaniques, le traitement des images, et tout particulièrement en électronique (C.A.O. de circuits électriques, placement et routage de composants, amélioration des performances ou du rendement de fabrication de circuits,…).

Le problème à résoudre peut souvent s'exprimer comme un problème d'optimisation : on définit une fonction objectif, ou fonction de coût (voire plusieurs), que l'on cherche à minimiser ou à maximiser par rapport à tous les paramètres concernés. La définition du problème d'optimisation est souvent complétée par la donnée de contraintes : tous les paramètres des solutions retenues doivent respecter ces contraintes, faute de quoi ces solutions ne sont pas réalisables. La solution optimale à ce genre de problèmes est connue comme l'optimum global.

Ce travail porte sur la résolution de problèmes d'optimisation combinatoire par des systèmes artificiels auto-organisés. Cette approche est motivée par le fait que de tels systèmes peuvent résoudre des problèmes complexes, bien que l'intelligence d'un seul de leurs composants soit limitée. Cette auto-organisation permet l'émergence de nouvelles fonctionnalités et des propriétés, jusque là inconnues au sein du système. La propriété recherchée dans notre contexte est la faculté à converger vers l'optimum global, i.e. la meilleure solution possible, celle ayant un coût optimal.

Toujours est-il que l'inspiration pour la création de nouvelles méthodes en ingénierie provient d'horizons très divers des mathématiques les plus abstraites aux sciences sociales, en passant par la biologie. L'étude de phénomènes réels est d'ailleurs une source fertile d'inspiration en ingénierie informatique, ou l'étude et la modélisation des systèmes complexes sont très présentes.

En recherche opérationnelle, et plus précisément dans le domaine de l'optimisation difficile, la majorité des méthodes sont inspirées par de telles études, et notamment par la biologie. Le fait que la biologie étudie souvent des systèmes présentant des comportements dits « intelligents » n'est pas étranger au fait qu'ils soient modélisés, puis transposés dans le cadre de problèmes réels. On parle parfois d'intelligence artificielle biomimétique ou bio-inspirée pour désigner de telles approches.

Parmi les domaines de la biologie fertiles en inspiration, l'éthologie (étude du comportement des animaux) a fortement donné lieu à plusieurs avancées significatives, dont la conception de systèmes de fourmis artificielles. Ces systèmes sont notamment étudiés en robotique, en classification ou encore en optimisation. On rencontre souvent le terme d' « intelligence en essaim » pour désigner ces systèmes, où l'intelligence du système entier est plus grande que celle de la simple somme de ses parties.

Cette approche a donné lieu à la création de nouvelles métaheuristiques. Les métaheuristiques forment une famille d'algorithmes d'optimisation visant à résoudre des

problèmes d'optimisation difficile, pour lesquels on ne connaît pas de méthodes classiques plus efficaces. Elles sont généralement utilisées comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans l'algorithme employé. Les algorithmes de colonies de fourmis forment ainsi une classe de métaheuristique fortement exploitées pour la résolution de problèmes d'optimisation difficile combinatoires.

Le comportement social des fourmis constitue un formidable modèle bio-inspiré d'auto-organisation. Dans cette optique, nous proposons dans ce travail, une nouvelle métaheuristique s'inspirant de l'auto-organisation chez l'espèce de fourmis néotropicale Pachycondyla apicalis. Ceci revient à utiliser la métaphore de cette espèce de fourmis pour concevoir des métaheuristiques d'optimisation adaptées aux problèmes combinatoires.

Les objectifs que nous nous sommes fixés consistent à comprendre et isoler les mécanismes intéressants de cette métaphore, utiliser une approche de conception de métaheuristiques suivant cette optique, et appliquer les algorithmes ainsi conçus à un problème d'optimisation.

Pour concevoir cette métaheuristique que nous avons appelé PAO pour Pachycondyla apicalis Ant Optimization, on a eu recours à un cadre de conception permettant de tracer le fonctionnement des métaheuristiques les plus efficaces. Ce cadre est celui de la programmation à mémoire adaptative.

Ainsi, l'élaboration de notre métaheuristique fut de créer un système multi-agents, où la communication joue un rôle central, en tant que processus permettant l'émergence d'un comportement global cohérent du système. La programmation à mémoire adaptative permet quant à elle de cerner les composants principaux d'une métaheuristique itérative, et donne ainsi des guides pour la conception de nouvelles méthodes. Les concepts de mémoire, de diversification et d'intensification sont donc, en un sens, des buts à atteindre pour construire une métaheuristique. Ces buts sont atteints via les concepts apportés par l'auto-organisation, qui décrit alors une voie de conception, où les interactions locales et les rétroactions sont centrales. Cette approche part donc, en quelque sorte, du bas (le comportement des fourmis) vers le haut (le comportement de la métaheuristique).

Notre métaheuristique a été implémentée en utilisant la plate-forme logicielle Paradiseo (PARAllel and DIStributed Evolving Objects). Cette dernière est dédiée à la réutilisation de la conception et à l'implémentation de métaheuristiques. Nous avons proposé Paradiseo-PAO comme un nouveau paquetage logiciel regroupant la métaheuristique PAO ainsi que plusieurs de ses variantes.

Comme validation de notre travail, nous avons mené des expérimentations sur le problème du voyageur de commerce (TSP : Travelling Salesman Problem). Ce problème est bien connu dans la littérature, et plusieurs approches basées sur les fourmis artificielles ont été portées sur ce dernier. Ceci nous a permis de faire une comparaison sur la qualité des solutions obtenues par notre approche avec celles d'autres métaheuristiques.

Le présent document est structuré comme suit :

- Nous rapporterons, au cours du premier chapitre, des définitions formelles de l'émergence et l'auto-organisation. Nous verrons des exemples de systèmes artificiels auto-organisés exhibent des comportements émergents.

- Dans le deuxième chapitre, nous exposons un ensemble de métaheuristiques d'inspiration biologique, mettant en avant la théorie de l'auto-organisation. Par ailleurs, Nous décrirons chaque métaheuristique selon le concept de programmation à mémoire adaptative. Ce concept nous guidera par la suite pour concevoir la notre.
- Nous proposons dans le troisième chapitre la métaheuristique d'optimisation par les fourmis Pachycondyla apicalis (PAO : Pachycondyla apicalis Ant Optimization). Ce chapitre décrit la modélisation du comportement social de cette espèce de fourmis. Également, la transposition faite de la métaphore éthologique à la méthode de résolution sera présentée.
- Le chapitre quatre expose et analyse les expérimentations menées comme validation de notre approche. Le réglage de paramètres et l'implémentation de notre approche seront également abordés. Par ailleurs, une comparaison est donnée avec d'autres métaheuristiques basées sur les fourmis artificielles.

Enfin, nous terminerons par une conclusion récapitulant nos contributions et proposant des perspectives pouvant enrichir nos travaux.

Chapitre 1

Émergence et auto-organisation dans les systèmes artificiels

1.1 Introduction

L'évolution rapide et constante des matériels informatiques, des logiciels et des réseaux de communication nécessite des approches nouvelles pour la conception de systèmes informatiques. Les futurs systèmes qui en résultent doivent permettre d'augmenter la prise en compte de la complexité, être plus robustes et plus autonomes. Il semble évident que pour répondre à ces futurs besoins, les techniques informatiques actuelles sont inadéquates.

De nouvelles approches, techniques et théories, doivent être recherchées dès aujourd'hui afin de pouvoir répondre aux besoins logiciels de demain. De nombreux chercheurs explorent de nouvelles frontières souvent inédites et assez éloignées des disciplines habituellement abordées par les informaticiens. En prenant comme sources d'inspiration les systèmes naturels, nous avons choisi de nous intéresser à celles qui nous semblent les plus encourageantes :*l'émergence* et *l'auto-organisation*.

1.2 La nature comme source d'inspiration

Comme décrit dans [45], généralement en Physique et en Chimie, les interactions entre des éléments simples donnent naissance à des structures complexes dotées de propriétés totalement nouvelles .La Thermodynamique est un exemple frappant de cet état de fait : alors que les molécules et leurs interactions sont parfaitement connues, les modèles des fluides ou liquides sont limités à des conditions fortement contraintes. L'utilisation des processus physiques ou chimiques à l'échelle moléculaire ou atomique pour développer des algorithmes peut permettre d'utiliser ce passage d'un micro niveau connu à un macro niveau aux fonctionnalités beaucoup plus complexes.

En Biologie, de nombreux systèmes naturels composés d'individus autonomes exhibent des aptitudes à effectuer des tâches qualifiées de complexes sans contrôle global. De plus, ils peuvent s'adapter à leur milieu soit pour y survivre, soit pour améliorer le fonctionnement du collectif. C'est le cas des colonies d'insectes sociaux tels que les termites, fourmis, ou araignées qui font effectivement preuve de remarquables capacités pour effectuer des tâches telles que : La construction de nids complexes, la construction de pont, la recherche efficace de ressources, la capture de proies …. L'étude des déplacements collectifs de vols d'oiseaux migrateurs ou de bancs de poissons exhibe également le fait que la tâche collective est le résultat d'interactions entre des individus autonomes.

Parmi tous ces systèmes étudiés, il y a un facteur commun : la dimension émergente des phénomènes. Il parait donc légitime d'étudier ce phénomène afin de pouvoir en comprendre

le fonctionnement ou du moins être capable de l'utiliser pour la conception de systèmes artificiels robustes, autonomes et adaptatifs.

1.3 Historique de l'émergence

Cet historique a été repris dans sa grande partie à partir de la thèse de J-P. Georgé [21] :

1.3.1 L'aire de la Grèce antique

Comme le résument *Ali* et *Zimmer* dans "*The question concerning Emergence*"[1], l'origine de l'émergence pourrait bien être le postulat datant de la Grèce antique : «*le tout est plus que la somme de ses parties*». On retrouve des traces du concept d'émergence et d'auto-organisation dans des écrits de Thalès et Anaximandre.

Dans le même ordre d'idées, Aristote parle du «*tout avant les parties*». Il réfère à l'explication admise de la précédence de l'entité représentant le tout par rapport aux parties sur lesquelles le tout est construit. Il se sert de cette notion pour expliquer certains phénomènes que l'on n'arrive pas à décomposer. Cependant le "*tout avant les parties*" suppose une entité cohérente est pré-donnée alors que l'émergence est une construction dynamique apparaissant au cours du temps.

1.3.2 L'aire du proto-émergentisme

Il faut attendre le milieu du XIXe siècle pour voir apparaître un mouvement de pensée autour du concept de l'émergence. Ce mouvement sera plus tard appelé Proto-émergentisme et se poursuivra jusque dans les années 30. Ses débuts sont surtout caractérisés par la distinction que fait G. H. Lewes entre phénomène résultant et phénomène émergent [39]. Ce philosophe anglais dont les travaux sont basés sur ceux de J. S. Mill, explique en 1875 que pour le résultant la séquence d'étapes qui produisent un phénomène est traçable, alors que pour l'émergent nous ne pouvons pas tracer les étapes du processus. L'incapacité décrite ici peut être de plusieurs natures, certains auteurs [1], y voient une limitation épistémologique[1] de l'observateur : ce dernier n'a alors pas les connaissances nécessaires pour trouver la trace du phénomène observé.

Même si les définitions données jusque ici se placent autour d'un point de vue philosophique, il n'en reste pas moins quelles cherchent à définir l'émergence afin de reconnaître un phénomène émergent et le différencier de phénomènes explicables. Ainsi, le processus d'émergence est vu comme une boîte noire. On ne peut discerner que les entrées de plus bas niveaux et les sorties de plus hauts niveaux. On ne sait pas comment les entrées sont transformées et reliés aux sorties (Figure 1.1.(a)).

1.3.3 L'aire du néo-émergentisme

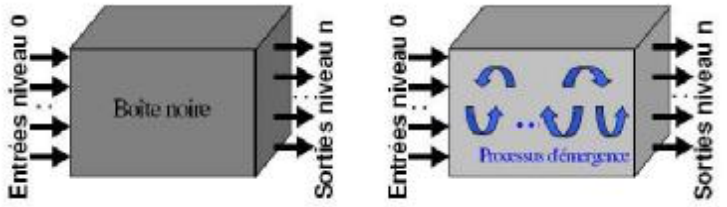
Ce n'est que dans la deuxième moitié du XXe siècle que la science s'est dotée de moyens permettant d'explorer cette boîte noire. On peut appeler la recherche récente sur

[1] Epistémologie : Partie de la philosophie qui étudie l'histoire, les méthodes, les principes des sciences.

l'émergence le *néo-émergentisme*. Elle est liée à la théorie de la complexité actuelle et prend ses racines dans diverses approches comme la dynamique des systèmes en physique, en mathématiques, en thermodynamique, et en informatique.

Nous pouvons maintenant ouvrir la boîte noire grâce aux découvertes de constructions mathématiques pertinentes, et aux nouvelles méthodes de recherche, ainsi que grâce à la puissance des ordinateurs actuels (Figure 1.1.(b)).

Un tel courant vise principalement à élaborer des méthodes visant à expliquer et démystifier le phénomène d'émergence, le rendant moins opaque et par conséquent moins miraculeux. Dans le domaine de l'informatique, ce courant est intimement lié à la théorie de la complexité et des systèmes complexes adaptatifs promulgués par le*Santa Fe Institute*[2].



(a) (b)

Figure 1.1. Déscription de l'émergence : (a) proto-émergentisme, (b) néo-émergentisme

1.4 L'émergence en informatique

Depuis près de trente ans, tout un champ de recherche s'est peu à peu créé autour de l'émergence afin d'en exploiter les caractéristiques si particulières au sein de systèmes informatiques. Ce terme apparaît maintenant régulièrement dans la littérature des systèmes complexes, de la vie artificielle, ou encore des systèmes multi-agents.

Le « *Emergent Computing* » ou le *Calcul émergent*, faisant référence aux travaux de Forrest [18], est le terme employé en général pour désigner cet axe de recherche dans lequel se situent nos travaux.

La computation émergente s'efforce de produire un comportement global intéressant à partir de nombreuses interactions locales. Le principe de base ressemble à celui de la computation parallèle dans le sens où elle s'efforce de décomposer des systèmes complexes en sous-unités indépendantes qui interagissent. Mais là où la computation parallèle requiert une contraignante supervision pour contrôler le bon fonctionnement du système et où les performances sont considérablement inférieures à une fonction linéaire du nombre de processeurs, la computation émergente va exploiter ces interactions. On obtient ainsi un gain d'efficacité, de flexibilité et une représentation plus naturelle.

L'utilisation de l'émergence s'est amplement répandue dans le domaine de l'informatique. De nombreuses disciplines ou champs d'étude manipulent le terme « *émergence* », tels que la vie artificielle, le calcul évolutionnaire (Algorithme Génétiques et programmation génétique) les réseaux neuronaux, les algorithmes de fourmis, les automates cellulaires, les systèmes multi-agents, la Simulation (systèmes physiques, chimiques, biologiques, sociaux),...

Nous retraçons de manière chronologique dans ce qui suit, quelques propositions données sur la théorie de l'émergence au sein des systèmes artificiels :

En 1977, John Holland précurseur des algorithmes évolutionnaires, publie un livre intitulé ''*Emergence from Chaos to Order''*[28]. Il insiste alors sur l'apparition dans des systèmes complexes de motifs dignes d'intérêt qu'il propose d'appeler des propriétés émergentes. Cependant, l'auteur se garde de donner une définition, préférant suggérer une intuition.

Chritopher Langton décrit dans [38], l'émergence ainsi : "*Un système est émergent s'il y a apparition de motifs au niveau macro conditionnés par la dynamique du niveau micro, ces motifs contraignent la dynamique micro*" Nous avons ici la première définition utilisant explicitement l'idée de niveaux. Il existe deux niveaux évidents pour un système composé : le niveau global qui correspond au système entier, et le niveau local correspondant aux composants[3]. Langton décrit donc ici une boucle d'influence entre deux niveaux ou plus.

En 1990, Stephanie Forrest dans [18], définit un calcul émergent comme constitué d'une collection d'agents, chacun suivant des instructions explicites (niveau microscopique) des interactions entre ces agents (dirigées par les instructions) qui aboutissent à des motifs implicites et globaux au niveau macroscopique, c'est à dire des épiphénomènes[4], et l'interprétation naturelle de ces épiphénomènes comme calculs. Dans cette définition, on peut noter une différence avec celle de Langton : là où il décrivait des motifs ayant une influence sur le niveau micro, Forest décrit des épiphénomènes qui sont des phénomènes ne contraignant pas la dynamique. La question de l'influence du phénomène sur l'évolution du système n'est donc pas abordée dans la dernière définition.

De son côté, Müller est parti d'une adaptation de la définition précédente. Dans [43, 40], il explique qu'un phénomène est émergent si :

- Il y a un ensemble d'entités en interaction dont la dynamique est exprimée dans un vocabulaire ou théorie D distincte du phénomène émergent à produire.
- La dynamique de ces entités inter-agissantes produit un phénomène global qui peut être un état structuré stable ou même la trace d'exécution.
- Ce phénomène global peut être observé et décrit dans un vocabulaire ou théorie D' distincte de la dynamique sous-jacente.

Selon Müller, pour que le dernier point soit possible, le phénomène global doit être globalement perçu et donc inscrit sur un support. Dans les systèmes naturels, l'environnement joue ce rôle important de médium d'inscription. Un exemple naturel est la planification d'un chemin par les fourmis. Les entités en interaction sont ici les fourmis. Une fourmi qui porte de la nourriture à son nid dépose une trace de phéromone dont la *dissipation* produit un gradient d'odeur qui est perçu par les autres fourmis. Le phénomène

[2] http://www.santafe.edu

[3] Un composant ici est la même chose qu'une partie du système, les deux termes sont fréquemment utilisés, on peut également utiliser le terme d'agent.

[4] Epiphénomène : un phénomène secondaire, sans importance par rapport à un autre .Ce qui s'ajoute à un phénomène sans réagir sur lui.

global est le va et vient des fourmis mais qui peut seulement être vu par un observateur de la colonie de fourmis. Il peut alors décrire le phénomène global en termes de chemin (théorie *D'* de nature géométrique). Les interactions entre les fourmis et l'environnement ne sont pas exprimées en termes de chemin mais en terme de gradient local de phéromones (théorie D de nature chimique). Les fourmis produisent une structure stable et globale qui devient un chemin aux yeux de l'observateur.

Plus récemment, Jean-Pierre Georgé propose dans [21], une définition « utilitaire » de l'émergence pour les systèmes artificiels. Il l'a décomposé en trois parties : ce que l'on veut faire émerger, à quelle condition il y a émergence, et comment nous nous en servons.

- Objet : Un système informatique a pour finalité de réaliser une fonction adéquate à ce que l'on attend de lui. C'est cette fonction, pouvant évoluer au cours du temps, que nous voulons faire émerger.
- Condition : Cette fonction est émergente si le codage du système ne dépend aucunement de la connaissance de cette fonction. Ce codage doit contenir des mécanismes permettant l'adaptation du système au cours de ses échanges avec l'environnement afin de tendre à tout instant vers la fonction adéquate.
- Méthode : Pour changer de fonction, il suffit de changer l'organisation des composants du système. Ces mécanismes sont spécifiés par des règles régissant l'auto-organisation entre les composants et ne dépendant pas de la connaissance de la fonction collective.

Selon cet auteur, le premier point de cette définition sert à identifier sur quoi porte l'émergence. Ici, la fonction du système doit émerger. Cette définition pragmatique vise à écarter l'ambigüité de l'émergence de formes « patterns » comme dans les automates cellulaires. Le deuxième point souligne la caractéristique d'irréductibilité des systèmes émergents. En effet, si la fonction globale du système est explicitement connue par les parties du système, la théorie de description du macro niveau est déductible des fonctions de micro niveau. Enfin, le troisième point d'ordre méthodologique, propose de voir l'auto-organisation comme moyen d'obtenir des phénomènes émergents.

Il reste maintenant à *opérationnaliser* ce concept d'émergence pour en faire une méthodologie systématique de conception de systèmes multi-agents (le niveau micro) à partir d'une spécification de leur(s) finalité(s) (le niveau macro).

Nous nous sommes appuyés sur cette démarche le long de nos travaux, afin de concevoir et élaborer les différentes métaheuristiques proposées dans ce mémoire de recherche.

1.5 Propriétés de l'émergence

Le concept d'émergence est important pour décrire comment un phénomène non-compositionnel[5] au niveau macro repose sur une structure d'interaction à un niveau micro.

[5] Non compositionnel : dans le sens que les descriptions au niveau macro ne sont pas une simple composition des propriétés individuelles au niveau micro.

Comme rapporté dans [21, 22], on trouve dans la littérature plusieurs particularités attribuées à l'émergence, à savoir :

- *Nouveauté et imprévisibilité*: l'émergence présuppose qu'il y a apparition de nouveautés propriétés, structures, formes ou fonctions, et d'autre part, elle implique qu'il est impossible de décrire, d'expliquer ou de prédire ces nouveaux phénomènes en terme physique à partir des conditions de bases définies aux niveaux inférieurs.
- *Temporalité* : le phénomène doit se construire dans le temps, il ne peut être instantané ;
- *Cohérence et corrélation*: le phénomène a une identité propre mais liée d'une manière ou d'une autre aux parties de micro-niveau ;
- *Irréductibilité* : Ceci implique l'impossibilité de déduire, à partir des propriétés du micro-niveau, les propriétés du macro-niveau. On ne peut pas réduire le phénomène à la somme de ses parties ;
- *Interdépendance des niveaux*: Langton dans [38], définit l'émergence en termes de relation de feed-back (rétroactions) entre les niveaux dans un système dynamique. Les micro-dynamiques locales causent les macro-dynamiques, et les macro-dynamiques globales contraignent les locales. Dans cette dynamique, la prise en compte du couplage (environnement, système) est importante, étant donné que l'environnement joue le rôle de moteur et d'acteur d'adaptation et d'évolution du système.

1.6 L'auto-organisation comme processus de l'émergence

Les traits vus précédemment (section 1.5) permettent d'identifier un phénomène émergent, mais du point de vue méthodologique, ils ne constituent pas une base suffisante pour construire des systèmes*émergents.*

Par contre, plusieurs caractéristiques communes à ces systèmes peuvent être analysées afin de guider la conception d'applications reposant sur la notion d'émergence. Parmi ces caractéristiques*l'auto-organisation.*

Nos travaux concernent particulièrement la mise en œuvre de l'auto-organisation en tant que mécanisme et processus permettant d'aboutir à la construire de systèmes émergents.

L'auto-organisation est un processus décrit dans plusieurs disciplines, notamment en physique [45], et en biologie [6]. En général, elle est associée à la notion d'émergence, c'est un processus convergeant vers le phénomène émergent.

Etant donné que les métaheuristiques présentées dans ce mémoire sont inspirées de phénomènes biologiques, nous avons choisi de considérer une définition dans ce cadre. Une définition a été proposée dans [6] dont la traduction est comme suit :

« L'auto-organisation caractérise un processus au cours du quel une structure émerge au niveau global uniquement d'un grand nombre d'interactions entre les composants de niveau local du système. De plus, les réglés spécifiant les interactions entre composants du système sont suivies en utilisant uniquement des informations locales, sans référence au modèle global ».

Un terme à retenir dans cette définition pour une bonne compréhension, « structure ». Ce mot est une traduction approximative du mot anglais « pattern », qui désigne la notion de modèle, et qui peut signifier aussi, configuration générale, forme, ou bien schéma. Il s'applique généralement à un arrangement organisé d'entités dans l'espace et dans le temps.

Selon Varela [50], ceci signifie que ce groupe acquière une autonomie, à la fois par rapport à ses constituants, et par rapport au substrat physique (à l'environnement). La structure d'un groupe d'entités (agents) est donc un lieu autonome de transformation, dont la transformation s'exprime comme les interactions entre les agents qui la composent. Ces interactions conduisent ce système à produire des structures, des comportements ou des propriétés spécifiques non dictés (non explicites) par l'extérieur. C'est ce qu'on désigne par phénomène *émergent* (Figure 1.2).

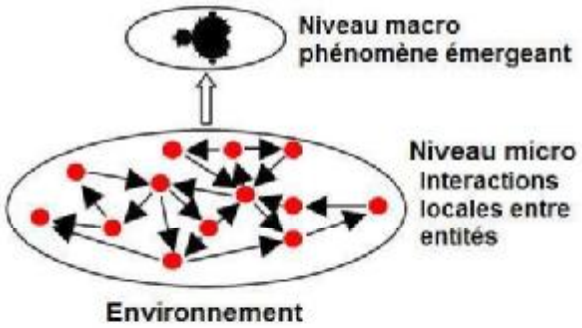


Figure 1.2. Émergence de propriétés à partir d'interactions locales

La question qui se pose est donc de comprendre comment les composants d'un système interagissent entre eux pour produire une structure complexe (i.e. plus complexe que les composants eux-mêmes). Un certain nombre de phénomènes nécessaires ont été identifiés: ce sont les *processus de rétroaction* et la *gestion des flux d'informations*.

