Université François Rabelais
Tours

École Doctorale : Santé, Sciences
et Technologies
Année Universitaire : 1999-2000

THÈSE POUR OBTENIR LE GRADE DE
DOCTEUR DE L'UNIVERSITÉ DE TOURS

Discipline : Informatique

présentée et soutenue publiquement

par :

Nicolas MONMARCHÉ

le 20 décembre 2000

Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation

---

Directeur de thèse : Professeur Gilles Venturini

---

jury :

Paul Bourgine	Examinateur	École Polytechnique, Paris
Jean-Louis Deneubourg	Examinateur	Université libre de Bruxelles
Jin-Kao Hao	Rapporteur	Université d'Angers
Philippe Preux	Rapporteur	Université du littoral, Calais
Marc Schoenauer	Examinateur	École Polytechnique, Palaiseau
Mohamed Slimane	Examinateur	E3i/Université de Tours
Gilles Venturini	Examinateur	E3i/Université de Tours

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction

Ce travail de thèse provient d'une intuition partagée par un petit nombre encore de chercheurs : les fourmis résolvant de nombreux problèmes de manière très efficace dans leur environnement, il doit être possible de transposer un certain nombre de ces mécanismes pour la résolution de problèmes informatiques.

L'informatique biomimétique

Lorsqu'un nouveau problème se pose en informatique, et plus généralement en ingéniérie, il faut parfois définir de nouvelles méthodes de résolution car les techniques existantes ne sont pas précisément adaptées au cas traité. Ainsi, lorsque l'on veut inventer une nouvelle méthode de résolution de problème, il faut souvent une source d'inspiration. Celle-ci peut être totalement imaginaire et n'est pas obligée de faire référence au monde réel comme par exemple des méthodes mathématiques abstraites ou peut au contraire être issue de la modélisation des systèmes complexes naturels. Il s'agit dans ce dernier cas de copier et d'adapter les concepts mis en œuvre par le monde du vivant pour la résolution de problèmes.

La source d'inspiration que constitue la biologie a de plus en plus de succès dans une branche de l'intelligence artificielle que l'on peut nommer informatique biomimétique. Le principe de base de cette méthode de développement suit presque invariablement le même schéma et les mêmes motivations comme le fait remarquer Arkin (Arkin, 1998) pour le domaine de la robotique biomimétique (figure 1). Ce schéma s'applique à tout autre domaine, comme par exemple la mécanique ou l'informatique. Ce dernier domaine sera précisément le cadre principal dans lequel se place ce travail de thèse.

L'objectif du processus de création est de concevoir un modèle résolvant un problème ou une catégorie de problèmes en s'inspirant de schémas de comportements mis au point en éthologie. La première étape se base sur des études menées en biologie et consiste à extraire de ces études un modèle réalisable du point de vue informatique. Les résultats obtenus par des simulations permettent ensuite d'évaluer la qualité du modèle relativement aux études biologiques. Les conséquences peuvent être utiles aux informaticiens afin d'améliorer le modèle lui même ou aux éthologistes qui peuvent développer et tester leur théories du comportement animal. Le double intérêt des deux disciplines n'est pas toujours immédiat et il faut reconnaître que c'est le plus souvent les informaticiens qui profitent d'abord de ces développements. Comme nous le verrons dans le chapitre 2, le domaine des fourmis artificielles figure parmi les plus récents, la génétique des populations avec les algorithmes évolutionnaires (Holland, 1975) ou

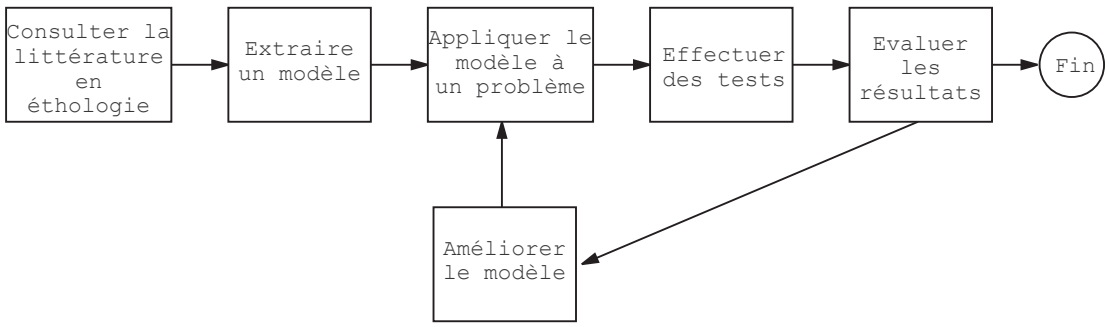


FIG. 1 – Méthodologie de conception en informatique/robotique biomimétique (adapté de (Arkin, 1998)).

la neurobiologie avec les réseaux de neurones (McCulloch and Pitts, 1943) ayant déjà donné lieu à de nombreux travaux depuis quelques décennies. Au chapitre 7, nous suivrons fidèlement ce schéma pour la modélisation du comportement de fourragement des fourmis de l'espèce *Pachycondyla apicalis*.