Les rétroactions positives sont des processus dont le résultat renforce l'action, par exemple par amplification, facilitation, auto-catalyse[6], etc. Les rétroactions positives sont capables d'amplifier les variations du système, permettant la mise à jour d'informations peu apparentes. De tels processus peuvent facilement entrainer une « explosion » du système, s'ils ne sont pas maintenus sous contrôle par des rétroactions négatives, qui jouent ainsi le rôle de stabilisateurs du système. Lorsqu'ils sont couplés, de tels processus de rétroaction sont de puissants générateurs de modèles [3].

Certaines caractéristiques des systèmes auto-organisés sont particulièrement intéressantes, en particulier leur dynamisme, ou encore leur capacité à produire des modèles stables. Dans le cadre de l'étude du comportement des insectes sociaux, certains concepts liés au principe de l'auto-organisation méritent d'être soulignés : la décentralisation intrinsèque de ces systèmes, leur organisation en hétérarchie dense et l'utilisation récurrente de la stigmergie. En effet, ces concepts sont parfois utilisés comme

[6] Catalyse : Accélération d'une réaction chimique par une substance (catalyseur) qui intervient dans la réaction et qui est régénérée à la fin de celle-ci.

autant d'angles de vue différents sur un même problème et recouvrent une partie des mécanismes de l'auto-organisation.

1.6.1 Stigmergie

Dans le cadre de la biologie du comportement, il est aisé de comprendre que les interactions entre les composants d'un système vont souvent mettre en jeu des processus de communication, et de transfert d'informations entre individus.

D'une manière générale, les individus peuvent communiquer directement entre eux par le biais de "signaux", soit par le biais de modification de l'environnement [6].

La stigmergie est précisément définit comme une ''forme de communication passant par le biais de modifications de l'environnement'', mais on peut rencontrer le terme ''interactions sociales indirectes'' pour décrire le même phénomène. Un exemple d'utilisation de la stigmergie est décrit dans la section 1.7.2.

1.6.2 Contrôle décentralisé

Dans un système auto-organisé, il n y a pas de prise de décision à un niveau donné, suivie d'ordres et d'actions prédéterminées. En effet, dans un système décentralisé, chaque individu dispose d'une vision locale de son environnement, et ne connaît donc pas le problème dans son ensemble.

La littérature des systèmes multi-agents emploie souvent ce terme ou celui « d'intelligence artificielle distribuée » [29], bien que, d'une manière générale, l'étude des systèmes multi-agents tende à utiliser des modèles de comportement plus complexes, fondés notamment sur les sciences de la cognition.

Les avantages d'un contrôle décentralisé sont notamment la *robustesse* et la *flexibilité* [4]: systèmes robustes, car capables de continuer à fonctionner en cas de panne d'un de leurs composantes ; flexibles, car efficaces sur des problèmes dynamiques.

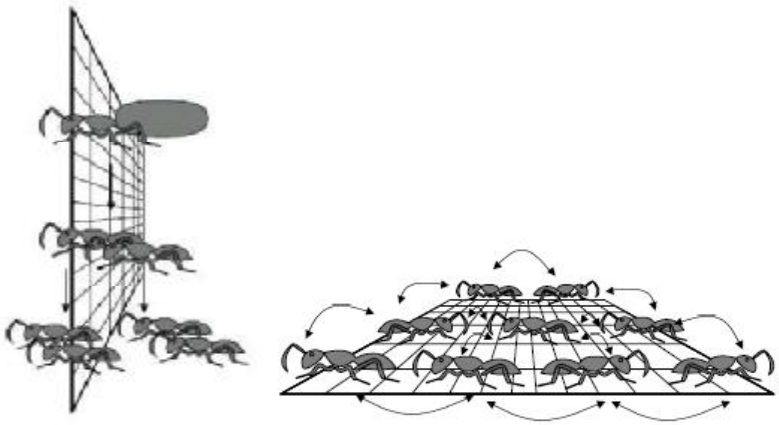
1.6.3 Hétérarchie dense

L' hétérarchie dense est un concept issu directement de la biologie, utilisé pour décrire l'organisation des insectes sociaux, et plus particulièrement des colonies de fourmis. La définition donnée dans [51], peut être traduite comme suit :

Une colonie de fourmis est une variante particulière de hiérarchie qui peut avantageusement être appelée une hétérarchie. Cela signifie que les propriétés des niveaux globaux agissent sur les niveaux locaux, mais que l'activité induite dans les unités locales influence en retour les niveaux globaux.

L' hétérarchie est dite *dense* dans le sens où un tel système forme un réseau hautement connecté, où chaque individu peut échanger des informations avec n'importe quel autre. Ce concept est en quelque sorte opposé a celui de hiérarchie, où dans une vision populaire mais erronée, la reine gouvernerait ses sujets en faisant passer des ordres dans une structure verticale, alors que dans une hétérarchie, la structure est plutôt horizontale (Figure 1.3).

On constate que ce concept rejoint celui de contrôle décentralisé, mais aussi celui de stigmergie, en ce sens que l'hétérarchie décrit la manière dont le flux d'information parcourt le système. Cependant, tout type de communication doit être pris en compte, tant la stigmergie que les échanges directs entre individus.



(a) (b)

Figure 1.3.Hiérarchie (a) et hétérarchie dense (b) : deux concepts opposés

1.7 Exemples de systèmes artificiels émergents

1.7.1 Automates cellulaires

Les Automates Cellulaires ont été inventés par deux des plus renommés mathématiciens du siècle dernier, Stanislas Ulam et John von Neumann [44]. Ils sont remarquables par leur simplicité. Le «*Jeu de la Vie* » est une des instances les plus étudiées de ces automates cellulaires [20].

Il suffit de concevoir une grille, comme pour un jeu de Dames, où chaque case peut prendre la valeur 0 ou 1. Le 0 signifiant que la case est « morte », le 1, qu'elle est « vivante ». Ensuite, chaque « cellule » ainsi définie, possède quelques règles très simples de fonctionnement qui régissent son évolution :

- Si une cellule morte a exactement 3 voisines vivantes, elle devient vivante.
- Une cellule vivante survit si elle a exactement 2 ou 3 voisines vivantes.
- Dans tous les autres cas, la cellule meurt ou reste morte (i.e. 4 ou plus, elle est étouffée, 1 ou 0, elle meurt d'isolement).

Que peut-on faire avec des règles aussi simples ?. En fait, si on pose certaines configurations de cellules sur une grille, on peut observer des phénomènes surprenants. L'exemple le plus populaire reste le « planeur » (Figure 1.4). En faisant évoluer cette configuration de cellules rapidement on observe un planeur qui suit une trajectoire descendante, en regagnant sa forme d'origine toutes les 4 étapes.

Il existe un grand nombre de ces configurations remarquables. Elles ont toutes en commun le fait d'être régies par les mêmes règles de base et que le résultat observable interprété ne peut être prédit.

Dans une vision plus informatique des choses, la théorie veut que l'on puisse ainsi simuler une Machine de Turing, les signaux transitant sous forme de groupes de cellules comme le planeur, et venant modifier l'état de certaines configurations stables.

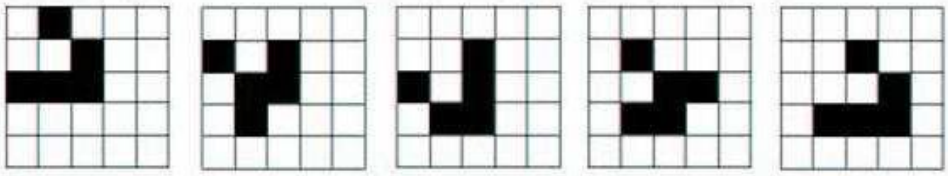


Figure 1.4. Évolution du planeur sur 5 étapes. On retrouve la même configuration de cellules à l'étape 5, qu'à l'étape 1, mais elles sont décalées vers le bas et vers la droite.

1.7.2 Algorithmes de colonie de fourmis

Il existe de nombreux phénomènes émergents issus du comportement des insectes sociaux tels que les fourmis ou les termites. Cela n'a rien de surprenant car si l'on prend par exemple une colonie de fourmis, ce système, bien évidement complexe, contient tous les ingrédients pour des phénomènes émergents : grand nombres d'entités autonomes, nombreuses interactions, fonctionnement simple des entités, capacités perceptives et cognitives limitées, système confronté à un environnement dynamique, fonctionnements de haut niveau remarquables et nécessaires à la survie. Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation : les insectes sociaux en général, et les fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Dans cette société d'insectes, on peut observer l'apparition de phénomènes de haut niveau qui ne sont pas décrits dans le fonctionnement des parties.

Parmi les problèmes étudiés par les biologistes, les problèmes de choix lors de l'exploitation de sources de nourriture. Certaines espèces de fourmis ont la particularité d'employer la stigmergie pour communiquer via des substances volatiles appelées phéromones. Elles sont attirées par ces substances qu'elles perçoivent. Les fourmis peuvent déposer des phéromones au sol, et forment ainsi des pistes odorantes, qui pourront être suivies par leurs congénères.

Les fourmis utilisent les pistes de phéromone pour marquer leur trajet, par exemple entre le nid et une source de nourriture. Au départ, un grand nombre de fourmis se déplacent à l'extérieur du nid, plus ou moins au hasard. En recherchant de la nourriture, elles déposent une légère trace de phéromones tout le long de leur chemin. Si l'une d'entre elles découvre une ressource quelconque, elle retourne au nid en déposant une trace beaucoup plus intense (cette intensité dépend éventuellement de la richesse de la ressource). Cette trace tend à attirer les congénères qui, en la suivant, vont parvenir à la nourriture. Ils vont alors retourner au nid et renforcer la trace à leur tour.

On assiste ainsi à la mise en place d'une boucle de rétroaction positive : la trace attire des individus qui renforcent la trace qui attire donc plus d'individus. Les fourmis tendent à suivre les pistes de phéromones qu'elles rencontrent, mais il ne s'agit que d'une tendance. A tout moment, la probabilité existe qu'un individu quitte la trace puis se déplace plus ou moins au hasard. A cette occasion, la fourmi égarée peut éventuellement découvrir une source de nourriture beaucoup plus riche que celle qu'exploitent ses congénères. En

déposant alors une trace de phéromone plus intense encore, elle va les attirer vers cette nouvelle ressource, formant une nouvelle boucle de rétroaction positive (Figure 1.5). Enfin, quand la satiété produit son effet, ou que la ressource est épuisée, une boucle de rétroaction négative se met en place. Dans ce cas, si l'on considère que les traces de phéromones s'évaporent assez rapidement, une fois que la nourriture sera épuisée, de moins en moins de fourmis auront tendance à suivre la trace qui va finir par disparaître.

Cet exemple est intéressant par sa simplicité car il permet de bien imaginer le fonctionnement du système et donc de comprendre qu'il est logique que les fourmis finissent par choisir le chemin le plus court. Mais pourtant, lorsque l'on analyse seulement les règles des fourmis, on n'y reconnaît pas la recherche du plus court chemin. Ce n'est qu'en considérant le système dans sa globalité que ce phénomène s'explique. Il y bien une différence de niveau micro/macro et on peut parler de phénomène émergent. Les algorithmes inspirés des colonies de fourmis ont été appliqués, aussi bien pour, l'optimisation combinatoire [17], la classification [10], ou pour le routage de paquets sur les réseaux de communication [7].

Figure 1.5. Le déplacement en masse des fourmis

1.7.3 Réseaux de neurones

Un RNA (Réseau de Neurones Artificiels) est un ensemble de neurones formels (d'unités de calcul simples, de nœuds processeurs) associés en couches (ou sous-groupes) et fonctionnant en parallèle [25].

Dans un réseau, chaque sous-groupe fait un traitement indépendant des autres et transmet le résultat de son analyse au sous-groupe suivant. L'information donnée au réseau va donc se propager couche par couche, de la couche d'entrée à la couche de sortie, en passant par une, ou plusieurs couches intermédiaires (dites couches cachées). Il est à noter qu'en fonction de l'algorithme d'apprentissage, il est aussi possible d'avoir une propagation de l'information à reculons (back propagation). Habituellement (excepté pour les couches d'entrée et de sortie), chaque neurone dans une couche est connecté à tous les neurones de la couche précédente et de la couche suivante.

Les RNAs ont la capacité de stocker de la connaissance empirique et de la rendre disponible à l'usage. Les habilités de traitement (et donc la connaissance) du réseau vont être stockées dans les poids synaptiques, obtenus par des processus d'adaptation ou d'apprentissage. En ce sens, les RNAs ressemblent donc au cerveau car non seulement, la connaissance est acquise au travers d'un apprentissage mais de plus, cette connaissance est stockée dans les connexions entre les entités, soit dans les poids synaptiques.

Les réseaux de neurones sont un exemple typique d'un système complexe dans le quel on a la faculté d'apprentissage qui émerge au niveau global par des interactions au niveau local entre les neurones. Ces interactions permettent l'adaptation des poids synaptiques, en vue d'obtenir l'activité requise au niveau de la synapse de sortie (Figure 1.6).

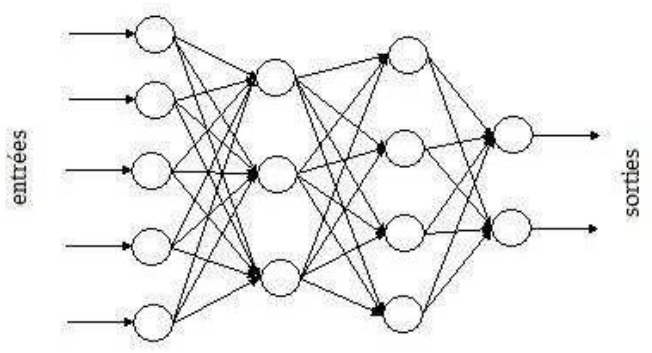


Figure 1.6. Vue simplifiée d'un réseau de neurones artificiels

1.7.4 Un système complexe pour la segmentation d'images

Cette section décrit les activités de recherche, et les contributions que nous avons réalisées depuis peu autour des systèmes complexes, de l'émergence, et de l'auto-organisation.

Dans l'article [37], nous avons proposé un système complexe dont la fonctionnalité émergente au niveau global est la segmentation d'images. Ce système est constitué d'entités autonomes qui ont la capacité d'estimer l'homogénéité d'une région à partir de leur emplacement. Chaque entité exhibe plusieurs comportements réactifs en réponse au stimulus local. Elle peut migrer, se reproduire, ou bien se diffuser au sein de l'image. Différentes entités explorent l'image et étiquettent les pixels lorsqu'ils appartiennent à des segments homogènes. Les interactions entre les entités permettent l'émergence de nouvelles propriétés qui sont : la recherche, la localisation et l'étiquetage des segments homogènes. On obtient un phénomène global émergent qui est un système de segmentation. La Figure 1.7 présente les résultats obtenus par notre système appliqué sur des IRM cérébrales

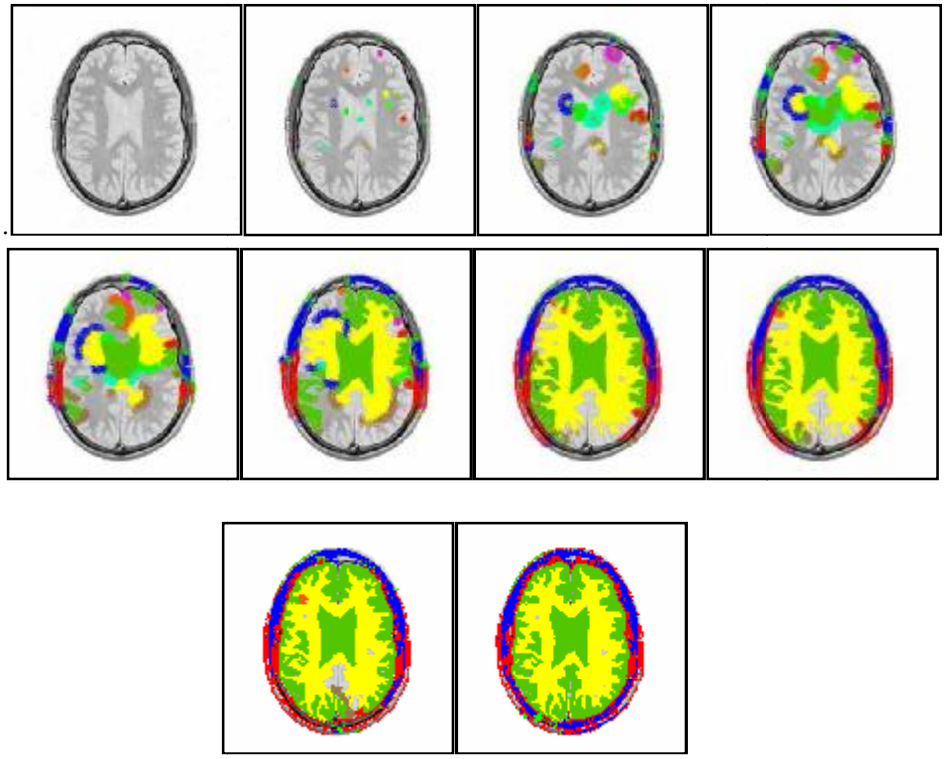


Figure 1.7.Déroulement du processus de segmentation d'une image IRM par notre système multi-agents

Par ailleurs, nous avons repris notre système multi -agents afin de l'adapter à la tâche d'extraction de contours [36]. Cette tâche constitue une phase préliminaire pour la reconnaissance de formes. La progression du processus d'extraction sur différ ents type images est donnée dans la Figure 1.8 et la Figure 1.9.

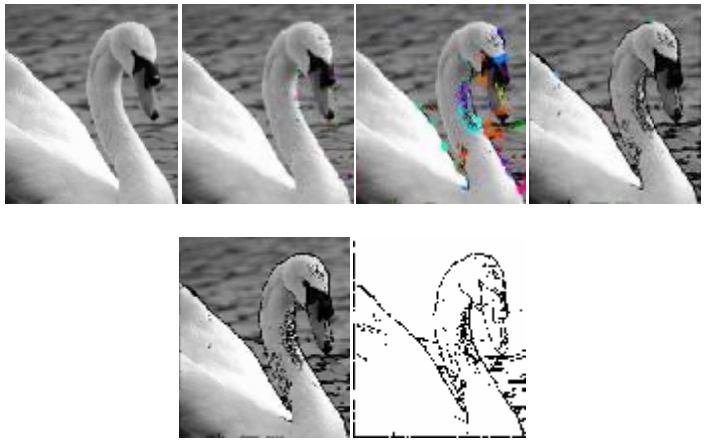


Figure 1.8. Étapes d'extraction de contours sur une entité

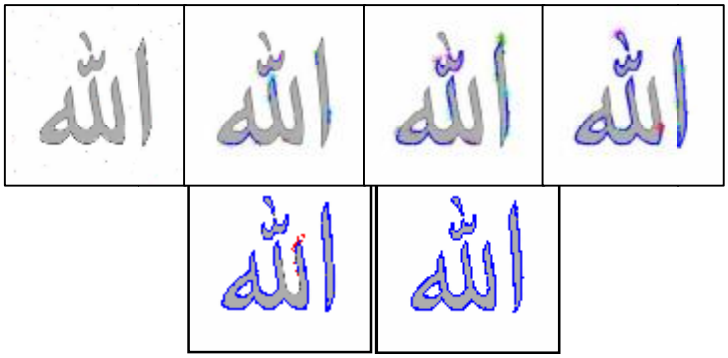


Figure 1.9. Déroulement de l'extraction de contours sur des car actères arabes

Enfin, le système proposé fait partie depuis peu des modèles de la plate-forme NetLogo[7]. Cette plate-forme est dédiée principalement à l'élaboration t au développement de systèmes multi-agents réactifs.

1.8 Conclusion

Dans ce chapitre nous avons tenté de cerner les notions d'émergence et d'auto-organisation au sein des systèmes complexes. Ces notions sont de plus en plus répandues dans le domaine informatique et notamment dans les systèmes multi-Agents.

En effet, l'objectif du domaine est l'étude et la conception de systèmes complexes ayant un comportement global produit par l'interaction de nombreux agents ayant un comportement autonome. Ce objectif rejoint les problématiques étudiées déjà par d'autres disciplines telles que la philosophie, la physique, la biologie, la thermodynamique, et la cybernétique.

En général, l'auto-organisation est associée à la notion d'émergence, c'est un processus permettant l'obtention de phénomènes émergents. Elle est donc définie comme un processus spontané d'ordre dans un système, dû à des relations internes au système et/ou à des relations avec son environnement, et à la manifestation de ces relations dans la durée du temps. Une définition intuitive de l'auto-organisation considère qu'il y a auto-organisation quand le système change son organisation sans commande externe explicite. Elle conduit le système à produire des structures, des comportements ou des propriétés non dictées par l'extérieur.

Le prochain chapitre abordera comment la théorie de l'auto-organisation est mise en place dans certaines métaheurstiques à base de population. En combinant cette théorie avec le concept de programmation à mémore adaptative, il est possible d'arriver à concevoir des métaheuristiques efficaces, flexibls, et adaptative

[7] http://ccl.northwestern.edu/netlogo

Chapitre 2

Métaheuristiques et Programmation à Mémoire Adaptative

2.1 Introduction

L'optimisation est un sujet central en recherche opérationnelle, un grand nombre de problèmes d'aide à la décision peuvent en effet être décrits sous la forme de problèmes d'optimisation. Les problèmes d'ordonnancement, l'apprentissage supervisé de réseaux de neurones, ou encore la recherche du plus court chemin sont par exemple des problèmes d'optimisation. Les méthodes génériques ou méthode de recherche globale, aussi appelées métaheuristique connaissent un réel succès pour traiter les problèmes NP-difficiles. Ces problèmes sont des problèmes pour lesquels aucun algorithme polynomial n'a, à ce jour, été découvert.

Un cadre général pouvant regrouper les métaheuristiques les plus compétitives et celui de la programmation à mémoire adaptative. Dans cette course vers l'optimalité, on observe que les méthodes les plus efficaces font basculer l'algorithme entre intensification et diversification de la recherche, en s'appuyant sur une mémoire représentant l'information récoltée par l'algorithme.

Par ailleurs, plusieurs de ces métaheuristique et notamment les métaheuristique à base de population font ressortir l'émergence et l'auto-organisation dans leur fonctionnement. Des entités simples qui interagissent au niveau micro, et dont l'ensemble des interactions permettent de générer un comportement au niveau macro se traduisant en la recherche de l'optimum global de la fonction objectif.

Dans ce chapitre, nous mettons en relation la théorie de l'auto-organisation et le concept de programmation à mémoire adaptative. Au sens où, la combinaison entre la théorie de l'auto-organisation et la programmation à mémoire adaptative, donne les clefs pour mettre en œuvre un nouveau paradigme pour la conception de métaheuristiques relevant de l'intelligence en essaim.

2.2 Optimisation combinatoire et méthodes de résolution

De nombreux domaines de l'industrie, comme la mécanique, la chimie, les télécommunications, la logistique ou les transports, sont confrontés à des problèmes d'optimisation complexes. De nombreuses recherches sont menées afin de développer des méthodes efficaces pour résoudre ces problèmes, c'est-à-dire trouver une solution de bonne

qualité. La qualité est le plus souvent exprimée par une fonction mathématique qui modélise typiquement le coût de la solution.

Un problème d'optimisation consiste à rechercher la valeur maximale ou minimale, appelée optimum global d'une fonction $f: S \to \Re$. f est appelée la fonction objectif.

S est couramment appelé espace décisionnel et le plus souvent on utilise $S = \Re^m$. Un élément $x = (x_1, \ldots, x_m)$ de S est appelé une solution du problème d'optimisation. Les composantes x_i de x sont les variables de décision. C'est en faisant varier les valeurs de ces variables que l'on modifie la valeur de la fonction objectif.

Une solution $x \in S$ est dite réalisable si elle respecte un ensemble C de contraintes d'égalité ou d'inégalité. L'ensemble $\Omega \subseteq S$ est l'ensemble des solutions réalisables que l'on appelle ensemble réalisable. Etant donné que maximiser une fonction f est équivalent à minimiser la fonction $-f$, nous considérons dans ce manuscrit que la fonction doit être minimisée.

Formellement, un problème d'optimisation s'écrit donc :

$$(PO) = \begin{cases} min\, f(x) \\ sous - contrainte\ C \end{cases}, x \in S. \qquad (2.1)$$

L'optimum global d'un problème d'optimisation peut alors se définir ainsi [31]:

Pour $\Omega \neq \emptyset$, on note $f^* = f(x^*)$ avec $f^* > -\infty$ et $x^* \in \Omega$ si :

$\forall x \in \Omega,\ f^* \leq f(x)$, f^* est appelé l'optimum global de (Ω, f).

Il est à noter que x^* peut ne pas être unique ; cependant il n'existe qu'un unique optimum global f^*. Dans le cas où n le nombre de fonctions objectifs à optimiser est égal à 1, on parle d'optimisation mono-objectif. Si le nombre d'objectifs $n >= 2$, alors on parle d'optimisation mulit-objectifs. Dans ce cas, les objectifs sont généralement contradictoires, i.e. l'amélioration de la valeur d'un objectif entraîne la dégradation des valeurs des autres et inversement.

Les problèmes d'optimisation combinatoire du domaine académique ou du monde réel (issus de domaines tels que les transports, les télécommunications, la génomie,...), sont en majorité NP-difficiles, signifiant qu'il n'existe pas d'algorithmes produisant une solution optimale en un temps d'exécution polynômial par rapport à la taille des données manipulées.

On distingue deux grandes familles de méthodes de résolution (Figure 2.1) : les méthodes exactes, et celles dites approchées. Les premières basées sur le paradigme de «séparation et évaluation» (Branch & X) garantissent l'optimalité des solutions produites. Bien que n'effectuant pas une exploration exhaustive de l'espace de recherche, elles ne sont cependant exploitables que sur des instances de taille raisonnable. La seconde famille de méthodes est essentiellement fondée sur les métaheuristiques, qui s'appliquent à des instances de taille nettement supérieure et/ou un nombre d'objectifs plus important. Elles ont pour objectif non pas de trouver la, ou les meilleure(s) solution(s), mais de générer rapidement une ou plusieurs solution(s) satisfaisante(s) compte-tenu du coût et des contraintes du problème.

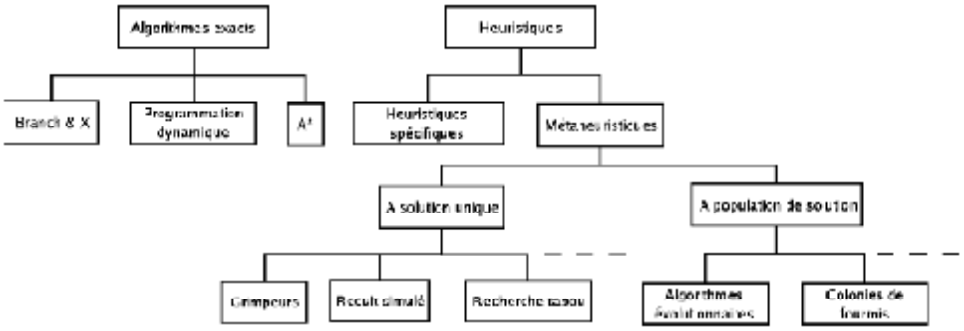


Figure 2.1. Une taxinomie des méthodes de résolutions appliquées à l'optimisation combinatoire

On distingue généralement deux grandes familles de métaheuristiques : celles à population de solutions, et les autres à base de solution unique. Les premières sont caractérisées par une tendance à l'exploration et à la diversification. Ces méthodes sont aptes à générer de « bonnes » solutions, uniformément réparties dans l'espace de recherche. Nous citerons, entre autres, les algorithmes évolutionnaires, les systèmes de fourmis, et la recherche par dispersion (scatter search).

D'autre part, les algorithmes à base de solution unique sont plus adaptés à intensifier la recherche dans une région prometteuse. Nous pouvons citer les algorithmes constructifs comme les recherches gloutonnes, la méthode de la descente, le recuit simulé, et la recherche tabou.

Par ailleurs, l'hybridation des deux classes de métaheuristiques, exploite de manière complémentaire les avantages de chacune d'elles, et permet d'améliorer grandement l'efficacité, s'appliquant à une exploration plus intense autour des bonnes solutions et plus diversifiée dans l'espace global.

2.3 Mémoire à programmation adaptative

La programmation à mémoire adaptative (PMA) est née de l'observation que les métaheuristiques récentes tendaient à devenir proches. En d'autres termes, on observe que les meilleures méthodes de résolution de problèmes d'optimisation combinatoire présentent les mêmes caractéristiques [48]:

- Les solutions générées par la recherche (ou bien une structure de données spéciale agrégeant des particularités de ces solutions) sont mémorisées ;
- Une solution provisoire est construite en consultant la mémoire ;
- La solution provisoire est améliorée à l'aide d'une recherche locale gloutonne ou avec une autre métaheuristique de base ;
- La nouvelle solution est utilisée pour mettre à jour la mémoire, pour adapter cette dernière aux nouvelles connaissances que la solution apporte.

La notion de programmation à mémoire adaptative insiste sur trois concepts fondamentaux: la mémoire, l'intensification et la diversification. Dans la littérature des

algorithmes évolutionnaires, ces deux dernières notions sont souvent respectivement désignées par les termes exploitation et exploration, ayant un sens similaire.

Nous avons choisi ici d'utiliser les termes de la définition originale de la programmation à mémoire adaptative, employés plus généralement dans la littérature de la recherche avec tabous ou du recuit simulé.

La mémoire représente ici l'information récoltée par l'algorithme, sur laquelle il va s'appuyer pour effectuer sa recherche. La mémoire est présente sous de nombreuses formes possibles, de la plus simple (une population de points) à des structures plus complexes (les pistes de phéromone des algorithmes de colonies de fourmis). La mémoire est une modélisation des solutions, elle est un moyen de décrire la fonction objectif. Cette modélisation s'opère par une distribution sur l'espace de recherche, biaisée vers les meilleures solutions, qui évoluent vers les optima du problème. Cette mémoire peut être définie comme globale (par rapport au problème dans son ensemble), individuelle, ou inter-individuelle (d'une solution relativement à une autre).

L'intensification consiste en l'utilisation des informations disponibles pour améliorer la pertinence de celles-ci. Du point de vue des métaheuristiques, il s'agit généralement tout simplement de recherche locale. Les algorithmes de recherche locale sont maintenant souvent employés en association avec d'autres métaheuristiques plus complexes, donnant lieu à des algorithmes hybrides [49].

La diversification est la recherche de nouvelles informations, afin d'augmenter la connaissance sur le problème. Elle fait souvent appel à des méthodes stochastiques, et il est pour le moment difficile de dégager des idées générales, tant la diversité d'approches de cette composante des métaheuristiques est grande.

En pratique, les trois composantes de la PMA sont liées, et il est parfois difficile de les repérer distinctement dans les métaheuristiques proposées. De fait, les métaheuristiques tentent d'équilibrer la balance entre diversification et intensification, et bien souvent les améliorations d'une métaheuristique existante consistent à faire pencher la balance dans un sens ou dans l'autre.

Le concept de programmation à mémoire adaptative se veut une forme de généralisation du mode de fonctionnement des métaheuristiques les plus efficaces. Cependant, la PMA propose une approche généraliste, sans entrer dans les détails de l'implémentation, qui induisent souvent des à priori sur la façon d'aborder tel ou tel problème.

2.4 Bases communes entre auto-organisation et la programmation à mémoire adaptative

L'auto-organisation nous renseigne sur la structure à employer pour concevoir des métaheuristiques flexibles et capables de s'adapter à un problème donné. En effet, la grande qualité de tels systèmes est de construire des comportements complexes à partir de règles simples, en se fondant sur une architecture fortement distribuée.

La programmation à mémoire adaptative nous renseigne quant à elle sur les méthodes employées par des métaheuristiques efficaces.

Les points communs entre ces deux théories sont nombreux et, du point de vue de la conception des métaheuristiques, il existe une relation simple entre elles : La PMA décrit le but à atteindre, et la théorie de l'auto-organisation un moyen pour atteindre ce but. Ainsi, une métaheuristique efficace devrait, selon la programmation à mémoire adaptative, mettre en place des mécanismes de mémoire, d'intensification et de diversification, reste la question des moyens à utiliser pour mettre en place ces mécanismes. L'auto-organisation propose un modèle de réalisation : un algorithme à base de population définissant des interactions simples au niveau local, permettant l'émergence d'un comportement complexe au niveau global.

Cette section présente quelques métaheuristiques à base de population s'appuyant sur l'auto-organisation. Nous avons délibérément choisi de restreindre la liste des métaheuristiques présentées à des classes d'algorithmes parmi les plus connues, afin d'éviter une énumération peu pertinente et forcément incomplète.

2.5 Algorithmes évolutionnaires

Les algorithmes évolutionnaires (AE) sont des techniques de recherche inspirées par la théorie de l'évolution biologique des espèces. Ils sont apparus au siècle dernier dans le début des années soixante, suite à plusieurs travaux de John Holland sur les systèmes adaptatifs [27]. L'ouvrage de David Goldberg [24], a largement contribué à les populariser.

Les algorithmes génétiques (AG) constituent certainement l'exemple le plus connu de cette classe d'algorithmes. Un algorithme génétique procède en faisant évoluer une population d'individus correspondant au départ à des solutions admissibles vers des solutions convergeant vers l'optimum global. Le déroulement d'un algorithme génétique se décrit comme suit :

Un ensemble de N points dans un espace de recherche, choisis à priori au hasard, constituent la population initiale ; chaque individu x de la population possède une certaine performance, qui mesure son degré d'adaptation à l'objectif visé : dans le cas de la minimisation d'une fonction objectif f, x est d'autant plus performant que $f(x)$ est plus petit. Au cours des générations, la composition de la population est maintenue à une taille constante. L'objectif est d'améliorer globalement la performance des individus; on s'efforce d'obtenir un tel résultat en appliquant les deux principaux mécanismes suivant :

- La sélection, qui favorise la reproduction et la survie des individus les plus performants.
- La reproduction, qui permet le brassage, la recombinaison et les variations des caractères héréditaires des parents, pour former des descendants aux potentialités nouvelles.

En pratique, une représentation doit être choisie pour les individus d'une population. Classiquement, un individu pourra être une liste d'entiers pour des problèmes combinatoires, un vecteur de nombres réels pour des problèmes numériques dans des espaces continus, une chaîne de nombres binaires pour des problèmes booléens, ou pourra même, au besoin, combiner ces représentations dans des structures complexes.

Comme le montre l'algorithme 2.1, le passage d'une génération à la suivante se déroule en quatre phases: une phase de sélection, une phase de reproduction (ou de variation), une phase d'évaluation des performances et une phase de remplacement. La phase de sélection désigne les individus qui participent à la reproduction. Ils sont choisis, éventuellement à plusieurs reprises d'autant plus souvent qu'ils sont performants. Les individus sélectionnés sont ensuite disponibles pour la phase de reproduction. Celle-ci consiste à appliquer des opérateurs de variation sur des copies des individus sélectionnés, pour en engendrer de nouveaux ; les opérateurs utilisés sont le croisement (ou recombinaison), qui produit un ou deux descendants à partir de deux parents, et la mutation qui produit un nouvel individu à partir d'un seul.

Algorithme 2.1. Algorithme génétique de base

- Initialisation : Générer aléatoirement une population initiale $P(t = 0)$.
- Evaluation de la population : Calcul de la fitness $f(i)$ de chaque individu de la population courante $P(t)$.

Tant que le critère d'arrêt n'est pas vérifié **faire**

- Sélection des parents pour la reproduction.
- Reproduction des parents en appliquant les opérateurs génétiques $(Croisement, Mutation)$.
- Evaluation des performances des enfants.
- Remplacement de certains individus dans la population P(t) par ceux créés lors de la phase de reproduction pour obtenir la nouvelle population P(t+1).

FinTantque

La structure des opérateurs de variation dépend étroitement de la représentation choisie pour les individus. Les performances des nouveaux individus sont ensuite mesurées, durant la phase d'évaluation, à partir des objectifs fixés.

Enfin, la phase de remplacement consiste à choisir les membres de la nouvelle génération : on peut, par exemple, remplacer les individus les moins performants de la population par les meilleurs individus produits, en nombre égal. L'algorithme est interrompu après un certain nombre de générations, selon un critère d'arrêt à préciser.

Du point de vue de l'auto-organisation, dans cette famille de métaheuristiques, les rétroactions sont parfois difficiles à cerner, tant les variantes sont nombreuses. D'une façon générale, les rétroactions positives sont implémentées sous la forme d'opérateurs de type sélection, alors que les rétroactions négatives sont typiquement mises en place par des opérateurs de mutation.

Pour ce qui est de la programmation à mémoire adaptative, la mémoire est située au niveau local, l'évolution de chaque individu d'une itération à l'autre étant liée à l'évolution des autres individus.

L'intensification est mise en place par l'opérateur de croissement, ce qui permet d'accentuer la recherche autours des meilleures solutions trouvées.

La mutation quant à elle permet d'introduire de la diversification dans la recherche, et ainsi explorer les régions de l'espace de recherche non encore visitées.

2.6 Algorithmes de colonies de fourmis en optimisation

L'idée d'imiter le comportement des fourmis pour trouver de bonnes solutions à des problèmes d'optimisation combinatoire a été proposée par Colorni & al [13]. Le principe de cette métaheuristique est basé sur la manière dont les fourmis cherchent leur nourriture et retrouvent leur chemin pour retourner dans la fourmilière.

Initialement, les fourmis explorent les environs de leur nid de manière aléatoire. Sitôt qu'une source de nourriture est repérée par une fourmi, son intérêt est évalué (quantité, qualité) et la fourmi ramène un peu de nourriture au nid. Pour retrouver son chemin, elle prend soin de laisser derrière elle une phéromone, trace chimique qu'elle arrive à détecter. Durant le chemin du retour, elle dépose également une quantité de phéromone dépendant de l'intérêt de la source de nourriture. Comme toutes les fourmis font de même, les traces laissées augmentent plus rapidement pour les sources de nourritures proches de la fourmilière, et lorsque plusieurs traces mènent à la même source, les traces correspondant aux chemins les plus courts sont renforcées à un rythme plus élevé. Il en résulte qu'après un certain temps, les chemins les plus rapides menant à la source de nourriture sont marqués par des traces plus importantes (Figure 2.2).

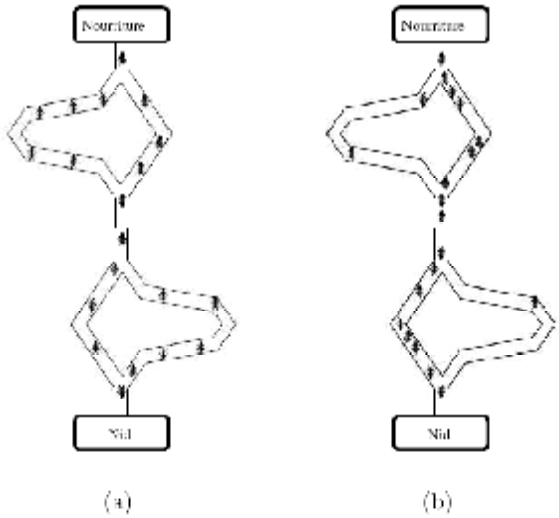


Figure 2.2. Expérience de sélection des branches les plus courtes par une colonie de fourmis : (a) au début de l'expérience, (b) à la fin de l'expérience.

Pour transposer ce comportement à un algorithme général d'optimisation combinatoire, on fait une analogie entre la zone dans laquelle les fourmis cherchent de la nourriture et l'ensemble des solutions admissibles du problème, entre la quantité ou la qualité de la nourriture et la fonction-objectif à optimiser, et enfin entre les traces de phéromones et une mémoire adaptative.

Le problème du voyageur de commerce (TSP : Travelling Salesman Problem) a fait l'objet de la première implémentation d'un algorithme de colonies de fourmis. Le passage de la métaphore à l'algorithme est ici relativement facile et le problème du voyageur de

commerce est bien connu et étudié. Ce problème consiste à trouver le trajet le plus court (tournée / 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.

Dans l'algorithme Ant System, 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 :

- 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
- l'inverse de la distance entre les villes : $n_{ij} = 1/d_{ij}$, appelée visibilité. Cette information statique est utilisée pour diriger le choix des fourmis vers des villes proches, et éviter les villes trop lointaines ;
- 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. La matrice de phéromones où chaque entrée correspond à l'intensité de chaque arête constitue la mémoire adaptative de l'algorithme.

La règle de déplacement d'une fourmi est la suivante :

$$p_{ij}^k(t) = \begin{cases} \dfrac{\left(\tau_{ij}(t)\right)^{\alpha}\left(n_{ij}\right)^{\beta}}{\sum_{l \in J_i^k}\left(\tau_{il}(t)\right)^{\alpha}\left(n_{ij}\right)^{\beta}} & si\ j \in J_i^k \\ 0 & sinon \end{cases} \tag{2.2}$$

Où α et β sont deux paramètres contrôlant l'importance relative de l'intensité de la piste $\tau_{ij}(t)$, et de la visibilité n_{ij}. Avec $\alpha = 0$, 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 sont prises en compte. Pour éviter une sélection trop rapide d'un trajet, un compromis entre ces deux paramètres jouant sur les comportements de diversification et d'intensification 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 :

$$\Delta\tau_{ij}^k(t) = \begin{cases} \dfrac{Q}{L^k(t)} & si\ (i,j) \in T^k(t) \\ 0 & sinon \end{cases} \tag{2.3}$$

Où $T^K(t)$ est le trajet effectué par la fourmi k à l'itération t, $L^k(t)$ 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 ainsi :

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \qquad (2.4)$$

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 $\tau_0 \geq 0$.

Le pseudo-code de l'algorithme ACS est présenté dans l'algorithme 2.2.

Algorithme 2.2. Algorithme de colonies de fourmis (Ant System)

Pour $t = 1, \ldots, t_{max}$ **faire**
  Pour chaque fourmi $k = 1, \ldots, m$ **faire**
    – Choisir une ville au hasard.
      Pour chaque ville non visitée i **faire**
      – Choisir une ville j, dans la liste J_i^k des villes restantes, selon la formule 2.2
      Fin Pour
    – Déposer une piste $\Delta\tau_{ij}^{k}(t)$ sur le trajet $T^k(t)$ conformément à l'équation 2.3
  Fin Pour
  – Evaporer les pistes selon la formule 2.4
Fin Pour

L'algorithme Ant Colony System (ACS) a été introduit plus tard pour améliorer les performances du premier algorithme [15, 16]. ACS est fondé sur des modifications du AS:

1. ACS introduit une règle de transition dépendant d'un paramètre q_0 $(0 \leq q0 \leq 1)$, qui définit une balance diversification/intensification. Une fourmi k sur une ville i choisira une ville j par la règle :

$$j = \begin{cases} argmax_{u \in J_i^k} \left[(\tau_{iu}(t)) (n_{iJ})^{\beta} \right] & si\ q \leq q_0 \\ J & si\ q > q_0 \end{cases} \qquad (2.5)$$

Où q est une variable aléatoire uniformément distribuée sur $[0,1]$ et $J \in J_i^k$ une ville sélectionnée aléatoirement selon la probabilité :

$$p_{ij}^{k}(t) = \frac{\left(\tau_{iJ}(t)\right)\left(n_{iJ}\right)^{\beta}}{\sum_{l \in J_i^k} (\tau_{il}(t))\,(n_{il})^{\beta}} \qquad (2.6)$$

En fonction du paramètre q_0, il y a donc deux comportements possibles : si $q > q_0$ le choix se fait de la même façon que pour l'algorithme AS, et le système tend à effectuer une diversification ; si $q \leq q0$, le système tend au contraire vers une intensification. En effet,

pour $q \leq q0$, l'algorithme exploite davantage l'information récoltée par le système, il ne peut pas choisir un trajet non exploré.

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 :

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \rho\,\tau_0 \tag{2.7}$$

Où τ_0 est la valeur initiale de la piste. A 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. A chaque itération, la mise à jour globale s'effectue comme ceci :

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \rho\,\Delta\tau_{ij}(t) \tag{2.8}$$

Où les arêtes (i,j) appartiennent au meilleur tour T^+ de longueur L^+ et $\Delta\tau_{ij}(t) = 1/L^+$. 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 plus proches villes 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 2.6, sinon c'est la plus proche des villes non visitées qui sera choisie.

Du point de vue de la PMA, la mémoire adaptative est mise en place par la matrice de phéromones (pistes de phéromones), cette mémoire s'adapte aux nouvelles connaissances acquises en cours de recherche.

Concernant l'auto-organisation, les rétroactions positives correspondent à l'attractivité des pistes de phéromones. Cette attractivité est maintenue sous contrôle par le mécanisme d'évaporation qui constitue la boucle de rétroaction négative.

2.7 Optimisation par essaim particulaire

L'optimisation par essaim particulaire (PSO : Particle Swarm Optimization) [32], est issue d'une analogie avec les comportements collectifs de déplacements d'animaux (la métaphore a de plus été largement agrémentée de socio-psychologie). En effet, chez certains groupes d'animaux, comme les bancs de poissons, ou les nuées d'oiseaux, on peut observer des dynamiques de déplacements relativement complexes, alors que les individus eux-mêmes n'ont accès qu'à des informations limitées, comme la position et la vitesse de leurs plus proches voisins. On peut par exemple observer qu'un banc de poissons est capable d'éviter un prédateur : d'abord en se divisant en deux groupes, puis en reformant le banc originel (voir Figure 2.3), tout en maintenant la cohésion du banc.

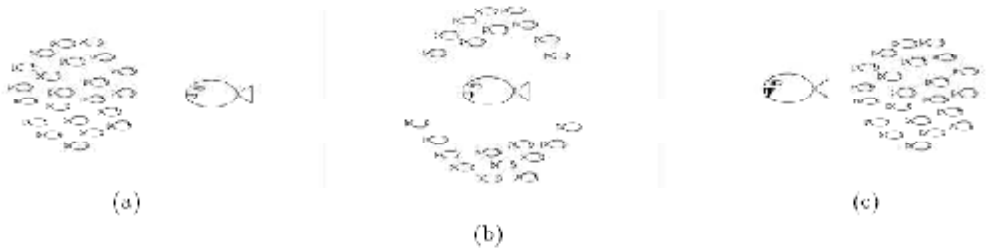


Figure 2.3. Schéma de l'évitement d'un prédateur par un banc de poissons. (a) le banc forme un seul groupe, (b) les individus évitent le prédateur en formant une structure en « fontaine », (c) le banc se reforme.

Ces comportements collectifs s'inscrivent tout à fait dans la théorie de l'auto-organisation. Pour résumer, chaque individu utilise l'information locale à laquelle il peut accéder sur le déplacement de ses plus proches voisins pour décider de son propre déplacement. Des règles très simples, comme « rester relativement proche des autres individus », « aller dans la même direction », à la « même vitesse » suffisent à maintenir la cohésion du groupe tout entier, et à permettre des comportements collectifs complexes et adaptés.

Les auteurs de la méthode d'optimisation par essaim particulaire se sont inspirés de ces comportements en mettant en perspective la théorie de la socio-psychologie sur le traitement de l'information et les prises de décisions dans des groupes sociaux. La méthode met en jeu de larges groupes de particules sous forme de vecteurs se déplaçant sur l'espace de recherche. Chaque particule i est caractérisée par sa position $\vec{x}_i$ et un vecteur de changement de position $\vec{v}_i$ (appelé vélocité). A chaque itération, la particule se déplace selon sa vitesse : $\vec{x}_i(t) = \vec{x}_i(t+1) + \vec{v}_i(t+1)$.

Le cœur de la méthode consiste à choisir comment définir $\vec{v}_i$. La socio-psychologie suggère que des individus se déplaçant (dans une carte socio-cognitive) sont influencés par leur comportement passé et par celui de leurs voisins (voisins dans le réseau social et non nécessairement dans l'espace). On tient donc compte, dans la mise à jour de la position des particules, de la direction de leur mouvement (leur vitesse), la meilleure position précédente $\vec{x}_{pbesti}$ et la meilleure position $\vec{x}_{gbest}$ parmi leurs voisins. Le changement de position s'effectue comme suit :

$$\begin{cases} \vec{v}_i(t) = \vec{v}_i(t-1) + \varphi_1\left(\vec{x}_{pbesti} - \vec{x}_i(t-1)\right) + \varphi_2\left(\vec{x}_{gbest} - \vec{x}_i(t-1)\right) & (2.9) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t-1) & (2.10) \end{cases}$$

Où les paramètres φ_1, φ_2 sont des variables aléatoires tirées dans $U_{[0,\varphi_{max}]}$ et qui ont pour rôle de pondérer les rôles relatifs de l'expérience individuelle (φ_1) et de la communication sociale (φ_2). C'est la notion de compromis psycho-social, mise en avant par les auteurs.