Sujet de la thèse

Ce travail de thèse vise plusieurs objectifs, dont le point commun est l'étude des fourmis artificielles. Tout d'abord, nous proposons d'étendre certains travaux menés dans le cadre des fourmis artificielles. Nous proposons en particulier d'étendre les travaux de LUMER et FAIETA (Lumer and Faieta, 1994) concernant la capacité de regroupement d'objets par les fourmis pour le problème de la classification non supervisée. Nous tentons ensuite de rapprocher un certain nombre de travaux inspirés des fourmis pour l'optimisation combinatoire d'une classe d'algorithmes évolutionnaires, les algorithmes basés sur la fréquence d'apparition des gènes. Ces développements permettent de constater un certain nombre de similitudes entre les algorithmes BSC et PBIL d'une part et ACO d'autre part. Enfin nous proposons une modélisation d'une espèce de fourmis particulière (*Pachycondyla apicalis*) pour la résolution de problèmes d'optimisation. Les trois objectifs que nous nous sommes efforcés de suivre sont donc les suivants :

- étendre un algorithme inspiré des fourmis pour la classification non supervisée ;
- comparer une classe d'algorithmes de fourmis à des algorithmes évolutionnaires ;
- proposer un nouvel algorithme biomimétique d'optimisation.

Organisation du document

Voici un guide de lecture pour aborder ce document :

- le chapitre 1 décrit les principales caractéristiques des fourmis réelles qui peuvent être exploitées pour la résolution de problèmes d'ingéniérie. Bien que cet aperçu

soit obligatoirement trop succinct en regard des travaux menés dans le domaine de la myrmécologie ou de l'étude du comportement des insectes sociaux (« enthomoéthologie » en quelque sorte), ce chapitre est le plus « biologique » de cette thèse. Son principal objectif est de mettre en appétit le lecteur curieux et de décrire quelques concepts utiles par la suite ;
- le chapitre 2 décrit un certain nombre de travaux en informatique qui ont été inspirés par les fourmis. C'est en quelques sorte l'état de l'art du domaine des fourmis artificielles si tant est que l'on puisse parler de domaine de recherche ;
- le chapitre 3 est une courte introduction au problème de la classification non supervisée, nous y présentons quelques méthodes de résolution classiques ;
- le chapitre 4 présente les travaux que nous avons menés sur la classification non supervisée en s'inspirant des fourmis ;
- le chapitre 5 introduit le problème de l'optimisation numérique et donne des points d'entrée sur les principales heuristiques d'optimisation ;
- une comparaison des algorithmes évolutionnaires avec une adaptation d'un algorithme basé sur les fourmis est présentée au chapitre 6. Nous mettons en avant les différences et similitudes de ces deux approches, notamment d'un point de vue expérimental ;
- la modélisation d'une espèce de fourmis (*Pachycondyla apicalis*) est proposée au chapitre 7 pour la résolution de problèmes d'optimisation numérique et combinatoire. Un certain nombre d'applications dans différents domaines sont proposées et expérimentées ;
- une conclusion sur l'ensemble de ces travaux ainsi que les perspectives d'avenir sont données à la fin du document.

Chapitre 1

Les fourmis réelles

Dans ce chapitre, nous donnons un aperçu des principales caractéristiques des fourmis réelles, par opposition aux fourmis artificielles que nous traiterons dans le chapitre suivant, et de quelques uns de leurs comportements qui susciteront par la suite un certain nombre de développements en informatique. L'étude des sociétés animales donne en effet un champ d'inspiration important pour la résolution de problèmes complexes. Nous survolons dans ce chapitre les propriétés que les fourmis possèdent et qui ont inspiré ou inspireront les sciences telles que l'informatique.

1.1 Introduction

L'étude des fourmis a longtemps été négligée par les enthomologistes alors qu'elles représentent un des stades d'évolution le plus abouti dans le monde des insectes. Hölldobler et Wilson ont corrigé cette lacune en 1990 en publiant un ouvrage concentrant tout ce que l'on connaissait alors des fourmis (Hölldobler and Wilson, 1990)[1] :

> « The neglect of ants in science and natural history is a shortcomming that should be remedied, for they represent the culmination of insect evolution, in the same sense that human beings represent the summit of vertebrate evolution. »

Les fourmis constituent à l'heure actuelle un support majeur pour les théories développées en écologie comportementale et en sociobiologie. On peut citer plusieurs raisons à cet engouement :