Le tirage aléatoire uniforme est justifié si l'on ne considère aucun à priori sur l'importance de l'une ou l'autre source d'informations. Pour éviter que le système n'explose en cas d'amplification trop grande d'oscillations, un paramètre V_{max} permet de limiter la vitesse (sur chaque dimension). Le pseudo-code pour la version la plus générale de l'algorithme est présenté dans l'algorithme 2.3.

Des améliorations peuvent être apportées à cet algorithme de base, notamment du point de vue du contrôle de la divergence. Dans la version de base, c'est le paramètre V_{max} qui va empêcher le système d''exploser par amplification des rétroactions positives.

L'utilisation de coefficients de *constriction* permet de mieux contrôler ce comportement [12]. Il s'agit ici d'ajouter, aux différents membres des équations de vélocité, des coefficients pouvant être manipulés pour contrôler la dynamique intensification/diversification du système. Dans la version la plus simple, un seul coefficient multiplie les deux membres de l'équation du calcul de la vélocité. Cette méthode de constriction permet de provoquer une convergence de l'algorithme (l'amplitude des mouvements des individus diminue jusqu'à s'annuler). L'algorithme réalise un compromis efficace entre intensification et diversification ; la seule restriction apparaît si les points $\vec{x}_{pbest_i}$ et $\vec{x}_{gbest}$ sont éloignés, auquel cas les particules continueront d'osciller entre ces deux points, sans converger. Une particularité intéressante est que, si un nouvel optimum est découvert alors que l'algorithme a convergé (donc, après une phase d'intensification), les particules iront explorer la région du nouveau point (phase de diversification).

Dans le même ordre d'idées, une version met en place un poids d'inertie (*Inertia Weight*), en multipliant chaque membre de l'équation de vélocité par un coefficient différent [46]. Pour résumer, le poids d'inertie décroît en fonction du temps, ce qui provoque une convergence contrôlable par ce paramètre. La dynamique générale reste la même que dans la version avec coefficient de constriction, à l'exception près de l'impossibilité de repartir dans une dynamique de diversification, si un nouveau meilleur point est trouvé.

Un autre paramètre important est la notion de voisinage, appliquée aux particules. Il semble être communément admis qu'un voisinage social (un individu x_2 ayant par exemple pour voisins les individus x_1 et x_3, quelles que soient les localisations spatiales de x_1, x_2, x_3) donne de meilleurs résultats qu'un voisinage spatial (fonction de la proximité des individus dans l'espace de recherche, par exemple). L'influence de la distribution initiale des particules a également été étudiée [47].

Ici, les rétroactions positives sont mises en place au niveau de l'attirance des particules les unes pour les autres. Les limitations de déplacement de chaque particule entre deux itérations forment les rétroactions négatives.

La mémoire est structurée au niveau local, entre particules voisines ; à chaque itération, chaque particule n'évolue qu'en fonction de ses proches voisins, et non pas selon l'état global de la population à l'itération précédente.

On trouvera une synthèse et une présentation détaillée de l'optimisation par essaim particulaire dans [11].

Algorithme 2.3. Algorithme de l'optimisation par essaim particulaire

N: nombre de particules.
$\vec{x}_i$: position de la particule P_i.
$\vec{v}_i$: vitesse de la particule P_i .
$pbest_i$: meilleure fitness obtenue pour la particule P_i.
$gbest$: meilleure fitness au sein du voisinage.
$\vec{x}_{pbest_i}$: position de la particule P_i pour la meilleure fitness.
$\vec{x}_{gbest}$: position de la particule ayant la meilleure fitness.

– Initialiser aléatoirement de l'essaim.
Répéter
Pour i de 1 à N faire
Si $(F(\vec{x}i\ ) > pbest_i)$ **alors**
$pbest_i \leftarrow F(\vec{x}_i)$
$\vec{x}_{pbesti} \leftarrow \vec{x}_i$
Fin Si
Si $(F(\vec{x}_i(t)) > gbest)$ **alors**
$gbest \leftarrow F(\vec{x}_i)$
$\vec{x}_{gbest} \leftarrow \vec{x}_i$
Fin Si
Fin Pour
Pour i de 1 à N **faire**
$\vec{v}_i(t) = \vec{v}_i(t-1) + \varphi_1\left(\vec{x}_{pbesti} - \vec{x}_i(t-1)\right) + \varphi_2\left(\vec{x}_{gbest} - \vec{x}_i(t-1)\right)$
$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t-1)$
Fin Pour
Jusqu'à ce que (le processus converge)

2.8 Systèmes immunitaires artificiels

Un système immunitaire artificiel (AIS : Artificial Immune System), s'applique à une vaste gamme de systèmes différents, notamment aux métaheuristiques d'optimisation inspirées du fonctionnement du système immunitaire des vertébrés. Un grand nombre de systèmes ont été conçus dans plusieurs domaines différents, tels que la robotique, la détection d'anomalies, ou l'optimisation [8, 9].

Le système immunitaire est responsable de la protection de l'organisme contre les agressions extérieures. La métaphore dont sont issus les algorithmes AIS met l'accent sur les aspects d'apprentissage et de mémoire du système immunitaire dit *adaptatif* (par opposition au système dit *inné*, notamment via la discrimination entre le *soi* et le *non-soi*. En effet, les cellules vivantes disposent sur leurs membranes de molécules spécifiques dites *antigènes*.

Chaque organisme dispose ainsi d'une identité unique, déterminée par l'ensemble des antigènes présents sur ses cellules. Les lymphocytes (un type de globule blanc) sont des cellules du système immunitaire qui possèdent des récepteurs capables de se lier spécifiquement à un antigène unique, permettant ainsi de reconnaître une cellule étrangère à l'organisme. Un lymphocyte ayant ainsi reconnu une cellule du non-soi va être stimulé à proliférer (en produisant des clones de lui-même) et à se différencier en cellule permettant de garder en mémoire l'antigène, ou en cellule permettant de combattre les agressions. Dans le premier cas, il sera capable de réagir plus rapidement à une nouvelle exposition à l'antigène : c'est le principe même de l'efficacité des vaccins. Dans le second cas, le combat contre les agressions est possible grâce à la production d'anticorps.

La Figure 2.4 résume ces principales étapes. Il faut également noter que la diversité des récepteurs dans l'ensemble de la population des lymphocytes est, quant à elle, produite par un mécanisme d'*hyper-mutation* des cellules clonées.

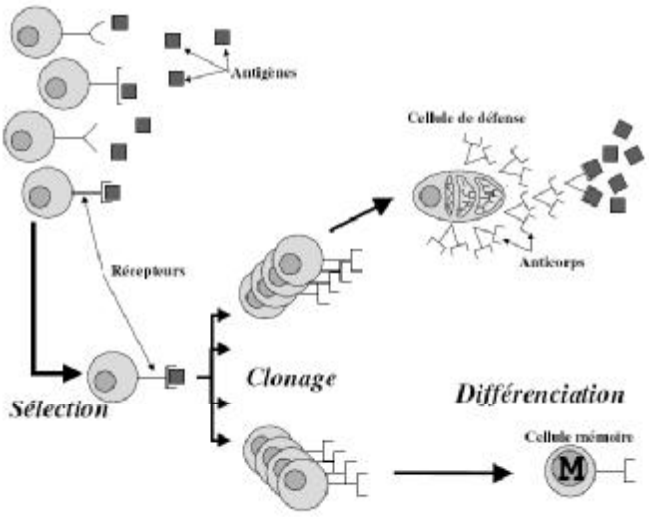


Figure 2.4. La sélection par clonage : des lymphocytes, présentant des récepteurs spécifiques d'un antigène, se différencient en cellule mémoire ou en cellule participant à la défense active de l'organisme par le biais d'anticorps.

Les principales idées utilisées pour la conception de la métaheuristique sont les sélections opérées sur les lymphocytes, accompagnées par les rétroactions positives permettant la multiplication et la mémoire du système. En effet, ces caractéristiques sont capitales pour maintenir les caractéristiques auto-organisées du système.

L'approche utilisée dans les algorithmes AIS est très proche de celle des algorithmes évolutionnaires, mais a également été comparée à celle des réseaux de neurones. On peut, dans le cadre de l'optimisation difficile, considérer les AIS comme une forme d'algorithme évolutionnaire présentant des opérateurs particuliers. Pour opérer la sélection, on se fonde par exemple sur une mesure d'affinité (i.e. entre le récepteur d'un lymphocyte et un antigène), la mutation s'opère quant à elle via un opérateur d'hyper-mutation directement issu de la métaphore. Au final, l'algorithme obtenu est très proche d'un algorithme génétique (voir algorithme 2.4).

Une description des fondements théoriques et de nombreuses applications des systèmes immunitaires artificiels peuvent être trouvée dans le livre de référence [14].

Algorithme 2.4. Un exemple d'algorithme de type système immunitaire artificiel

(1) Engendrer un ensemble de solutions P, composé d'un ensemble de cellules mémoires P_m ajoutées à la population présente P_r : $P = P_m + P_r$.
(2) Déterminer les n meilleures cellules P_n parmi la population P, en se fondant sur la mesure de l'affinité.
(3) Cloner les n individus pour former une population C. Le nombre de clones produits pour chaque cellule est fonction de l'affinité.
(4) Effectuer une hyper-mutation des clones, engendrer ainsi une population C^*. La mutation est proportionnelle à l'affinité.
(5) Sélectionner les individus de C^* pour former la population mémoire P_m.
(6) Remplacer les plus mauvais individus dans P pour former P_r.
(7) Si un critère d'arrêt n'est pas atteint, retourner en (1).

2.9 Conclusion

La programmation à mémoire adaptative présentée dans ce chapitre, permet de mieux comprendre le fonctionnement des métaheuristiques existantes et d'orienter la conception de nouvelles métaheuristiques. Les concepts importants à retenir sont l'utilisation par les métaheuristiques modernes de la mémoire, de l'intensification et de la diversification. L'aspect distribué et les qualités de flexibilité de ces algorithmes sont également des caractéristiques notables.

Cependant il faut souligner la difficulté de conception d'un système auto-organisé, ce qui explique que l'inspiration vienne de la biologie, où de tels systèmes sont relativement courants. Les difficultés principales sont les suivantes :

- concevoir une mémoire échantillonnant correctement le problème et dont il est aisé d'extraire l'information pertinente pour orienter la recherche,
- équilibrer la balance entre des techniques d'intensification et de diversification,
- maintenir la flexibilité de l'algorithme, de façon à ce qu'il s'adapte au problème, ou qu'il résolve des problèmes dynamiques,

Les perspectives ouvertes par la programmation à mémoire adaptative, et l'auto-organisation, permettront la conception et la mise en œuvre de nouvelles métaheuristiques.

Chapitre 3

Optimisation par auto-organisation chez l'espèce de fourmis Pachycondyla apicalis

3.1 Introduction

Les algorithmes inspirés par les modèles de comportements de colonie de fourmis connaissent un succès croissant parmi la communauté de chercheurs en informatique, et ceux de la recherche opérationnelle. On peut maintenant trouver des applications de ces algorithmes dans différents secteurs tels que la robotique, les réseaux de transmission et l'optimisation combinatoire.

Dans ce chapitre nous nous intéressons aux travaux menés autour de la modélisation des fourmis de l'espèce *Pachycondyla apicalis*. L'intérêt de ces fourmis pour l'optimisation vient du fait qu'elles utilisent des principes relativement simples à la fois d'un point de vue global et local pour chercher leurs proies. L'auto-organisation qui caractérise cette espèce de fourmis peut être reprise pour concevoir des métaheuristiques d'optimisation.

Nous proposerons dans ce chapitre une nouvelle métaheuristique nommée PAO pour Pachycondyla apicalis Ant Optimization, qui s'inspire des modèles sociaux d'interactions chez cette espèce de fourmis Nous pensons que l'auto-organisation et le concept de programmation à mémoire adaptative, peuvent conduire à concevoir des méthodes efficaces d'optimisation. Ces métaheuristiques ont la faculté de faire basculer la recherche entre intensification et diversification.

3.2 Éthologie et comportement de fourragement de l'espèce de fourmis Pachycondyla apicalis

Cette section donne un aperçu de la biologie des fourmis *Pachycondyla apicalis* et notamment de leur stratégie de recherche de nourriture (fourragement). Cette espèce est une ponérine néotropicale que l'on rencontre en Amérique du Sud, et en particulier au Mexique [19]. Sa morphologie et le peu de communication entre individus, la classe parmi les fourmis dites primitives mais certains aspects de son adaptation démentent ce jugement. Les fourmis *Pachycondyla apicalis* vivent en petites colonies comportant quelques dizaines d'individus (40 à 100 ouvrières). La Figure 3.1 présente les fourmis de cette l'espèce.

Le nid est installé dans de vieilles souches ou des arbres morts en décomposition qui offrent ainsi un habitat instable pour des fourmis qui ne savent pas construire de fourmilière. Quand la vétusté de leur nid devient trop importante, elles doivent déménager et chercher un nouveau refuge.

Leur nourriture se compose de petits insectes qu'elles capturent ou de cadavres d'insectes. Les ouvrières prospectent individuellement autour de la fourmilière et ramènent les proies au nid. Toutes ne participent pas à la recherche de nourriture, celles restant au nid s'occupent principalement du couvain.

Figure 3.1. Fourmis de l'espèce *Pachycondyla apicalis*

Le comportement de recherche de nourriture n'utilise pas un recrutement en masse. Par exemple, les ouvrières ne déposent pas de message chimique sur le sol pour indiquer à d'autres fourrageuses le chemin menant à une source de nourriture. On peut supposer que la nature des proies recherchées n'encourage pas spécialement ce genre de communication : la présence des proies étant relativement aléatoire, la capture d'une proie ne donne que peu d'informations sur la stabilité spatiale de la source de nourriture. Autrement dit, la probabilité de retrouver une proie dans la même zone qu'une précédente capture n'est pas suffisante pour y canaliser les forces de fourragement de la colonie. Cependant, d'un point de vue individuel, les ouvrières mémorisent leur site de capture et lors de leur prochaine sortie du nid, elles retournent systématiquement sur le dernier site de chasse fructueux. Cette spécialisation sectorielle est une réponse pour l'adaptation nécessaire à la découverte et l'exploitation de sources de nourriture.

Ce type de fourragement solitaire se retrouve particulièrement chez les espèces peu populeuses, les espèces à population importante utilisent des mécanismes de recrutement massif beaucoup plus couramment. Les fourrageuses solitaires développent en conséquence des mécanismes d'apprentissage plus évolués.

De sorties en sorties, les ouvrières s'éloignent du nid car la probabilité de trouver une proie est inversement proportionnelle à la densité des fourrageuses qui décroît évidemment quand la distance du nid augmente. Ainsi, les fourrageuses ont un comportement collectif indirect puisque de manière statistique elles coopèrent pour couvrir au mieux leur espace de recherche que constitue le voisinage du nid. Elles construisent de cette façon une mosaïque de zones de chasse qui couvre la périphérie du nid. La Figure 3.2 présente les aires de fourragement d'une colonie Pachycondyla apicalis.

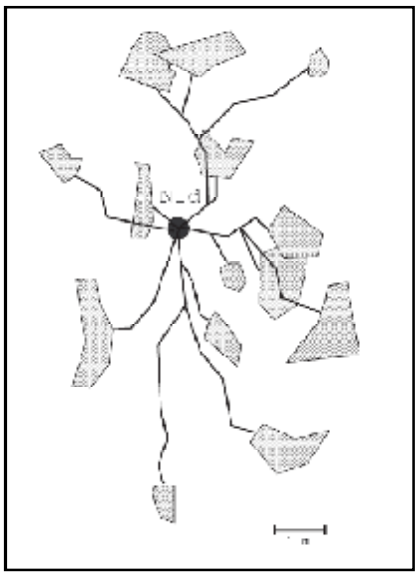


Figure 3.2. Exemple de carte des trajets et aires de récolte des fourrageuses

Le comportement de fourragement de *Pachycondyla apicalis* peut être résumé en trois règles [19, 41, 42]:

- La découverte d'une proie entraîne toujours le retour sur ce site lors de la sortie suivante, c'est là que la fourrageuse reprend ses nouvelles prospections.
- La découverte d'une proie pèse sur la décision de sortie des fourrageuses en réduisant l'intervalle de temps qu'elles passent au nid.
- Les fourrageuses semblent progressivement apprendre une association entre une direction de recherche opposée au nid et l'augmentation de la probabilité de succès à trouver des proies.

Ces règles ont l'avantage évident d'être très simples. On peut cependant préciser quelques points :

- Lors de ses premières sorties, la fourmi sort du nid dans une direction aléatoire mais s'éloigne peu du nid et retourne à l'abri de celui-ci à la moindre alerte. Son trajet est relativement courbé.
- Si la fourmi capture une proie, elle retourne directement au nid et mémorise visuellement le chemin qu'elle emprunte. Ce retour s'effectue en ligne droite.
- Après une capture, la fourmi utilise le chemin qu'elle a mémorisé pour retourner au site de capture.
- Le retour sur le site de chasse se soldant par un échec peut se produire plusieurs fois de suite.
- Un site de chasse ne produisant plus le renforcement que constitue la capture d'une proie est abandonné mais n'est pas obligatoirement oublié par la fourrageuse.

- L'exploration d'un site de capture privilégie les directions qui éloignent la fourmi du nid dans une limite de périmètre imposée par le coût énergétique prohibitif que représente un échec à une grande distance du nid.

Comme nous l'avons déjà mentionné, la fragilité du nid impose des déménagements réguliers. Des fourmis éclaireuses partent alors à la recherche d'un nouvel abri. Le déménagement s'effectue grâce à des mécanismes de *recrutement en tandem* où une fourmi se fait guider par une de ses congénères jusqu'au nouvel emplacement. Ce changement de nid a pour effet de réinitialiser la mémoire des fourrageuses. C'est à cette occasion que l'utilisation de repères visuels prend toute son importance : le marquage de chemins avec des phéromones perturberait l'adaptation des fourmis à la situation de leur nouveau nid, car les anciens chemins interféreraient avec les nouveaux. Avec une mémoire visuelle, les fourrageuses reconstruisent un réseau de sites de chasse autour du nouveau nid plus aisément, plus rapidement.

On peut donc parler d'apprentissage en ce qui concerne la recherche de proies : la fourmi apprend les sites qui lui rapportent de la nourriture et elle est capable de les oublier quand elle n'y trouve plus de proies, ou lorsque le nid change de localisation.

L'aspect individuel de la recherche est particulièrement adapté à la fréquence d'apparition des proies: si une proie tombe sur le sol et est capturée, la fourrageuse reviendra explorer le site toute seule. Si la proie est trop lourde, elle sera découpée puis transportée en plusieurs voyages par la même fourmi qui utilise de cette façon sa mémoire. Si le site de chasse se trouve être un gisement (par exemple des termites) la fourrageuse reviendra systématiquement jusqu'à épuisement du site.

L'intérêt de la stratégie de fourragement des fourmis *Pachycondyla apicalis* réside dans sa simplicité et la bonne couverture de l'espace de recherche qui en résulte. On a ici un phénomène d'*émergence* : à partir de règles de recherche simples et individuelles, qui ne tiennent pas compte du travail des autres fourmis, on obtient une exploration radiale de l'espace centrée sur le nid.

3.3 Modélisation algorithmique

Nous présentons dans cette section l'adaptation de la stratégie de fourragement des fourmis *Pachycondyla apicalis* proposée par Fresneau dans [19], pour la résolution de problèmes d'optimisation. La modélisation que nous proposons est inspirée toute fois de l'algorithme API proposé par Monmarché dans [42].

Notre modélisation met en avant de nouveaux comportements collectifs d'intelligence en essaim. Ces comportements trouvent leur source en ce modèle d'auto-organisation.

Nous nous appuyons également sur le concept de mémoire adaptative pour guider l'élaboration de notre métaheuristique. Le but est de faire basculer la stratégie de recherche de notre algorithme entre intensification et diversification, en se basant sur une mémoire hiérarchique.

L'auto-organisation est mise en place par des boucles de rétroaction qui mettent en place les comportements sociaux des fourmis artificielles.

3.3.1 Espace de recherche et fonction d'évaluation

Nous allons considérer une population de n fourmis fourrageuses $a_i, \ldots, a_n$ de l'espèce *Pachycondyla apicalis*. Ces agents sont positionnés dans l'espace de recherche noté S, et vont tenter de (minimiser/maximiser) une fonction d'évaluation f définie de S dans R. Chaque point $s \in S$ est une solution valide du problème, ce qui signifie que f est définie en tout point de S.

Notre métaheuristique PAO (Pachycondyla apicalis Ant Optimization) est générique en ce qui concerne l'espace de recherche S (discret, continu,…). La définition des deux opérateurs suivants permet de déterminer facilement l'exploration des fourmis :

- Un opérateur O_{global} d'exploration globale. Cet opérateur permet de générer un point s à partir de l'emplacement du nid s_N. Le voisinage du nid correspond à tout point de l'espace de recherche. L'exploration autour du nid est paramétrée par une amplitude notée $A_{globale}$. Cette amplitude fixe la portée de l'exploration autour de s_N.
- Un opérateur de voisinage O_{local}. Cet opérateur nous permet de générer un point s' dans le voisinage d'un point s. La taille du voisinage de s est paramétrée par une amplitude, notée A_{locale}. Cette amplitude fixe la portée de l'exploration autour de s.

3.3.2 Comportement local des fourmis

Chaque fourmi a_i est caractérisée par son site de chasse courant s (a_i). Ce site désigne le site de fourragement. Par ailleurs, chaque fourmi retient une liste de p sites de chasse notée $L(a_i)$. Elle garde également en mémoire, le meilleur site de chasse qu'elle a pu trouver le long de ses recherches. Ce site est noté $s_b(a_i)$.

Si un site est épuisé, i.e. il ne lui rapporte plus de proies, la fourmi l'abandonne et se constitue un nouveau site de chasse. Elle se déplace à l'aide d'un vecteur de déplacement noté $d(a_i)$. Elle s'oriente en fonction de l'emplacement du nid s_N, et l'emplacement de son meilleur site de chasse $s_b(a_i)$.

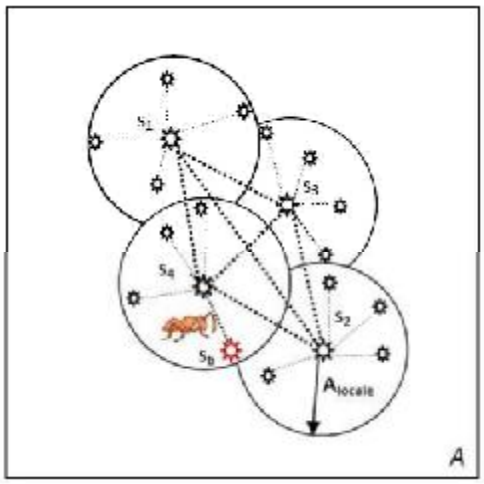


Figure 3.3. Exploration locale : la fourmi se constitue une liste de 4 sites de chasse s1, s2, s3, et s4. La fourmi explore le voisinage de chaque site dans une zone de rayon A_{locale}.

A l'initialisation, et à chaque déménagement du nid, chaque fourmi a_i quitte le nid pour se constituer une liste de p sites de chasse qu'elle mémorise.

Dans le contexte de l'optimisation, un site de chasse est un point de l'espace de recherche S, construit par l'opérateur O_{global} avec une amplitude $A_{globale}(a_i)$ dans le voisinage de nid correspondant au point s_N. La fourmi a_i procède par la suite à une exploration locale autour d'un de ses sites de chasse (Figure 3.3). Initialement, quand l'intérêt des sites est inconnu, la fourmi choisit un site s au hasard parmi les p dont elle dispose. L'exploration locale consiste à construire un point s' dans le voisinage de s grâce à l'opérateur O_{local} avec une amplitude $A_{locale}(a_i)$. La fourmi a_i capture une proie si cette exploration locale a permis de trouver une meilleure valeur de f, ce qui revient à avoir $f(s') < f(s)$. Ceci dans un cadre de minimisation de la fonction objectif.

Une amélioration de f modélise donc la capture d'une proie. A chaque fois qu'une fourmi parvient à améliorer $f(s)$, elle mémorise s' à la place de s et sa prochaine exploration locale aura lieu dans le voisinage de s'. Si l'exploration locale est infructueuse, pour la prochaine exploration, la fourmi choisira un site au hasard parmi les p sites qu'elle a en mémoire.

Quand un site a été exploré successivement plus de P_{locale} fois sans avoir rapporté de proie, il est définitivement oublié et sera remplacé par un nouveau site à la prochaine itération (c'est-à-dire la prochaine sortie du nid). Le paramètre P_{locale} représente une patience locale.

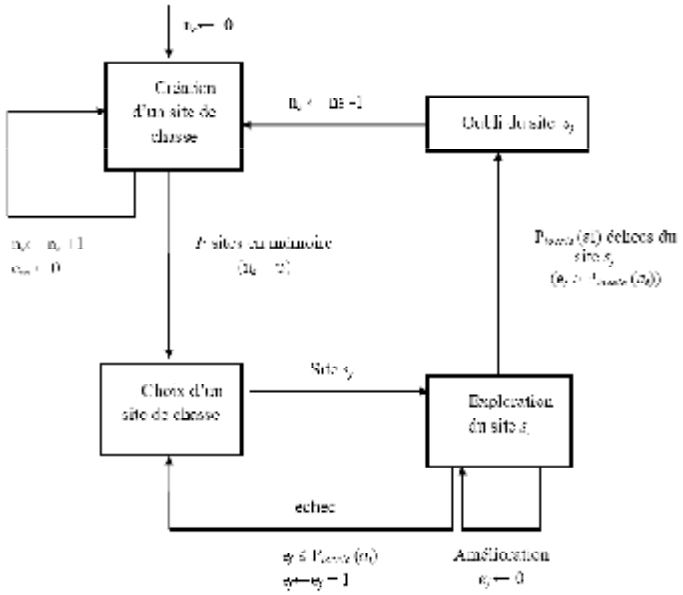


Figure 3.4. Automate représentant le comportement individuel de fourragement d'une fourmi

L'automate de la Figure 3.4 résume le comportement individuel d'une fourrageuse. n_s représente le nombre de sites que la fourmi mémorise à un instant donné. Le compteur e_j désigne le nombre d'explorations infructueuses successives sur un site s_j donné.

La création d'un nouveau site de chasse à la suite d'échecs successifs se traduit par un compromis psycho-social. Lorsque une fourmi a_i vient à abandonner un site, on tient en compte lors de la création d'un nouveau site pour chaque fourmi : ses déplacements antérieurs $d(a_i)$, son meilleur site de chasse $s_b(a_i)$, son site courant $s(a_i)$ qu'elle devra abandonner, et de l'emplacement s_N du nid. Ce dernier correspond au meilleur site de chasse trouvé au sein de la colonie.

La création d'un nouveau site $s'(a_i)$ après échec de l'exploration locale à l'itération t , se traduit comme suit :

$$d(a_i)_{t+1} \leftarrow d(a_i)_t + C_1\big(s_b(a_i) - s(a_i)\big) + C_2\big(s_N - s(a_i)\big) \quad (3.1)$$

$$s'(a_i) \leftarrow s(a_i) + d(a_i)_{t+1} \quad (3.2)$$

Où les paramètres c_1, c_2 sont des variables aléatoires tirées de manière uniforme dans l'intervalle [0,1], et qui ont pour rôle de pondérer les rôles relatifs à l'expérience individuelle (c_1) et de la communication sociale (c_2). C'est la notion de « compromis psycho-social ». La Figure 3.5 illustre le déplacement d'une fourmi d'un site ne lui rapportant plus de proies vers un nouveau site. Ce déplacement est régi par son vecteur de déplacement, son meilleur site de chasse, ainsi que l'emplacement du nid.

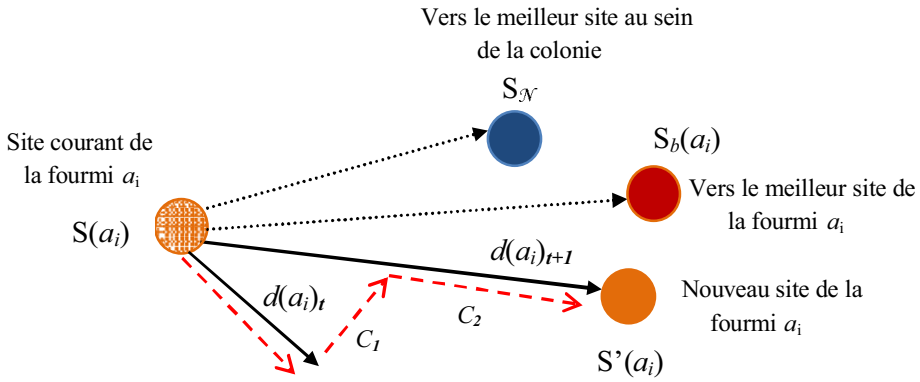


Figure 3.5. Déplacement d'une fourmi après échecs d'exploration sur un site

3.3.3 Exploration globale

D'un point de vue globale, la métaheuristique PAO place le nid à une position $\mathcal{S}$ dans S, et procède à l'exploration de l'espace de recherche à partir du nid. Les fourmis construisent leur liste de sites de chasse à partir de l'emplacement du nid dans un rayon $A_{globale}$ de celui-ci.

A l'initialisation, le nid est placé dans S de manière uniformément aléatoire. Le nid est déplacé toutes les P_N sorties du nid. Il est alors placé sur le meilleur point s^+ trouvé depuis son dernier déplacement. A chaque déménagement du nid, les fourmis reprennent leur exploration à partir de la nouvelle position du nid. La

Figure 3.6 illustre la stratégie d'exploration globale.

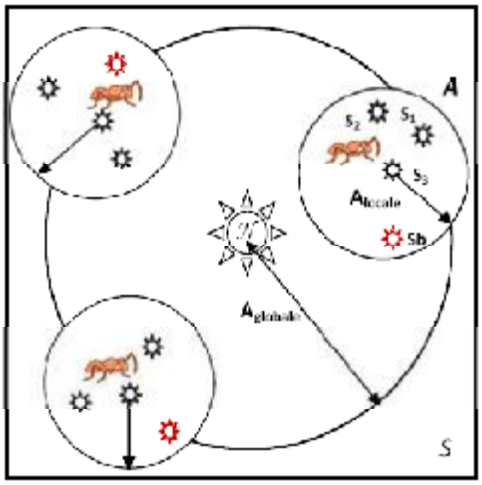


Figure 3.6. Exploration globale des fourmis

Le paramètre P_N représente un paramètre de patience pour le nid. On peut fixer cette patience suivant la patience locale d'une fourmi P_{locale}, et la taille p de la liste des sites de chasse mémorisés par une fourmi :

$$P_N = 2 \times P_{locale} \times p \quad (3.3)$$

Ce calcul a pour but de laisser suffisamment d'itérations entre chaque déplacement de nid pour que les fourmis puissent créer et explorer leurs p sites de chasse. Une autre solution serait de ne déplacer le nid qu'uniquement si l'optimum n'a pas été amélioré depuis un certain nombre d'itérations.

Enfin, à chaque déplacement du nid, la mémoire des fourmis est vidée et elles doivent reconstruire leurs p sites de chasse. Du point de vue de l'optimisation, ceci permet d'éviter d'être piégé par des minima locaux dans lesquels les fourmis resteraient enfermées. Ainsi, les fourmis se rassemblent autour du meilleur point trouvé et peuvent dès lors concentrer leurs recherches. On pourrait cependant procéder d'une manière plus douce : il suffirait de placer le nid à la position du minimum global trouvé par la colonie à chaque fois que celui-ci est amélioré sans réinitialiser toutes les fourmis. Ainsi, quand une fourmi crée un nouveau site de chasse, elle le ferait dans le voisinage de l'optimum global. On se débarrasse alors du choix de la patience du nid. Une autre méthode douce de déplacement du nid serait de le placer à une position intermédiaire entre sa position actuelle et s^+ (meilleure position trouvée lors de l'itération courante).

3.4 Algorithme de la métaheuristique PAO (Pachycondyla apicalis Ant Optimization)

Dans cette section nous présentons la métaheuristique PAO. L'algorithme de la métaheuristique est donné par l'algorithme 3.1. La condition d'arrêt peut être l'une des suivantes : un certain nombre de déménagements de nid a été effectué, l'emplacement du nid $s_{\mathcal{N}}$ n'a pas été amélioré depuis un certain nombre d'itérations, un certains nombre d'évaluation de solutions de la fonction objectif a été atteint.

Pour la complexité asymptotique de notre algorithme, si on suppose que le nombre de déménagements autorisés est de d, P_N est la patience du nid, et n est la taille de la colonie, on a au total $d \times P_N \times n$ itérations, ce qui donne un algorithme en $O(d \times P_N \times n)$.

Algorithme 3.1. Algorithme de la métaheuristique PAO (Pachycondyla apicalis Ant) Optimization)

(1) Choisir aléatoirement l'emplacement du nid.
(2) Les fourmis sont simulées en parallèle, pour chaque fourmi a_i, $i \in [1..n]$:
(3) Créer une liste de p sites de chasse à partir de l'emplacement du nid à l'aide de l'opérateur O_{global}.
(4) Assigner au meilleur site individuel, le meilleur site de la liste de sites chasse, seulement s'il y a *initialisation.*
(5) Se positionner sur un site choisi aléatoirement dans la liste des sites de chasse.
(6) Faire une exploration locale sur le site courant à l'aide de l'opérateur O_{local}.
(7) **Si** l'exploration précédente a rapporté des proies **alors**
   - Retourner à ce site à la prochaine sortie du nid.

Sinon

- Sélectionner un autre site aléatoirement parmi les p sites de chasse en mémoire pour la prochaine sortie.

Finsi

(8) **Si** un site a été exploré plus de P_{locale} fois sans rapporté de proie **alors**

- Créer un nouveau site de chasse en se déplaçant et en se positionnant selon les équations (3.1) et (3.2).
- Abandonner le site ne constituant plus de renforcement en l'effaçant de la liste de sites de chasse.

Finsi

(9) **Si** le site courant $S(a_i)$ rapporte plus de proies que le meilleur site individuel $S_b(a_i)$ **alors**

- Mettre à jour le meilleur site par le site courant.

Finsi

(10) **Si** le critère d'arrêt est vérifié **alors**

- Arrêt de l'algorithme

Sinon

Si P_N sorties du nid de l'ensemble de la colonie ont eu lieu **alors**

- Déplacer le nid S_N sur le meilleur site trouvé par les fourmis durant leurs recherches.
- Effacer pour chaque fourmi la liste des sites de chasse.
- Aller à (3).

Sinon

- Aller à (6).

Finsi

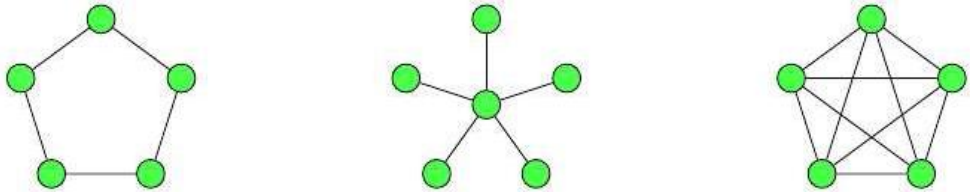
Finsi

3.5 Topologie du voisinage

La topologie du voisinage définit avec qui chacune des fourmis va pouvoir communiquer. On peut envisager de nombreuses combinaisons dont les suivantes sont les plus utilisées dans les essaims (Figure 3.7) :

- *Topologie en étoile* : chaque fourmi a connaissance de toutes les autres fourmis, ie. l'optimum du voisinage est l'optimum global.
- *Topologie en anneau* : chaque fourmi est reliée à n fourmis (en général, $n = 3$).
- *Topologie en rayon* : les fourmis ne communiquent qu'avec une seule fourmi centrale.

Le voisinage est plutôt social que géographique. Le voisinage géographique n'est pas nécessairement pertinent, car il s'agirait d'un voisinage trop local et très lourd en terme de calculs, car nécessitant de recalculer le voisinage à chaque itération.



(a) (b) (c)

Figure 3.7. Trois Topologies d'essaim différentes : (a) en anneau (n =3), (b) en rayon, (c) en étoile.

3.6 Variantes de la métaheuristique PAO

Dans cette section nous exposons des comportements sociaux coopératifs venant enrichir la métaheuristique PAO par de nouvelles explorations. Ils permettent de mieux contrôler la stratégie de recherche de notre métaheuristique. Ces comportements se résument en :

3.6.1 Recrutement en tandem

Nous avons repris dans cette métaheuristique le recrutement en tandem running (marche en tandem) décrit dans [42]. Ainsi, une fourmi qui parvient à améliorer un site, tente de recruter une autre fourmi en lui transmettant la position de son site de chasse. Si elle y parvient, le nombre de recherches locales pour ce site va augmenter.

Le paramètre P_{recrut} représente la probabilité pour qu'une fourmi tentant un recrutement y parvienne. Le recrutement est mis en œuvre en recrutant la fourmi ayant le plus mauvais site de chasse. Il est plus judicieux de régler le paramètreP_{recrut} de façon à avoir un recrutement progressif qui devient de plus en plus important avec l'âge du nid. Ceci évite de déplacer les fourmis prématurément dès les premières itérations de la recherche.

Le paramètre Nb_{recrut} quant à lui désigne le nombre maximum de fourmis que peut recruter une fourmi lors de cette phase. Par la suite nous noterons l'algorithme PAQ lorsque le recrutement est utilisé.

3.6.2 Exploration inter-sites

Nous avons introduit dans PAO l'exploration inter-sites. Ainsi, une fourmi peut intensifier sa recherche autour de deux de ses sites de chasse. Cette exploration a pour rôle d'introduire à l'itération prochaine deux nouveaux sites obtenus par la recombinaison (croisement) de deux sites. L'opérateur de sélection de sites O_{select} correspond à une sélection par roulette. Avec cette procédure, les meilleurs sites ont plus de chance d'être sélectionnés. L'opérateur de recombinaison de sitesO_{recomb} est un croisement en deux points [24].

Le paramètre P_{sites} représente la probabilité pour qu'une fourmi engageant une exploration inter-sites y parvienne. Nous noterons l'algorithme PAQlorsque l'exploration inter-sites est utilisée.

3.6.3 Exploration inter-fourmis

Deux fourmis peuvent combiner leurs efforts autour de leurs meilleurs sites de chasse. Cette exploration permet d'introduire à la prochaine itération de nouvelles solutions (sites) issus de la recombinaison entre les meilleurs sites de chasse des deux fourmis.

Le paramètre P_{ants} représente la probabilité pour qu'une exploration inter-fourmis se réalise entre deux fourmis tentant une exploration. Un opérateur de sélection est appliqué en vue de sélectionner les meilleurs sites des fourmis à recombiner. Nous noterons par la suite l'algorithme PAO_a lorsque l'exploration inter-fourmis est utilisée.

3.7 Auto-organisation et mémoire adaptative dans PAO

L'auto-organisation permet une structuration de la colonie offrant une exploration radiale de l'espace de recherche centrée sur le nid. Elle est mise en place par des boucles de rétroactions. Les boucles de rétroactions positives correspondent à l'attachement des fourmis aux sites de chasse qui leurs rapportent des proies. Les déménagements périodiques du nid permettent de réinitialiser la recherche en direction de nouveaux sites de chasse, tout en abandonnant les sites non prometteurs. Cette réinitialisation constitue les rétroactions négatives.

La gestion du flux d'information est mise en place par la topologie de voisinage de la colonie. Elle définit le mode de communication et d'interaction entre les fourmis composant la colonie.

Concernant la programmation à mémoire adaptative. La mémoire est structurée en niveaux hiérarchiques:

Mémoire globale : Cette mémoire désigne l'emplacement du nid. Elle est mise à jour à chaque déménagement. Elle prend la meilleure solution trouvée par la l'ensemble de la colonie.

Mémoire individuelle : Elle est structurée en deux niveaux :

- **mémoire de premier niveau :** elle se traduit par la liste des sites de chasse que se constitue la fourmi à l'initialisation, et qu'elle visite à chaque itération.
- **mémoire de second niveau :** elle correspond au meilleur site de chasse qu'a pu trouver la fourmi le long de ses recherches.

En s'appuyant sur cette mémoire, la stratégie de recherche de notre métaheuristique peut basculer entre diversification et intensification.

La diversification est apportée par les déménagements successifs, qui ont pour rôle de réinitialiser la recherche et favoriser l'exploration de nouvelles solutions.

L'intensification est mise en œuvre par les explorations locales et individuelles, qui permettent une exploitation des solutions pertinentes et prometteuses.

3.8 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle métaheuristique que nous avons appelé PAO pour Pachycondyla apicalis Ant Optimization [33, 34, 35] . Cette dernière s'inspire du comportement social et collectif de l'espèce de fourmis *Pachycondyla apicalis*.

Elle repose sur des bases solides, notamment, la théorie de l'auto-organisation et le concept de programmation à mémoire adaptative.

Son élaboration a été pensée en s'appuyant sur une mémoire adaptative hiérarchique. Cette mémoire permet de guider la recherche vers les meilleures solutions, tout en favorisant l'intensification et la diversification dans la recherche. Elle adopte des stratégies issues de comportements d'intelligence en essaim. Ces comportements constituent des mécanismes de rétroaction nécessaires à la mise en place d'auto-organisation.

On a un algorithme à base de population définissant des interactions simples au niveau local, permettant l'émergence d'une nouvelle fonctionnalité au niveau global. Cette fonctionnalité est la capacité à converger vers un optimum global, tout en évitant d'être piégé par des minimas locaux.

Le prochain chapitre sera consacré à la résolution d'un problème combinatoire bien connu dans la littérature, à savoir le problème du voyageur de commerce. Des expérimentations seront réalisées sur un ensemble de jeux de tests. Le réglage de paramètres sera également discuté.

Chapitre 4

Expérimentations et Résultats

4.1 Introduction

Après avoir décrit notre métaheuristique, dans ce chapitre nous allons exposer et discuter les résultats d'expérimentations menées sur différents jeux de tests. Nous avons choisi de traiter le problème du voyageur de commerce (TSP : Travelling Salesman Problem). Le réglage des paramètres et l'implémentation de notre approche seront également abordés. Par ailleurs, les résultats obtenus par notre métaheuristique PAO et ses différentes variantes, seront fournis et discutés. Ces résultats serviront à comparer sur le plan qualitatif notre métaheuristique avec d'autres algorithmes basés sur les fourmis artificielles.

4.2 La plate-forme Paradiseo (PARAllel and DIStributed Evolving Objects)

Notre approche a été implémentée par dessus la plate-forme Paradiseo (PARAllel and DIStributed Evolving Objects)[5][8]. Cette plate-forme logicielle est écrite en C++, et est dédiée à la réutilisation de la conception et l'implémentation de métaheuristiques, mono et multi-objectifs. Elle renferme une large gamme de métaheuristiques, en passant par les métaheuristiques à base de solution unique (recherche tabou, recuit simulé,...), aux métaheuristiques à base de populations (algorithmes génétiques, optimisation par essaim particulaire). Elle propose des schémas d'hybridation entre métaheuristiques, et des modèles parallèles d'exploitation (modèle insulaire, fermier-travailleurs,...). Par ailleurs, elle offre une interface simple permettant le déploiement, et la mise à l'échelle de ces modèles sur grilles de calcul et clusters.

Dans ce travail, nous proposons le paquetage logiciel Paradiseo-PAO (Paradiseo-Pachycondyla apicalis Ant Optimization). Ce dernier renferme notre métaheuristique PAO et ses différentes variantes PAO, PAO$_s$, PAO$_a$.

En annexe de ce mémoire, les diagrammes de conception en UML de notre Paradiseo-PAO.

4.3 Problème traité

Dans ces expérimentations, nous avons traité le problème du voyageur de commerce (TSP : Travelling Salesman Problem)[30]. Son énoncé est le suivant : étant donné n points (des villes) et les distances séparant chaque point, trouver un chemin de longueur totale

[8] Paradiseo Framework website : http://paradiseo.gforge.inria.fr

minimale qui passe exactement une fois par chaque point et retourne au point de départ. Le but est de trouver un cycle hamiltonien passant par toutes les villes.

Ce problème peut servir tel quel à l'optimisation de trajectoires de machines-outils par exemple, pour minimiser le temps total que met une fraiseuse à commande numérique pour percer n points dans une plaque de tôle. Plusieurs d'autres problèmes de routage se ramènent également au voyageur de commerce. Il est utilisé dans l'élaboration de tournées de service dans le transport routier des marchandises. Il est connu comme étant NP-complet, i.e. lorsque le nombre d'instances augmente, le nombre de solutions faisables augmente de manière exponentielle. Ainsi, il est impossible de trouver une solution optimale à ce problème en un temps raisonnable. Si n est le nombre de villes, le nombre de possibilités (nombre de trajets possibles) est de $(n-1)! / 2$. Il faut donc faire appel aux méthodes approchées pour résoudre ce problème.

Le choix de ce problème dans notre travail est justifié par le fait que la plus part des algorithmes basés sur les fourmis artificielles ont été portés sur ce problème. Ceci nous permettra par la suite de comparer qualitativement les solutions obtenues par notre métaheuristique avec celles d'autres métaheuristiques.

4.4 Adaptation de PAO pour le TSP

Toute métaheuristique doit être adaptée au problème auquel elle fait face. Cette adaptation passe par la représentation (i.e. codage), mais également par les opérateurs que la métaheuristique met en œuvre pour la recherche de nouvelles solutions. Il est important de souligner que notre métaheuristique peut être facilement adaptée à différents problèmes qu'ils soient discrets ou continus. Elle trouve son intérêt dans son indépendance vis-à-vis de l'espace de recherche.

Dans cette section nous présentons en détail l'adaptation de la métaheuristique PAO au problème du TSP.

4.4.1 Représentation et codage de la colonie

Généralement, la représentation d'une solution consiste à représenter sous forme de chaînes (binaire/entière /réelle) toute l'information nécessaire à la description d'un point dans l'espace de recherche. Cette représentation doit minimiser au mieux l'espace mémoire, ainsi que le nombre d'opérations effectuées sur les données relatives à la solution.

Nous exposons dans ce qui suit la représentation élaborée pour traiter le problème du TSP :

- La colonie est un ensemble de fourmis, chacune désignée par son identificateur.
- Une fourmi est représentée par une liste de sites de chasse. Ces sites sont générés à l'initialisation du nid dans l'espace de recherche.
- Un site de chasse représente une solution faisable. Pour le problème du TSP, il correspond à un chemin hamiltonien passant par toutes les villes.

Il est à noter que la représentation est fortement liée aux opérations auxquels elle est sujette. Dans ce cas, une opération de décodage est parfois nécessaire pour aboutir à une solution réalisable.

Nous avons opté pour une représentation en nombre entier. Cette représentation présente l'avantage d'être simple, mais également peu gourmande en temps de calcul, contrairement à des représentations en nombre réel envisageables. Ces dernières peuvent s'avérer inapplicables sur des benchmarks avec un grand nombre d'instances.

La représentation d'un site de chasse avec n villes à visiter se résume ainsi :

Villes :	1	2	3	...	...	...	n
Index :	i_1	i_2	i_3	…	…	…	i_n

Par exemple, pour 5 villes à visiter, un site de chasse correspondant à une solution possible serait :

Villes : 1	2	3	4	5
Index : 4	2	1	5	3

La translation de cette représentation en tournée effective se traduit ainsi :

Tournée : 3 – 2 – 5 – 1 – 4

Au final, en rajoutant le nœud de départ comme dernière destination du voyageur afin d'indiquer le retour de ce dernier à la ville de départ, on obtient la tournée complète suivante:

Tournée : 3 – 2 – 5 – 1 – 4 – 3

Chaque dimension du vecteur *Index* est bornée dans les limites de l'intervalle [1,n]. Par ailleurs, un décodage est nécessaire pour passer de la représentation utilisée à une tournée effective. Ce décodage intervient uniquement lors de la phase d'évaluation de la solution.

4.4.2 Opérateurs d'exploration

Exploration locale

L'exploration locale d'une fourmi a_i consiste à construire un point s' dans le voisinage du site de chasse courant $s(a_i)$, grâce à l'opérateur O_{local} selon une amplitude $A_{locale}(a_i)$.

Le voisinage N(s) d'une solution s, et l'ensemble des voisins que l'on peut obtenir à partir de s, en appliquant une transformation locale. Il est bien connu que les voisinages utilisant les échanges de type k-opt sont à la base des heuristiques les plus performantes pour le problème du voyageur de commerce.

De ce fait, l'opérateur local O_{local} se résume en l'application de l'heuristique de recherche locale k-opt sur le site de chasse courant $s(a_i)$. Cette heuristique consiste à échanger deux arcs non adjacents par deux nouveaux arcs, pour obtenir une nouvelle

solution [26]. Elle fournit la meilleure amélioration possible d'un tour donné en échangeant *k* couples d'arêtes à la fois. D'où son nom,*k*-optimal.

La Figure 4.1 montre le schéma de voisinage 2-opt, dans lequel on échange 2 arcs de la solution courante par 2 nouveaux arcs, et on inverse la chaine qui se trouve entre ces nouveaux arcs.

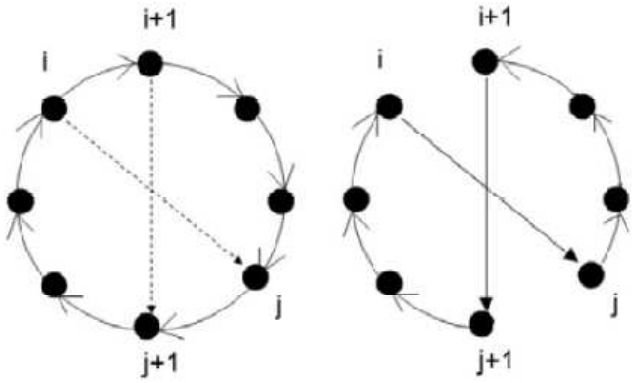
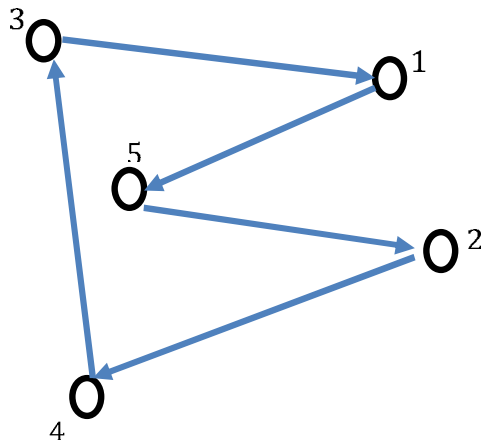


Figure 4.1. Voisinage 2-opt

Pour l'exploration locale, l'amplitude A_{locale} désigne le nombre de permutations *k*. La valeur de cette amplitude doit permettre une intensification de la recherche sur le site de chasse exploré. Par conséquent, il est nécessaire de minimiser le nombre de permutations à appliquer sur la solution initiale. Ainsi, on préconise une amplitude A_{locale} égale à 2 ou 3.

Comme exemple de l'application de l'opérateur O_{local} , soit un problème avec 5 villes. La tournée initiale est donnée comme suit :

Tournée : 3 – 1 – 5 – 2 – 4 – 3



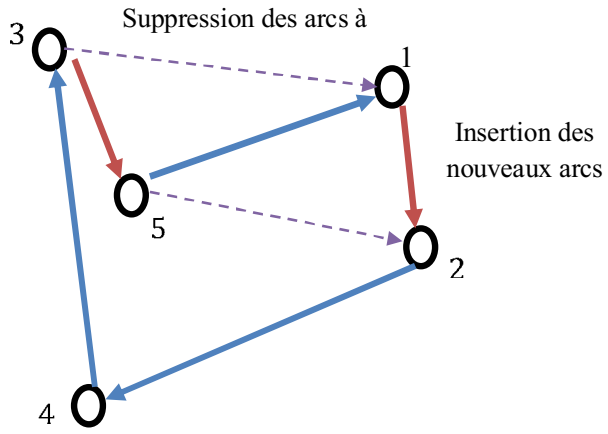
En appliquant l'opérateur O_{local} avec une amplitude A_{locale} égale à 2 sur la tournée précédente, il est envisageable d'obtenir la nouvelle tournée suivante :

$$3 - \mathbf{1 - 5 - 2 - 4} - 3 \xrightarrow{O_{local}} 3 - \mathbf{5 - 1 - 2 - 4} - 3$$

Ainsi, la tournée obtenue après application de l'opérateur O_{local} est :

Nouvelle tournée : 3 – **5** – **1** – **2** – **4** – 3

Schématiquement cette nouvelle tournée se traduit comme suit :



Le choix des arcs à échanger est aléatoire. Toute fois, il est nécessaire qu'ils soient non adjacents, sinon cela exclurait du tour le sommet commun aux deux arcs.

Déplacement d'une fourmi

Lorsqu'un site ne lui rapporte plus de proies, une fourmi se déplace à la recherche d'un nouveau site. Elle s'oriente à l'aide de son vecteur de déplacement. L'emplacement du nid, ainsi que l'emplacement de son meilleur site de chasse interviennent également dans son orientation.

D'une manière plus formelle, le déplacement d'une fourmi a_i est régi comme suit :

Soit $d(a_i)$ le vecteur de déplacement de la fourmi a_i. Ce vecteur est de dimension n (nombre de villes à visiter). Chaque dimension prend une valeur dans l'intervalle $[-n , n]$.

Comme indiqué dans les équations (3.1) et (3.2), la mise à jour du vecteur de déplacement s'effectue à l'itération t comme suit :

$$d(a_i)_t \leftarrow d(a_i)_{t-1} + C_1\big(s_b(a_i) - s(a_i)\big) + C_2\big(s_N - s(a_i)\big)$$

$$s'(a_i) \leftarrow s(a_i) + d(a_i)_t$$

$S_b(ai)$ représente le meilleur site de chasse de la fourmi a_i. S_N correspond à l'emplacement du nid, en d'autres termes, au meilleur site de la colonie.

Pour illustrer la mise à jour du vecteur de déplacement : soit un problème avec 5 villes à visiter.

Une fourmi a_i ayant comme site de chasse courant $S(a_i)$, le site suivant :

Villes : 1	2	3	4	5
Index : 4	2	1	5	3

Ce site se traduit par la tournée suivante :

Tournée ($S(a_i)$): 3 – 2 – 5 – 1 – 4

Soit maintenant, le vecteur de déplacement de la fourmi a_i à l'itération t-1 :

$d(a_i)_{t-1}$: -3 – 2 –1 – 0 – -1

En supposant que le meilleur site de chasse $S_b(a_i)$ trouvé par la fourmi a_i est le site suivant :

Villes : 1	2	3	4	5
Index : 5	1	3	2	4

La tournée correspondante à cette représentation est :

Tournée ($S_b(a_i)$) : 2 – 4 –3 – 5 – 1

Enfin, le site de l'emplacement du nid S_N peut être à titre d'exemple :

Villes : 1	2	3	4	5
Index : 1	5	4	2	3

La traduction de cette représentation en tournée se traduit comme suit :

Tournée (S_N): 1 – 4 – 5 – 3 – 2

Le déplacement d'une fourmi suggère de trouver un nouveau site de chasse et d'abandonner le site de chasse non prolifique. A chaque itération, la fourmi s'oriente à l'aide de son vecteur de déplacement. Ce vecteur est calculé par l'application de l'équation (3.1) qui se résume ainsi :

$$d(a_i)_t \leftarrow d(a_i)_{t-1} + C_1\left(s_b(a_i) - s(a_i)\right) + C_2\left(s_N - s(a_i)\right)$$

En supposant que les coefficients de pondération C_1 et C_2 sont à 1(influence similaire), on obtient :

$C1 \times (S_b(a_i) - S(a_i)) = (S_b(a_i) - S(a_i)) =$ (2 – 4 –3 – 5 – 1) - (3 – 2 – 5 – 1 – 4)

$(S_b(a_i) - S(a_i)) =$ -1 – 2 – -2 – 4– -3

$C2 \times (S_N - S(a_i)) = (S_N - S(a_i)) = (1 - 4 - 5 - 3 - 2) - (3 - 2 - 5 - 1 - 4)$

$(S_N - S(a_i)) = -2 - 2 - 0 - 2 - -2$

Ainsi, le nouveau vecteur de déplacement à l'itération t se calcule facilement comme suit :

$$d(a_i)_t \leftarrow d(a_i)_{t-1} + C_1(s_b(a_i) - s(a_i)) + C_2(s_N - s(a_i))$$

$d(a_i)_t = (-3 - 2 - 1 - 0 - -1) + (-1 - 2 - -2 - 4 - -3) + (-2 - 2 - 0 - 2 - -2)$

$d(ai)_t = (-3 - 2 - 1 - 0 - -1) + (-3 - 4 - -2 - 6 - -5)$

$d(ai)_t = -6 - 6 - -1 - 6 - -6$

On normalise le vecteur de déplacement de sorte que chaque dimension prend une valeur dans les limites de l'intervalle [-5 , 5]. On obtient le vecteur normalisé suivant :

$$d(ai)_t = -5 - 5 - -1 - 5 - -5$$

Le nouveau site de chasse est obtenu en additionnant le vecteur de déplacement au site de chasse courant selon l'équation (3.2) comme suit :

$S'(a_i) = S(a_i) + d(a_i)_t$

$S'(a_i) = (3 - 2 - 5 - 1 - 4) + (-5 - 5 - -1 - 5 - -5)$

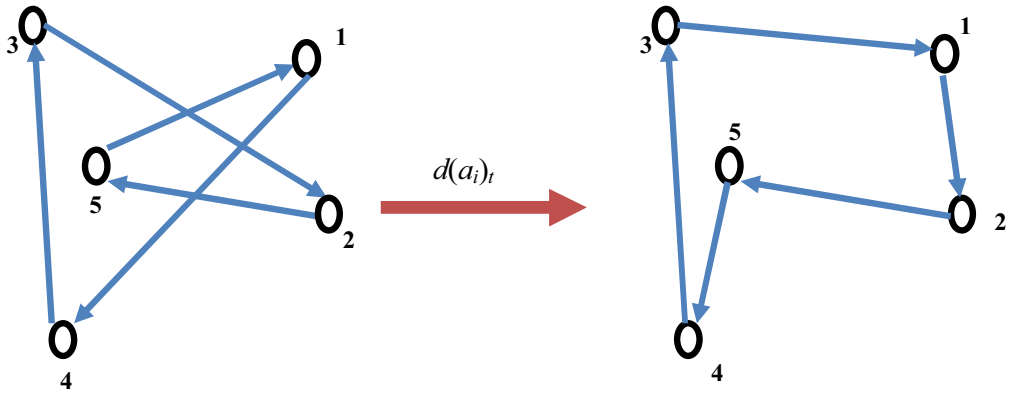
$S'(a_i) = -2 - 7 - 4 - 6 - -1$

On normalise le site de chasse obtenu de telle façon que chaque dimension prend une valeur dans l'intervalle [1, 5], on obtient le site de chasse normalisé suivant :

$S'(a_i) = 2 - 2 - 4 - 1 - 1$

$S'(a_i) = 2 - 5 - 4 - 3 - 1$

Pour résumer le déplacement de la fourmi, on illustre dans ce qui suit la création concluante du nouveau site de chasse $S'(a_i)$:



$S(a_i) = (3 - 2 - 5 - 1 - 4 - 3)$ $S'(a_i) = (2 - 5 - 4 - 3 - 1 - 2)$

Exploration globale

L'exploration globale consiste à explorer le voisinage de l'emplacement du nid S_N. Cette exploration permet à une fourmi de se constituer une liste de sites de chasse. Elle s'opère à l'aide de l'opérateur O_{global} selon une amplitude $A_{globale}$. Dans le cadre du problème du voyageur de commerce, nous avons introduit cette exploration par un opérateur de mutation génétique [3].

Ceci revient à appliquer une mutation sur le site du nid S_N. Une mutation adaptée à notre problème consiste à permuter deux villes choisies au hasard.

Souvent, la probabilité de mutation Pm par bit et par génération est fixée entre 0,001 et 0,01. On peut prendre également $Pm = 1/n$ où n est le nombre de villes composant un trajet. Il est possible d'associer une probabilité différente à chaque ville. Ces probabilités peuvent également être fixes ou évoluer dans le temps.

Dans notre contexte, l'amplitude globale indique le nombre de permutations à opérer. L'amplitude $A_{globale}$ est proportionnelle au nombre n de villes à visiter. Elle est calculée comme suit : $A_{globale} = n \times \delta$, tel que le facteur de pondération $\delta \in [0,1]$.

Par exemple, pour un problème dont le nombre d'instances est égal à 100, et pour δ égal à 0.1, le nombre de permutations à réaliser est de $0.1 \times 100 = 10$ permutations. Les villes à permuter sont choisies de manière aléatoire.

En appliquant l'opérateur de mutation O_{global} avec une amplitude $A_{globale}$ égale à 1, il est nécessaire d'effectuer n permutations sur le site du nid. Avec une telle amplitude, l'application de cet opérateur se résume en une recherche purement aléatoire.

Pour illustrer l'application de cet opérateur de mutation sur une ville du trajet : soit un problème avec 5 villes à visiter. Le site correspondant à l'emplacement du nid S_N est comme suit :

Villes :	1	2	3	4	5
Index :	5	1	3	2	4

Ce site se traduit par la tournée suivante :

Tournée (S_N) : $2 - 4 - 3 - 5 - 1$

L'application de l'opérateur O_{global} sur la deuxième ville, correspond en une permutation d'index avec une ville choisie au hasard, soit la ville choisie, la quatrième ville de l'ensemble des instances. De ce fait, le nouveau site de chasse $S(a_i)$ généré par la fourmi a_i à partir de l'emplacement du nid est :

Villes :	1	2	3	4	5
Index :	5	2	3	1	4

La translation de cette représentation en tournée est :

Tournée (S(a_i)) : 4 – 2 –3 – 5 – 1

La mutation joue le rôle de bruit et empêche l'évolution de se figer. Elle permet d'assurer une recherche aussi bien globale que locale, selon le poids et le nombre des villes mutés. Elle sert à éviter une convergence prématurée de l'algorithme en évitant d'être piégé par des minimas locaux. Ceci en introduisant constamment de nouveaux sites pour la colonie.

4.5 Variantes de PAO

4.5.1 Exploration inter-sites (PAOs)

L'exploration inter-sites consiste à faire un croisement entre deux sites de chasse d'une même fourmi. Ce croisement est réalisé par l'intermédiaire de l'opérateur O_{recomb}.

Cette exploration a pour rôle d'introduire à l'itération prochaine deux nouveaux sites obtenus par cette recombinaison. Pour déterminer quels sites sont plus enclins à obtenir les meilleurs résultats, une sélection est opérée. Ce processus est analogue à un processus de sélection naturelle, les individus les plus adaptés gagnent la compétition du croisement.

Une sélection par roulette est opérée [23]. Pour chaque site, la probabilité d'être sélectionné est proportionnelle à son adaptation au problème. Afin de sélectionner un site, on utilise le principe de la roue de la fortune biaisée. Cette roue est une roue de la fortune classique sur laquelle chaque site est représenté par une portion proportionnelle à son adaptation (fitness). On effectue ensuite un tirage au sort homogène sur cette roue. Meilleur est la fitness d'un site, plus grandes sont ses chances d'être sélectionné.

Pour illustrer l'application de l'opérateur de recombinaison de sites O_{recomb}, soit un problème avec 5 villes à visiter, et une fourmi a_i ayant parmi ses sites de chasse, les sites $S_1(a_i)$, et $S_2(a_i)$. Ces sites sont décrient comme suit :

Le site $S_1(a_i)$:

Villes : 1	2	3	4	5
Index : 4	2	1	5	3

Cette représentation se traduit ainsi :

Tournée S1(ai) : 3 – 2 – 5 – 1 – 4

Le site $S_2(a_i)$:

Villes : 1	2	3	4	5
Index : 5	1	3	2	4

La tournée correspondante à cette représentation est :

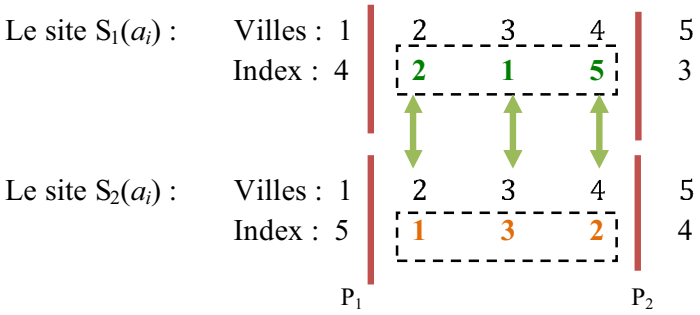
Tournée ($S_2(a_i)$) :2 – 4 –3 – 5 – 1

L'opérateur de recombinaison O_{recomb} correspond à un croisement PMX (Partial Mapped Xover) [2, 3]. C'est un croisement qui s'effectue en deux points. Le choix des points de croisement est strictement aléatoire. Généralement, ce type de croisement est considéré comme plus efficace qu'un croisement en un point.

L'idée est que les sites de chasse d'une même fourmi s'échangent des parties de leurs chaînes, pour donner de nouveaux sites. En d'autres termes, mutuellement, un segment de chaque site est transmis à l'autre. Ce croisement intervient selon la probabilité de recombinaison inter-sites P_{sites}.

Les sites enfants résultants de ce croisement doivent être normalisés pour qu'ils soient des solutions faisables. Il a pour conséquence d'intensifier la recherche localement entre deux des meilleurs sites d'une fourmi.

On illustre dans ce qui suit l'application de ce croisement sur les deux sites précédents $S_1(a_i)$, $S_2(a_i)$. Les points P_1 et P_2 désignent les points de croisement :



Après croisement et normalisation, on obtient deux sites enfants. Ces sites prendront la place des sites parents s'ils ont une meilleur fitness, i.e. les solutions qu'ils représentent ont un coût inférieur par rapport au coût des solutions parents.

Le site $S'_1(a_i)$: Villes : 1 2 3 4 5
Index : 4 1 3 2 5

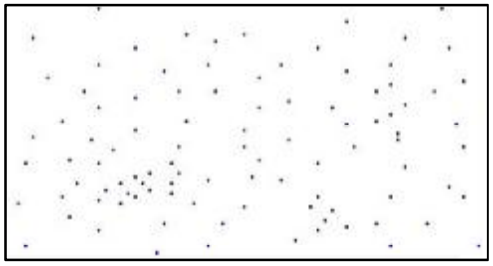
Le site $S'_2(a_i)$: Villes : 1 2 3 4 5
Index : 3 2 1 5 4

4.5.2 Exploration inter-fourmis (PAOa)

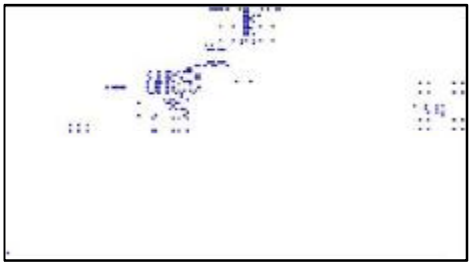
L'exploration inter-fourmis consiste à intensifier la recherche autour des meilleurs sites de chasse de deux fourmis. Elle est réalisée à l'aide de l'opérateur O_{recomb}. Cet opérateur est un croisement PMX. A la différence de l'exploration inter-sites, cette exploration fait intervenir des sites appartenant à différentes fourmis.

4.6 Les benchmarks utilisés

Nous avons repris certains jeux de tests de la librairie de benchmarks TSPLIB[9]. Cette librairie référencie les problèmes ainsi que les meilleures solutions relatives connues à ce jour. Un benchmarks comporte les cordonnées spatiales des villes à visiter. La distance entre les villes est simplement une distance Euclidienne. La Figure 4.2 illustre la distribution spatiale des instances (villes) de trois différents benchmarks sur un plan à deux dimensions. Les noms des benchmarks font référence à leur concepteur ainsi qu'au nombre d'instances qu'ils comportent. Par exemple *eil 51* est un benchmark de *Christofides & Eilon*, et dont le nombre d'instances est égal à 51 villes. Ces benchmarks différent dans la distribution spatiale des clients qui varie entre uniformité et regroupement (cluster). Il est à noter que les problèmes traités dans ce travail sont des problèmes dits symétriques, i.e. la distance séparant un nœud *i* d'un nœud *j*, est la même que celle séparant le nœud *j* du nœud *i*.

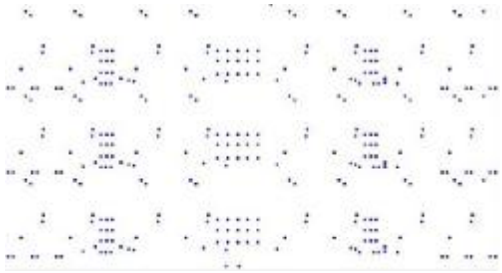


(eil 101)



(d198)

[9] Benchmarks du TSP: http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/



(lin318)

Figure 4.2. Visualisation des instances de différents benchmarks du TSP

4.7 Réglage des paramètres

Comme chaque métaheuristique est sujette à des paramètres, nous avons tenu à faire l'étude de sensibilité de notre métaheuristique selon certains paramètres.

Pour la métaheuristique PAO, nous avons pris comme paramètre l'amplitude globale $A_{globale}$. Pour les variantes de notre métaheutristique PAO, PAO_s, et PAO_a, les paramètres choisis pour cette étude sont :

- La probabilité de recrutement P_{recrut}.
- La probabilité de l'exploration inter-sites P_{sites}.
- La probabilité de l'exploration inter-fourmis P_{ants}.

Généralement, on procède en effectuant les expérimentations sur le premier paramètre. On le fixe après selon le meilleur résultat obtenu, et on réitère les expérimentations en fixant à la fin de ces expérimentations la valeur du deuxième paramètre selon le nouveau meilleur résultat, si jamais il y a amélioration. On poursuit de la même manière avec le reste des paramètres, jusqu'à ce que tous les paramètres soient considérés.

Les paramètres liés à la métaheuristique qu'on fixe sont relatifs au nombre de fourmis présentent dans la colonie, au nombre de sites que chaque fourmi exploite, à la patience locale, ainsi que la patience du nid.

Le nombre de fourmis constituant la colonie est égal à 20, la taille de la liste de sites de chasse de chaque fourmi est égale à 3. La patience locale P_{locale} est également fixée à 4. La patience du nid désignant le nombre d'itérations entre deux déménagement du nid se calcule simplement par $P_{nid} = 2 \times P_{locale} \times p = 2 \times 4 \times 3 = 24$.

L'amplitude locale A_{locale} est fixée à 2. Ceci implique l'application de l'heuristique 2-opt lors de l'exploitation du voisinage d'un site de chasse donné.

Par ailleurs, pour équilibrer entre exploration globale et locale, nous avons positionné le facteur de l'expérience individuelle (c_1), ainsi que celui de la communication sociale (c_2), à valeur égale à 1. Ceci permet d'avoir une influence similaire pour chaque type d'exploration.

La condition d'arrêt de notre algorithme est liée à un nombre de déménagements du nid autorisés. On fixe cette condition à 100 déménagements.

Enfin, la topologie choisie pour la colonie est une topologie en étoile, i.e. chaque fourmi est entièrement connectée au reste de la colonie. Par conséquent, toutes les fourmis de la colonie constituent son voisinage.

4.8 Résultats et analyse

Les résultats décrient dans les tableaux ci-dessous, correspondent à la distance parcourue par le voyageur de commerce pour le différents benchmarks. Ces résultats sont rapportés à la suite de 50 exécutions de notre algorithme pour chaque valeur de paramètre. La distance moyenne, et la plus petite distance parcourue sont indiquées. L'écart-type est également rapporté. Le meilleur résultat correspondant à la plus petite distance parcourue est mis en caractère gras. Il est à noter que pour le calcul d'un trajet, la distance s'exprime en distance Euclidienne. L'optimum de chaque problème est indiqué. Ce dernier est rapporté par la TSPLIB. Cet optimum désigne la meilleure solution connue à ce jour, en d'autres termes, la solution ayant le coût minimal.

Dans les colonnes, S_m représente la longueur moyenne du trajet parcouru par le voyageur sur la totalité des essais, et σ l'écart type correspondant. La meilleure solution trouvée S^* représente la longueur du plus court chemin trouvé.

Tableau 4.1. Influence de la variation de l'amplitude globale

Benchmarks	$A_{globale}$					Optimum
	$n \times 0.1$	$n \times 0.25$	$n \times 0.5$	$n \times 0.75$	$n \times 1$	
	S_m, S^*, σ	**S_m, S^*, σ**	**S_m, S^*, σ**	**S_m, S^*, σ**	**S_m, S^*, σ**	
eil51 (n=51)	437.72 **415** 9.07	439.12 421 9.73579	482.38 444 27.0281	537.7 464 32.6994	556.88 524 31.3717	**415**
eil76 (n=76)	569,9 550 10,754	571.96 **540** 13.69	574,3 564 11,776	575.7 545 13,622	573 562 8,756	**538**
kroA100 (n=100)	22470,33 22490 82,282	22660 22541 121,577	22570 **22380** 214	22719,5 22655 91,217	22848 22784 90,510	**21282**
eil101 (n=101)	691.38 669 11.535	667.74 **636** 13.9296	668.6 650 10.8757	673.36 645 12.9503	722.74 687 17.6293	**629**
d198 (n=198)	16826 16756 98,995	16410,5 16250 226,981	16502.6 **16229** 252.6	16518,66 16571 267,369	16801,5 16707 133,643	**15780**
lin318 (n=318)	48760 48421 479,418	47803 47287 729,734	4634,66 46277 577,507	4639.2 **45634** 447.833	7444 46975 663,266	**42029**
rat783 (n=783)	9852,5 9770 116,673	9895 9838 80,610	9787.4 **9667** 102.65	9878,66 9794 78,850	9825,66 9712 99,651	**8806**

Le Tableau 4.1 décrit les différents résultats obtenus sur les différents jeux de test en faisant varier l'amplitude globale $A_{globale}$. Elle est égale à $n \times \delta$, tel que n est le nombre de villes à visiter, et le facteur de pondération $\delta \in [0,1]$. Chaque fourmi génère sa liste de sites de chasse selon cette amplitude à partir de la solution que constitue l'emplacement du nid S_N.

Globalement, on remarque que pour les problèmes avec un nombre d'instances réduit, les meilleures solutions obtenues sont celles dont l'amplitude globale varie entre 10% - 25% du nombre d'instances. Ainsi, pour le problème eil51, l'optimum global est obtenu avec une amplitude globale égale à 10% du nombre d'instances.

Pour les problèmes combinatoires avec un nombre d'instances plus conséquent, les meilleures solutions obtenues sont celles ayant une amplitude globale variant entre 25%-75% du nombre de villes à visiter. Par exemple, pour le problème rat783 la meilleure solution est celle dont $A_{globale}$ est égale à $0.5 \times 783 = 391$.

Pour résumer, plus le nombre d'instances augmente, plus il est nécessaire d'introduire de la diversité dans les solutions en augmentant l'amplitude globale.

Tableau 4.2. Influence de la variation de la probabilité de recrutement

Benchmarks	P_{recrut}					Optimum
	0.1	0.25	0.5	0.75	1	
	S_m, S^*, σ	S_m, S^*, σ	S_m, S^*, σ	S_m, S^*, σ	S_m, S^*, σ	
eil51	436.72 418 8.74309	437.42 **417** 9.14569	434.26 419 7.03082	435,25 425 7,320	448,66 440 13,317	**415**
eil76	560 572,8 10,426	572.8 549 12.5252	572.64 **542** 12.8713	567,16 565 8,395	575.96 552 12.6459	**538**
kroA100	22702.6 21872 904.41	22611 22008 1037,506	22587.8 **21639** 910.563	23163 21872 1118,039	22668.6 22064 799.485	**21282**
eil101	667.74 641 12.3981	667.9 **633** 13.6693	664.26 639 10.8385	668.24 653 12.2956	675,33 664 13,317	**629**
d198	16689.8 1**6410** 58.2	16741 16809,8 82,669	16615.8 16500 163.2	16832 16760 101,823	16391 16624 248,109	**15780**
lin318	45863 **45427** 477,934	46755,66 46394 315,570	47174,33 46996 181,583	47263,5 47168 135,057	46666,25 46398 313,641	**42029**
rat783	9784 9677 116.024	9727.6 9655 59.1358	9757,66 **9620** 171,658	9844 9767 68,462	9805.33 9733 15.6667	**8806**

Le Tableau 4.2 rapporte l'influence de la variation de la probabilité de recrutement P_{recrut} sur la qualité des solutions obtenues par la variante de notre métaheuristique PAQ On peut facilement remarquer que les meilleures solutions obtenues sont celles avec une probabilité $P_{recrut} \leq 0.5$.

On peut expliquer cette situation par le fait que trop de recrutements peuvent amener à une convergence prématurée de notre algorithme vers des solutions loin de l'optimum. Cette convergence est due à la redondance de certaines solutions auxquelles l'algorithme converge dans le cas d'une probabilité de recrutement élevée.

Tableau 4.3. Influence de la variation de la probabilité d'exploration inter-sites

Benchmarks	P_{sites}					Optimum
	0.1	0.25	0.5	0.75	1	
	S_m, S^*, σ	**S_m, S^*, σ**	**S_m, S^*, σ**	**S_m, S^*, σ**	**S_m, S^*, σ**	
eil51	434.66 421 12,044	437,44 423 9,382	433.24 **420** 8.37272	433,7 421 7,394	437.24 **420** 8.826	**415**
eil76	572.06 549 11.9003	565 **553** 11,533	572.3 555 11.5694	569 562 7,550	567.7 542 10.9202	**538**
kroA100	23365 23252 159,806	22497,5 22283 303,349	22694.6 21748 630.901	22133.6 21742 460.211	22741 **21660** 589.17	**21282**
eil101	671,6 663 6,804	670 651 11,446	665 **640** 13.3611	667 651 15,100	667 645 6.028	**629**
d198	16606,5 16506 142,128	16811 16707 147,078	16656.6 16428 26.4	16603.4 16325 103.6	16454.4 **16209** 245.4	**42029**
lin318	48760 48421 479,418	46859,66 45964 1027,917	47141,66 46503 638,001	46679.3 **45337** 400.667	46431.2 45379 1052.17	**15780**
rat783	9840 9870 42,426	10059 9960 140,007	9714,5 9655 84,146	9791.8 **9620** 120.39	9797,5 9733 91,217	**8806**

Le Tableau 4.3 décrit la sensibilité de la variante PAOs de notre métaheuristique par rapport à la probabilité P_{sites} d'exploration inter-sites. On remarque que les meilleures solutions sont obtenues avec une probabilité $P_{sites} \geq 0.5$.

Ces résultats expriment bien le rôle de ce genre d'exploration, qui se résume en un croisement entre les meilleurs sites de chasse d'une même fourmi. Il permet d'apporter aussi bien de la diversification dans la recherche par l'introduction des sites enfants issues

du croisement des sites parents, mais également d'intensifier la recherche autour des meilleures solutions présentes localement au sein de la liste de chasse d'une même fourmi.

Tableau 4.4. Influence de la variation de la probabilité d'exploration inter-fourmis

Benchmarks	P_{ants}					Optimum
	0.1	0.25	0.5	0.75	1	
	S_m, S^*, σ	**S_m, S^*, σ**	**S_m, S^*, σ**	**S_m, S^*, σ**	**S_m, S^*, σ**	
eil51	441,4 423 16,652	435.12 417 8.68249	436.38 **421** 9.51187	435.86 423 8.60932	432,8 424 9,230	**415**
eil76	568,33 559 9,018	575 564 13.1733	573.86 556 11.52	571.3 547 10.9092	571.06 **546** 13.17	**538**
kroA100	22646.6 21671 642.798	22944.8 21875 634.003	22048.4 **21407** 867.781	23148 22596 572.749	22582.6 23176 440.547	**21282**
eil101	665.52 639 12.4406	666.2 **638** 13.2635	666.2 642 13.2635	666 673,5 10,472	667.2 646 12.9167	**629**
d198	16587 16301 146	16836.8 16681 103.8	16635.8 16491 76.2	16443.6 16334 129.4	16456.2 **16141** 324.8	**15780**
lin318	47570.2 46923 479.833	46827.3 45979 362.667	46679.3 **45337** 913.55	47023 46572 773,374	47570.2 46923 479.833	**42029**
rat783	9796,33 9703 94,522	9727.6 9655 59.1358	9805.33 9733 15.6667	9775 **9585** 136.56	9805.8 9712 67.7979	**8806**

Le Tableau 4.4 rapporte les solutions obtenues par la variante de notre métaheuristique PAO_a en faisant varier la probabilité de l'exploration inter-fourmis P_{ants}. On peut noter que pour l'ensemble des benchmarks, la meilleure solution obtenue est celle dont la probabilité $P_{ants} \geq 0.25$.

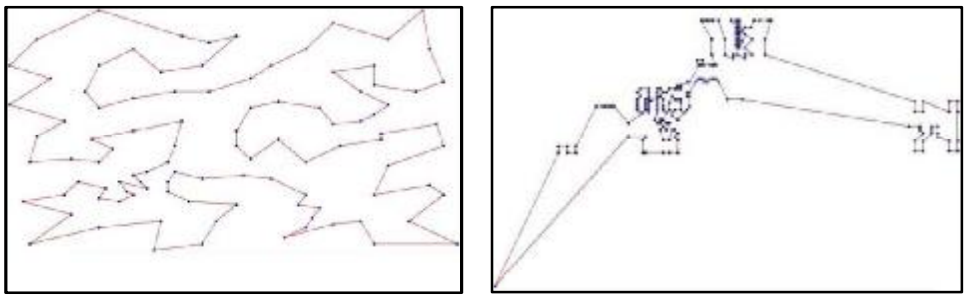
Ceci reflète l'intérêt majeur de cette exploration qui met en jeu un croisement entre les meilleurs sites de chasse de la colonie. Cette exploration permet aux fourmis de partager des connaissances sur le paysage de l'espace de recherche, et d'intensifier cette recherche en mettant en avant les sites les plus prometteurs.

Tableau 4.5 Meilleures solutions obtenues par PAO et ses variantes PAO $_r$, PAO$_s$, PAO$_a$

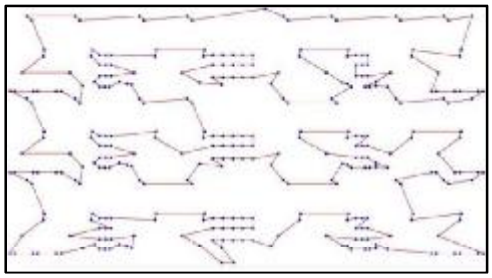
Benchmarks	*Métaheuristiques*				Optimum
	PAO	**PAO$_r$**	**PAO$_s$**	**PAO$_a$**	
	S_m, S^*, σ	**S_m, S^*, σ**	**S_m, S^*, σ**	**S_m, S^*, σ**	
eil51	437.72 **415** 9.07	434.26 420 7.03	433.24 421 8.37	435.86 423 8.60	**415**
eil76	571.96 **540** 13.69	572.64 542 12.87	572.3 548 11.56	571.06 546 13.17	**538**
kroA100	22570 22380 214	22668.6 22064 799.485	22694.6 21748 630.901	22048.4 **21407** 867.781	**21282**
eil101	667.74 636 13.9296	667.9 **633** 13.6693	665 640 13.3611	666.2 642 13.2635	**629**
d198	16502.6 16229 252.6	16615.8 16229 163.2	16454.4 16209 245.4	16456.2 **16141** 324.8	**15780**
lin318	46839.2 45634 447.833	46318.4 45427 791.17	46679.3 **45337** 913.55	47027.3 45535 689.667	**42029**
rat783	9787.4 9667 102.65	9727.6 9655 121.19	9791.8 9620 120.39	9775 **9585** 136.56	**8806**

Enfin, le Tableau 4.5 résume les meilleures solutions obtenues par la métaheuristique PAO ainsi que ses différentes variantes PAO$_r$, PAO$_s$, PAO$_a$. On note que pour les problèmes avec un nombre d'instances élevé, les meilleures solutions ont été obtenues par les variantes PAO$_s$, PAO$_a$. Ces extensions intègrent un mécanisme de croisement entre sites de chasse. L'intérêt de ce mécanisme se traduit par l'apport de nouvelles solutions en cours de recherche. Ceci par l'introduction des sites enfants issues du croisement. D'autre part, recombiner les meilleures solutions localement par une exploration inter-sites, ou globalement (i.e. sur l'ensemble de la colonie) par une exploration inter-fourmis, permet d'intensifier la recherche autours des meilleurs sites de chasse. Les meilleurs segments de trajets présentant un coût minimal se retrouvent hérités dans les prochaines générations.

La Figure 4.3 illustre les meilleurs trajets obtenus par la métaheuristique d'optimisation par fourmis *Pachycondyla apicalis*(PAO) sur quelques benchmarks traités.



(a) eil101 (b) d198



(c) lin 318

Figure 4.3. Visualisation des solutions faisables obtenues par PAO sur différents benchmarks

4.9 Comparaison des performances

Nous avons comparé PAO à l'algorithme API élaboré par Monmarché [42], ainsi qu'à la métaheuristique d'optimisation par colonie de fourmis ACS[15]. Cette comparaison est réalisée afin de mesurer les performances de notre métaheuristique par rapport à d'autres métaheuristiques basées sur les fourmis artificielles. Dans ce but, nous avons repris les résultats obtenus par API et ACS pour le problème du voyageur de commerce dans [41]. Dans un objectif d'équité, les paramètres de chaque algorithme ont été ajustés pour que le même nombre d'évaluation de la fonction objectif soit effectué. Le critère d'arrêt est fixé à 200 évaluations.

Les paramètres adoptés pour ACS sont les suivants:

- Le nombre de fourmis $n = 20$.
- La visibilité de la piste $\beta = 2$.
- Le coefficient de recherche $q0 = 0.98$.
- Le coefficient d'évaporation $\rho = 0.1$.
- La quantité initiale de phéromone $\tau 0$ est égale à 0.2.

Pour l'algorithme API, les paramètres fixés sont :

- La taille n de la colonie est de 20 fourmis.
- La patience du nid $P_N = 44$,
- La liste de chasse de chaque fourmi est composée de 2 sites.

Pour notre métaheuristique PAO nous avons repris les résultats obtenus précédemment dans le Tableau 4.5. Pour rappel, les paramètres utilisés sont :

- Le nombre de fourmis constituant la colonie est $n = 20$,
- L'amplitude locale $A_{locale} = 2$,
- La patience du nid $P_N = 24$,
- La patience locale $P_{locale} = 4$,
- La taille de la liste de chasse $p = 3$,
- Le coefficient de l'expérience individuelle $c_1 = 1$,
- Le coefficient de la communication sociale $c_2 = 1$.

Le Tableau 4.6 donne une comparaison sur la qualité des solutions trouvées dans PAO, API, et ACS. Dans les colonnes, S_m représente la longueur moyenne du trajet parcouru par le voyageur sur 50 essais, et σ l'écart type correspondant. La meilleure solution trouvée S^* représente la longueur du plus court chemin parcouru. La distance entre les villes s'exprime en distance Euclidienne.

Tableau 4.6.Comparaison des solutions obtenues entre différentes métaheuristiques

Benchmarks	*Métaheuristiques*			Optimum
	PAO	**API**	**ACS**	
	S_m, S^*, σ	**S_m, S^*, σ**	**S_m, S^*, σ**	
eil51	437.72 **415** 9.07	439.50 419 9.45	430.76 419 6.75	**415**
eil76	571.96 **540** 13.69	591.48 566 14.54	554.88 542 4.85	**538**
kroA100	22570 **22380** 214	59581.20 56879 1332.12	50995.44 26501 10098.60	**21282**
d198	16502.6 **16229** 252.6	21775.02 18769 1594.90	16977.78 16606 144.18	**15780**
lin318	46839.2 **45634** 447.833	75989.66 67902 3598.25	47043.08 46455 369.42	**42029**
rat783	9787.4 **9667** 102.65	18893.86 17081 846.62	10103.17 9880 71.80	**8806**

Les résultats obtenus par notre métaheuristique sont meilleurs en qualité que ceux obtenus par API et ACS sur les problèmes traités ci-dessus (Tableau 4.6). On note en moyenne une amélioration en qualité des solutions à hauteur de 50.17% pour PAO par rapport à API. Tandis que le gain moyen enregistré entre PAO et ACS s'élève à 4.34%. Bien que ACS utilise l'information de distance entre les villes comme heuristique locale, PAO a donné de bien meilleurs résultats, offrant un bon compromis entre exploration de l'espace de recherche et exploitation des solutions intéressantes.

4.10 Conclusion

Les résultats obtenus par notre métaheuristique PAO (Pachycondyla apicalis Ant Optimization) pour le problème du voyageur de commerce (TSP : Travelling Salesman Problem) sont très satisfaisants, et démontrent qu'en joignant la théorie de l'auto-organisation au concept de programmation à mémoire adaptative, on peut arriver à concevoir des métaheuristiques offrant une bonne balance entre intensification et diversification dans la recherche.

Notre métaheuristique ainsi que ses différentes variantes PAO$_p$, PAO$_s$, PAO$_a$, suggèrent une recherche s'inspirant des comportements auto-organisés de fourragement chez l'espèce de fourmis néotropicale *Pachycondyla apicalis*.

Cette métaheuristique est encadrée par le concept de mémoire adaptative qui consiste à offrir des moyens et des processus pouvant faire varier la stratégie de recherche de solutions, tout en se basant sur une mémoire adaptative. Cette dernière est représentée dans notre métaheuristique sur divers niveaux hiérarchiques (individuelle, globale).

L'auto-organisation, et le concept de programmation à mémoire adaptative, représentent les ingrédients indispensables à l'élaboration de métaheuristiques efficaces. Ils se présentent comme un nouveau paradigme sur lequel reposeront d'autres métaheuristiques et méthodes de résolution.

Conclusion générale et Perspectives

L'objectif de ce travail été de proposer de nouvelles méthodes de résolution de problèmes d'optimisation combinatoire, en s'appuyant sur des systèmes artificiels auto-organisés.

Dans ce but, une étude sur l'émergence et l'auto-organisation a été réalisée. Cette étude nous a permis de mieux cerner ces notions présentes dans différents domaines de recherche dont l'informatique.

D'un autre côté, nous avons essayé de recenser les métaheuristiques reposant sur la théorie d'auto-organisation, tout en décrivant leur fonctionnement selon le concept de programmation à mémoire adaptative. Ce concept présente les ingrédients de toute métaheuristique efficace. Ces ingrédients se résument en des processus de diversification, d'intensification, et sur une mémoire adaptative. Cette mémoire représente l'information récoltée par l'algorithme sur laquelle il va se baser pour mener sa recherche. Elle se caractérise par des mécanismes de mise à jour, lui permettant de s'adapter aux nouvelles connaissances acquises en cours de recherche. Ce concept nous a éclairé sur la façon dont fonctionnent les métaheuristiques les plus efficaces ; comme les algorithmes génétiques, les algorithmes de colonie de fourmis, ou les essaims particulaires.

La théorie de l'auto-organisation et la programmation à mémoire adaptative, nous ont servi comme base solide à l'élaboration d'une nouvelle métaheuristique.

Dans ce cadre, nous avons proposé la métaheuristique PAO (Pachycondyla apicalis Ant Optimization) ou l'optimisation par les fourmis *Pachycondyla apicalis*. Cette métaheuristique trouve son inspiration dans les comportements de fourragement des fourmis de l'espèce néotropicale Pachycondyla apicalis. Les fourmis de cette espèce présentent des comportements auto-organisés, où des interactions simples au niveau local permettent l'émergence de comportements ou de fonctionnalités globales complexes. Dans le contexte de l'optimisation combinatoire, cette fonctionnalité se résume en la faculté de converger vers l'optimum global, en évitant au mieux d'être piégé dans des minimas locaux.

En se reposant sur le concept de programmation à mémoire adaptative, notre métaheuristique intègre une mémoire hiérarchique permettant de guider la recherche de solutions. Cette recherche est opérée par les fourmis sur leurs sites de chasse. Ces sites constituent des solutions faisables. La mémoire de notre métaheuristique est présente à l'échelle locale (individuelle), aussi bien que globale (colonie).

Les comportements auto-organisés des fourmis artificielles offrent des mécanismes de diversification et d'intensification comme préconisé par le concept de programmation à mémoire adaptative. Par ces comportements, la stratégie de recherche de solutions s'adapte pour mieux répondre au problème traité.

Notre métaheuristique se présente en plusieurs déclinaisons. Ces dernières mettent en jeu d'autres comportements sociaux.

Elles se traduisent d'une part, par l'agrégation des efforts de recherche à l'aide de la variante de recrutement PAOr. Cette variante consiste à intégrer un comportement de recrutement dans le quel une fourmi recrute l'une de ses congénères afin d'apporter une intensification dans la recherche.

D'autres part, des comportements d'explorations peuvent impliquer aussi bien une seule fourmi que plusieurs. Dans le premier cas, la fourmi opère une recombinaison entre ses meilleurs sites de chasse. Cette recombinaison est introduite par la variante PAOs. Dans le second cas, la variante PAOa fait conjuguer les efforts de deux fourmis en effectuant une recombinaison entre leurs meilleurs sites de chasse.

L'implémentation de notre approche a été réalisée sur la plate-forme Paradiseo (PARAllel and DIStributed Evolving Objects). Cette plate-forme est dédiée à la conception et à l'implémentation de métaheuristiques. Dans ce cadre, nous avons proposé Paradiseo-PAO comme un nouveau paquetage logiciel regroupant la métaheuristique PAO, ainsi que ses différentes variantes.

Comme validation de notre approche, nous avons mené des expérimentations sur différents benchmarks du problème du voyageur de commerce (TSP : Traveling Salesman Problem). Le choix de ce problème est justifié par le fait que la plus part des métaheuristiques s'inspirant des colonies de fourmis ont été portées sur ce dernier. Par ailleurs, des comparaisons avec d'autres métaheuristiques ont été réalisées.

Notre métaheuristique PAO semble prometteuse. Elle repose sur des concepts et des théories solides, qui lui procurent une excellente aptitude à traiter les problèmes d'optimisation qu'ils soient discrets ou continus.

Comme perspective de notre travail, nous envisageons hybrider notre métaheuristique avec d'autres méthodes bio-inspirées [33]. Cette hybridation nous permettra de tirer profit des points forts de différentes méthodes de résolutions, et d'élaborer plusieurs schémas d'hybridation et de coopération entre ces méthodes.

Par ailleurs, nous comptons paralléliser (gridifier) notre métaheuristique. La gridification d'une métaheuristique consiste à la faire tourner sur grille de calcul, dans le but d'améliorer la qualité des solutions en réduisant les temps de calculs. Un schéma possible de parallélisation est celui des colonies coopératives (multi-colonies), où plusieurs colonies coopèrent, tout en évoluant en parallèle sur différents nœuds de calcul. Dans ce but, Paradiseo-PEO permet une exploitation des métaheuristiques sur grilles et clusters. Le développeur aura à définir les interactions entre les différentes colonies et les politiques de migration de solutions entre les différents nœuds de calcul.

Une autre perspective est celle qui traite des problèmes dynamiques. Ces problèmes sont connus comme étant des problèmes sujets à des changements perpétuels au cours du temps. Ces changements peuvent aussi bien affectés la fonction objectif que l'optimum global. Le but dans ce contexte est de traquer l'optimum global durant tout le temps de l'optimisation. Un algorithme traitant efficacement ce genre de problème doit être flexible et adaptatif pour pouvoir suivre l'évolution de l'optimum global. Par ailleurs, les informations récoltées par l'algorithme doivent être rafraîchies continuellement, d'où l'intérêt d'une mémoire adaptative pour l'algorithme de résolution.

Nous croyons que les caractéristiques des systèmes auto-organisés sont particulièrement intéressantes pour faire face aux problèmes dynamiques, en particulier leur grande force d'adaptation. Également, l'usage d'une mémoire adaptative permet de mieux cerner la dynamicité des problèmes traités. Ces deux aspects permettront la mise en place d'un nouveau paradigme sur lequel seront élaborées des métaheuristiques pouvant traiter efficacement des problèmes d'optimisation dynamiques.

Enfin, dans ce travail nous avons contribué à la mise en œuvre de nouvelles méthodes de résolution reposant sur la théorie de l'auto-organisation, et orchestré par le concept de programmation à mémoire adaptative. Les perspectives sont nombreuses, aussi bien dans la résolution des problèmes statiques, que dynamiques. Ce travail, ouvre certainement la voie à d'autres méthodes de résolution qui reposeront sur le paradigme proposé.

Annexe

Dans cet annexe nous détaillons la conception en UML (Unified Modeling Langage[10]) de notre métaheuristique d'optimisation par les fourmis Pachycondyla apicalis (PAO : Pachycondyla apicalis Ant Optimization). Cette dernière a été implémentée à l'aide de la plate-forme Paradiseo (PARAllel and DIStributed Evolving Objects[11]), pate-forme dédiée à la conception et à l'implémentation de métaheuristiques. Dans ce cadre nous proposons la paquetage logiciel Paradiseo-PAO. Ce paquetage regroupe notre métaheuristique PAO, ainsi que ses différentes variantes :

- PAO_r : PAO avec recrutement en tandem.
- PAO_s : PAO avec exploration inter-sites.
- PAO_a : PAO avec exploration inter-fourmis.

Dans ce qui suit, nous présentons la version adaptée au problème du voyageur de commerce (TSP : Travelling Salesman Problem). Les principales classes développées sont :

eoVectorSite<FitT,TownType> : Cette classe hérite de façon multiple et public à la fois du conteneur std ::vector <TownType>, ainsi que de la classe template EO<FitT>. Un site de chasse représente une solution faisable au problème traité. Chaque solution est caractérisée par sa fitness (coût). Cette classe hérite de l'attribut fitness défini dans la classe abstraite EO (Figure A. 1).

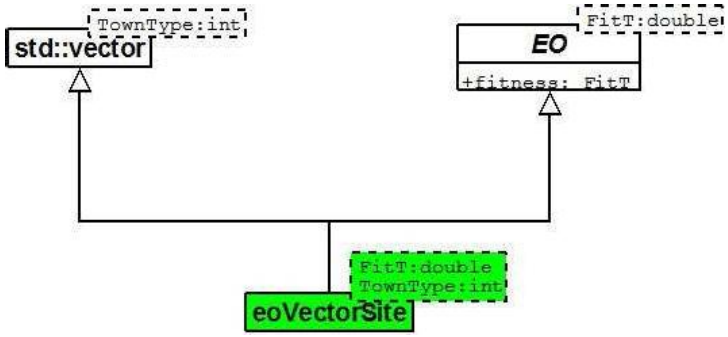


Figure A. 1. Schéma d'héritage d'un site de chasse représentant une solution au problème TSP

La classe eoPachyAnt<FitT, TownType, MovingType> désigne une fourmi artificielle. Chaque fourmi a connaissance d'un ensemble de sites de chasse. Par ailleurs, elle garde en mémoire le meilleur site de chasse qu'elle a pu trouver le long de ses recherches. Lorsqu'un site ne lui rapporte plus de proie, elle se constitue un nouveau site de chasse en s'orientant à l'aide de son vecteur de déplacement. Plus formellement, la classe eoPachyAnt encapsule les attributs suivants (Figure A.2):

[10] OMG UML Specification : http://uml.org

[11] Paradiseo Framework website : http://paradiseo.gforge.inria.fr

- *setHuntingSites* : Ensemble des sites de chasse de la fourmi.
- *bestSite* : Meilleur site de chasse de la fourmi.
- *MovingVector* : Vecteur de déplacement de la fourmi.

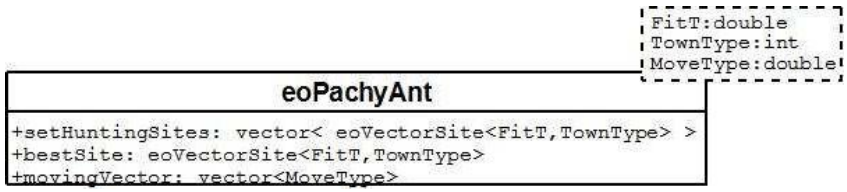


Figure A.2. Classe représentant une fourmi Pachycondyla apicalis

La classe eoColony<EOT> désigne toute la colonie des fourmis artificielles. Elle est représentée par l'ensemble des fourmis, ainsi que par l'emplacement du nid. Ce dernier représente la meilleure solution au sein de la colonie.

Cette classe dérive de la classe eoPop<EOT> définie dans la librairie Paradiseo. Par ailleurs, eoColony encapsule l'attribut*nestSite* . Cet attribut fait référence à l'emplacement du nid (Figure A.3).

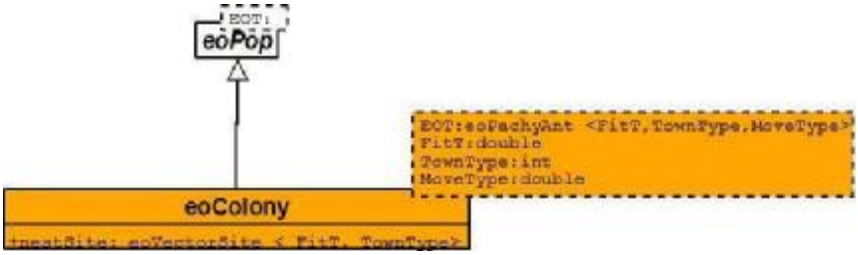


Figure A.3. Schéma d'héritage d'une colonie

La classe eoEasyPAO permet l'instanciation de notre algorithme de résolution. Elle correspond à une agrégation de plusieurs composants (Figure A.4):

- *eoInitializer* : Composant permettant d'initialiser la colonie. Ce qui revient à initialiser la liste de sites de chasse de chaque fourmi à partir de l'emplacement du nid.
- *eoEvalFuncPtr*: Opère l'évaluation des sites de chasse des fourmis. Le coût d'une solution (site) correspond à la distance parcourue par le voyageur.
- *eoRelocateContinue*: Condition d'arrêt de l'algorithme. Cette dernière correspond au nombre autorisé de déménagements du nid.
- *eoStandardMoving*: Gère le déplacement d'une fourmi dans le but de constituer un nouveau site de chasse dans l'espace de recherche.

- *eoStarTopology* : Composant définissant la topologie de la colonie. Le voisinage est défini comme un voisinage en étoile. Chaque fourmi est connectée à l'ensemble de ses congénères.
- *eoLocalExploration* : Composant gérant l'exploration locale de la colonie. Il se traduit par une recherche locale (2-opt). Cette recherche est opérée par chaque fourmi sur l'un des ses sites de chasse.
- *eoGlobalExploration* : Composant gérant l'exploration globale. Cette dernière se résume en une mutation sur le site correspondant à l'emplacement du nid.
- *eoUpdatingMemory* : Permet la mise à jour de la mémoire adaptative, à la fois localement (meilleur site de chasse de chaque fourmi), et également de manière globale (emplacement du nid).

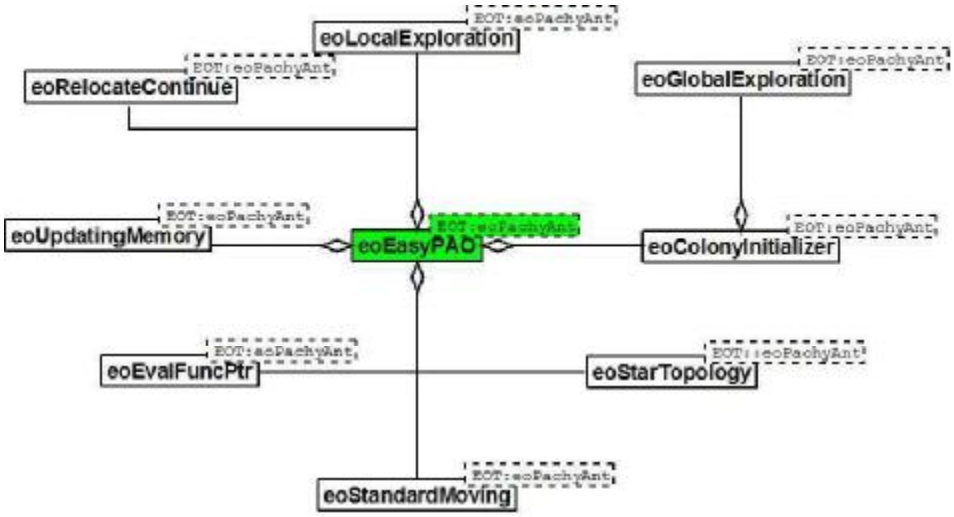


Figure A.4. Diagramme UML de l'algorithme PAO

Plus formellement dans la Figure A. 5, le composant eoLocalExploration agrège le composant TwoOpt défini dans la librairie Paradiseo. Il permet d'appliquer l'heuristique 2-opt sur une solution afin d'en explorer le voisinage.

Un autre composant d'exploration est eoGlobalExploration. Il applique une permutation (CitySwap) sur le site correspondant à l'emplacement du nid.

Les deux classes TwoOptMut et CitySwapMut dérivent de eoMonOp. Cette dernière est définie dans la librairie de classes de Paradiseo. Elle correspond à la super classe dont dérives toutes celles effectuant une mutation (perturbation) sur une solution unique.

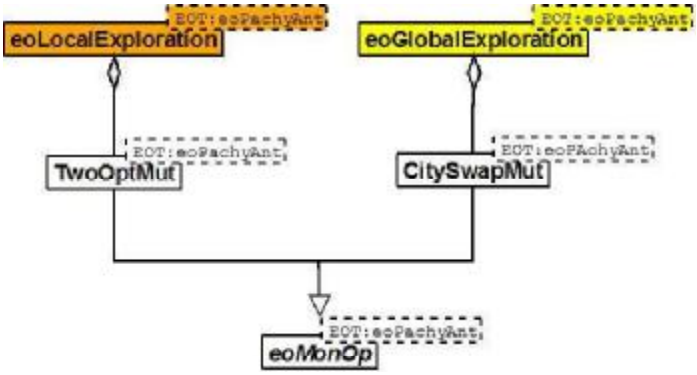


Figure A. 5. Diagramme UML des composants d'exploration locale et globale

Pour la déclinaison de recrutement de PAO, l'algorithme PAOr est instancié à partir de la classe eoPaoRecruitment. Il regroupe tous les composant précédents, vient s'agréger également le composant eoTandemRecruitment. Ce dernier permet d'opérer un recrutement tenté par une fourmi (Figure A.6)

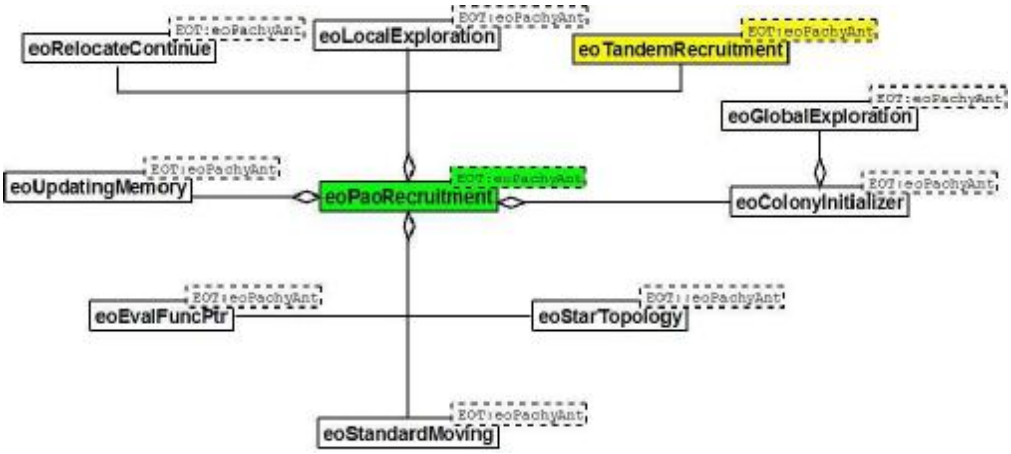


Figure A.6. Diagramme UML de la variante PAOr

L'exploration inter-sites correspondant à la varainte PAQ est instanciée de la classe eoPaoInterSites. Elle agrège tous les composants de eoEasyPAO, en plus du composant eoInterSitesExploration. Ce composant permet d'effectuer un croisement entre les meilleurs sites de chasse d'une même fourmi (Figure A.7).

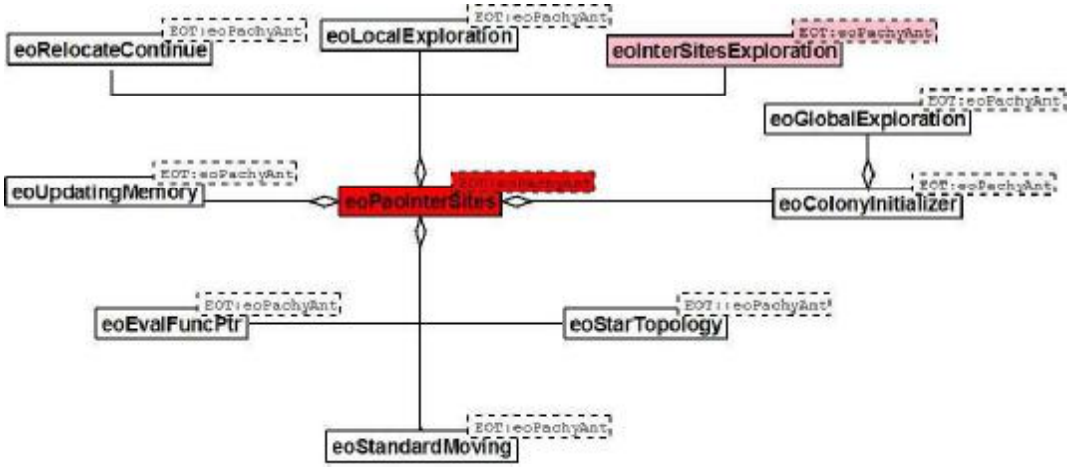


Figure A.7. Diagramme UML de la variante PAOs

Enfin, pour la variante PAO$_a$ correspondant à l'exploration inter-fourmis, l'algorithme est instancié à partir de la classe eoPaoInterAnts. Elle agrège comme toutes les variantes, les composants de eoEasyPAO, s'ajoute à cela le composant eoInterAntsExploration. Ce composant gère l'exploration sociale qui se traduit par une recombinaison entre les meilleurs sites de chasse de deux fourmis (Figure A.8).

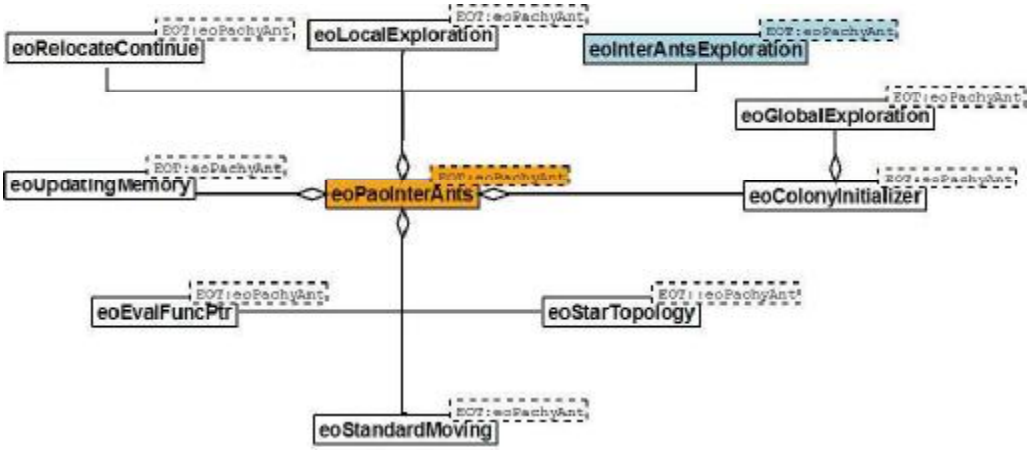


Figure A. 8. Diagramme UML de la variante PAOa

Bibliographie

[1] S. M. Ali and R.M. Zimmer. The question concerning emergence. In *First International Conference On Computing Anticipatory Systems, Casys'97*, 1997.

[2] D. Beasley, D.R. Bull, and R.R Martin. An overview of genetic algorithms : Part 1, fundamentals. *University Computing*, 15(2):58–59, 1993.

[3] D. Beasley, D.R. Bull, and R.R Martin. An overview of genetic algorithms : Part 2, research topics. *University Computing*, 15(4):170–181, 1993.

[4] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm intelligence, from natural to artificial systems. Technical report, Oxford University Press, 1999.

[5] S. Cahon, N. Melab, and E-G. Talbi. Paradieo: A framework for the reusable designe of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):1381–1231, 2004.

[6] S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. Self-organization in biological systems. Technical report, Princeton University Press, 2002.

[7] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artifcial Intelligence Research*, 9:317–365, 1998.

[8] L. De Castro, Castro, and F. Von Zuben. Artificial immune systems : Part i : Basic theory and applications. Technical report, TR-DCA 01/99, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil, 1999.

[9] L. De Castro, Castro, and F. Von Zuben. Artificial immune systems : Part ii - a survey of applications. Technical report, DCA-RT 02/00, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil., 2000.

[10] L. Chen, X-H. Xu, and Y-X. Chen. An adaptative ant colony clustring algorithm. In *Third International Conference On Machine Learning And Cybernetics*, 2004.

[11] M. Clerc. L'optimisation par essaim particulaire : principes, modèles et usages. *Technique et Science Informatiques*, 21:941–964, 2002.

[12] M. Clerc and J. Kennedy. The particle swarm : explosion, stability and convergence in a multi-dimensional complex space. *IEEE Transactions on Evolutionnary Computation*, 6:58–73, 2002.

[13] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of ECAL'91 - First European Conference on Artificial Life*, pages 134–142, 1992.

[14] D. Dasgupta. *Artificial Immune Systems and their applications* Springer Verlag, 1999.

[15] M. Dorigo and L. M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81, 1997.

[16] M. Dorigo and L.M. Gambardella. Ant colony system : A cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comp*, 1:53–66, 1997.

[17] M. Dorigo, V. Maniezzo, and A .Colorni. The ant system : optimization by a colony of cooperating agents.*IEEE Trans. Syst*, Man Cybern B(26):29–41, 1996.

[18] S. Forrest. Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. In*The Ninth Annual Clns Conference*, 1990.

[19] D. Fresneau. *Biologie et comportement social d'une fourmi ponérine néotropicale Pachycondyla apicalis*. PhD thesis, Universitéde Paris XIII, 1994.

[20] M. Gardner. The fantastic combinations of john conway's new solitaire game life. *Scientific American*, 223:120–123, 1970.

[21] J-P. Georgé.*Résolution de problèmes par émergence : etude d'un environnement de programmation émergente*. PhD thesis, Université Toulouse III-Paul Sabatier, 2004.

[22] M-P. Gleizes. *Vers la résolution de problèmes par émergence*. Hdr, HDR de l'Université Toulouse III-Paul Sabatier, 2004.

[23] D. Goldberg.*Genetic Algorithms*. Addison Wesley, 1989.

[24] D. E. Goldberg.*Genetic Algorithms in Search, Optimization and Machine learning* Addison-Wesley, 1989.

[25] S. Haykin. *Neural Networks, a Comprehensive Foundation, 2.nd. Edition*. Prentice-Hall, 1999.

[26] K. Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[27] J. H. Holland. Outline for logical theory of adaptive systems.*Journal of the ACM*, 9(3):297–314, 1962.

[28] J.H. Holland.*Emergence : from chaos to order*. Perseus Books, 1977.

[29] N. R. Jennings. Coordination techniques for distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley & Sons, 1996.

[30] D.S. Johnson and L.A. McGeoch.*The Traveling Salesman Problem : A case study in Local Optimization, Local Search in Combinatorial Optimization*. John Wiley and Sons, 1997.

[31] N. Jozefowiez. *Modélisation et résolution approchée de problèmes de tournées muti-objectif*. PhD thesis, Université des sciences et technologies de Lille (Lille-1), 2004.

[32] J. Kennedy and R.C. Eberhart. Particle swarm optimization.*Proc. IEEE Int. Conf. on Neural Networks*„ IV:1942–1948, 1995.

[33] M.R. Khouadjia, M. Batouche, and S. Chikhi. Hybridation co-évolutionnaire entre métaheuristiques d'optimisation: la colonie de fourmis pachycondyla apicalis et l'optimisation par essaim particulaire. In*MCSEAI'08: the 10th Maghrebian Conference on Information Technologies*, 2008.

[34] M.R. Khouadjia, S. Chikhi, and M. Batouche. Résolution de problèmes d'optimisation combinatoire par auto-organisation chez l'espèce de fourmis pachycondyla apicalis. In *CARI'08: Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées*, 2008.

[35] M.R. Khouadjia, S. Chikhi, and M. Batouche. Une nouvelle métaheuristique inspirée de l'auto-organisation chez la colonie de fourmis pachycondyla apicalis. In *COSI'08 : Colloque sur l'Optimisation et les Systèmes d'Information*, 2008.

[36] M.R. Khouadjia, S. Meshoul, and M. Batouche. An adaptative approach for the images segmentation: implementation on multi-agents framework. In *Euro XXII : 22Nd European Conference On Operational Research*, 2007.

[37] M.R. Khouadjia, S. Meshoul, and E-G. Talbi. A complex system for the images segmentation. In *Meta'06 : International Conference On Metaheuristics And Nature Inspired Computing*, 2006.

[38] C. Langton. Computation at the edge of the chaos, phase transition and emergent computation. In *The International Conference On Artificia Intelligence*, pages 12–37, 1990.

[39] G.H. Lewes. *Problems of life and mind*. Kegan Paul, trench, Turbner, &Co, 1875.

[40] J-P. Müller. Vers une méthologie de conception de systèmes multi-agents de résolution de problème par émergence. In *Actes des sixièmes Journées Francophones D'Intelligence Artificielle Distribuée et Systèmes Multi-Agents - Systèmes multi-agents de l'interaction à la socialité* -. Editions Hermès, 1998.

[41] N. Monmarché. *Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation*. PhD thesis, Université François Rabelais, Tours, 2000.

[42] N. Monmarché, G. Venturini, and M. Slimane. On how pachycondyla apicalis ants suggest a new search algorithm. *Future Generation Computer Systems*, 16:937–946, 2000.

[43] J-P. Muller and H. Van Dyke Parunak. Multi-agent systems and manufacturing. In *9th Symposium on Information Control in Manufacturing INCOM98*, 1998.

[44] J. Von Neumann. *The general and logical theory of automata* Collected Works, 1963.

[45] G. Nicolis and I. Prigogine. *Self-Organization in Non- equilibrium Systems* Wiley, New York, 1977.

[46] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *IEEE International Conference on Evolutionnary Computation*, 1998.

[47] Y. Shi and R. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionnary Computation*, pages 1945–1950, 1999.

[48] E.D. Taillard, L.M. Gambardella, M.Gendreau, and J-Y. Potvin. Adaptive memory programming : A unified view of meta-heuristics. *European Journal of Operational Research*, 135(1):1–16, 1998.

[49] E-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[50] F. Varela. L'auto-organisation: de l'apparence au mécanisme. In *Colloque De Cerisy: L'Auto-Organisation, De La Physique Au Politique*, 1983.

[51] E. Wilson and B. Hölldobler. Dense heterarchy and mass communication as the basis of organization in ant colonies. *Trend Ecol. Evol*, 3:65–68, 1988.

Solving combinatorial optimization problems by self-organized artificial systems

Abstract

In this report we present a new metaheuristic based on self-organization in *Pacchycondyla apicalis* ant species. This self-organization is summarized in simple and parallel interactions between the ants of colony.

Furthermore, it leans on adaptive memory programming for its design. This concept offers a unified view of the design of metaheuristics. It is based on a adaptive memory and different mechanisms as intensification and diversification. These mechanisms guide the research of solutions.

The proposed metaheuristic is implemented using ParadisEO which is a dedicated framework for the design and implementation of metaheuristics. Our metaheuristic is applied on Travelling Salesman Problem. The results are very encouraging, and bring to light self-organization theory as a novel paradigm for the design of metaheuristics.

Index-Terms: Bio-inspired metaheuristics, Ant colony algorithms, *Pachycondyla apicalis* ants, self-organization, Adaptive Memory Programming, ParadisEO framework.

حل مسائل رياضيات الإستمثال الإ ندماجية عبر أنظمة إصطناعية ذاتية التنظيم

موجز

في هذه المذكرة نقدم مساعد عام للكشف مستوحى من ذاتية التنظيم عند صنف النمل *Pachycondyla apicalis*.
هذه ذاتية التنظيم تتلخص في تفاعلات بسيطة و متوازية لجميع مستعمرة النمل. أيضا، تطويرها بني على مفهوم البرمجة ذات الذاكرة ذاتية التكيف.

هذا المفهوم يوفر إطار موحد لتصميم مساعد عام للكشف، إعتمادا على ذاكرة، وعلى آليات التكثيف والتنويع، لتوجيه البحث على الحلول.

المساعد العام للكشف المقترح تم تنفيذه على منصة الحاسوب ParadisEO ،هذه المنصة مكرسة لتصميم و تنفيذ أي مساعد عام للكشف عموما.

تم تطبيق المساعد العام للكشف على مشكلة الرحالة التاجر، النتائج التي تم الحصول عليها نتيجة للاختبارات مرضية جدا، وتسلط الضوء على نظرية التنظيم الذاتي بوصفها نموذجا جديدا لتصميم الطرق العامة للكشف.

الكلمات الرئيسية : مساعد عام للكشف مستوحى من البيولوجيا ، النمل *Pachycondya apicalis*، خوارزميات مستعمرات النمل، ذاتية التنظيم ، البرمجة ذات الذاكرة ذاتية التكيف ، منصة الحاسوب ParadisEO.