– l'influence des fourmis sur leur environnement naturel est extrèmement important. Il a par exemple été montré qu'elles déplacent plus de terre en forêt tropicale que les vers de terre, ou encore que le poids total des fourmis sur terre est du même ordre de grandeur que le poids des humains. De plus, la domination des

[1] Une version allégée s'adressant à un public plus large a été publiée en 1996 par ces mêmes auteurs (Hölldobler and Wilson, 1996).

fourmis est une preuve de leur adaptation à des environnements très variés : « Dans la forêt vierge amazonienne du Brésil, le poids sec de l'ensemble des fourmis est environ quatre fois supérieur à celui de tous les vertébrés terrestres (mammifères, oiseaux, reptiles et amphibiens) réunis » (Hölldobler and Wilson, 1996). On trouve ainsi des fourmis dans tous les écosystèmes terrestres situés entre les deux cercles polaires. La connaissance de leur mode de vie est donc primordiale à la compréhension des espèces animales et végétales qui les côtoient ;
- l'étude des fourmis se fait assez facilement en laboratoire car elles s'adaptent sans trop de difficultés à des environnements différents de leur habitat d'origine ;
- et comme nous allons le voir dans la suite, les fourmis possèdent une gamme de comportements très variés, collectifs ou individuels.

1.2 Les insectes sociaux

1.2.1 Le succès écologique

La place des fourmis dans l'étude des sociétés animales est centrale car elles ont développé des formes très avancées de socialité allant jusqu'à partager leur activité de reproduction en confiant la transmission de leurs gènes à quelques individus de la colonie (les reines et les mâles).

Le nombre d'espèces sociales (environ 13 500 connues (Hölldobler and Wilson, 1996)) est assez réduit par rapport au nombre d'espèces d'insectes répertoriées, soit environ 750 000, alors que les insectes sociaux représentent la moitié de la biomasse des insectes. La grande diversité des fourmis (environ 10 000 espèces connues (Hölldobler and Wilson, 1996)) propose une large variété de morphologies et de comportements. L'étude des fourmis, la *myrmécologie*, est donc un vaste et passionant champ d'investigation.

1.2.2 La colonie en tant que super-organisme

Une fourmilère peut aussi être assimilée à un super-organisme s'apparentant à un organisme vivant composé de cellules. Chaque cellule remplit un rôle précis tout comme les fourmis accomplissent certaines tâches pour la survie et le développement du nid. Les fourmis sexuées ont par exemple le même rôle que les cellules sexuelles. Les parallèles entre ces deux types de systèmes sont étonnants et l'étude de leur développement, la morphogénèse pour un organisme et la sociogénèse pour une société d'insectes, permettent de faire certains rapprochements. L'étude de la sociogénèse a l'avantage d'être plus facile à réaliser puisque l'on peut retirer et injecter des constituants sans mettre en péril le développement du super-organisme.

1.2.3 Les autres sociétés

Les mammifères, les poissons et les oiseaux présentent aussi des comportements sociaux en ce qui concerne la reproduction avec des relations de dominance ou la recherche de nourriture avec des stratégies de coopération pour la capture de proies. En ce qui concerne l'intelligence collective du système immunitaire, la question est

par exemple exposée par John Stewart et Francisco Varela dans (Bonabeau and Theraulaz, 1994, chapitre 4).

1.2.4 Définition de la socialité

On peut distinguer la vie en colonie de la vie en société. Dans le premier cas, c'est la continuité organique entre les individus qui forme la colonie (il y a peu d'exemples), alors que la vie en société est caractérisée par les échanges de signaux entre ses individus, la société est alors une unité qui repose sur le transfert d'informations (Bonabeau and Theraulaz, 1994, chapitre 3).

Toutes les fourmis vivent en société. La *socialité* des insectes est dite avançée (*éosocialité*) si l'espèce présente les caractéristiques suivantes (Jaisson, 1993) :

Définition 1.1 *L'éosocialité est caractérisée par :*
- *la superposition, dans un même groupe social, de plusieurs générations adultes ;*
- *la cohésion entre les membres du groupe ;*
- *la division des rôles avec spécialisation d'un nombre restreint d'individus dans la fonction reproductrice, les autres étant stériles (au moins partiellement) et engagés dans des actes altruistes ;*
- *l'élevage coopératif des jeunes.*

1.2.5 La sélection de parentèle

L'apparition de la socialité n'a pas été facile à expliquer par les théories de l'évolution. La caractéristique la plus étonnante des insectes sociaux est qu'un certain nombre d'individus sont stériles. La théorie de l'évolution de Darwin suppose que les êtres vivants ont pour principale motivation, outre de survive (« l'instinct de survie »), de transmettre leur patrimoine génétique. Comment expliquer alors que certaines espèces aient développé des individus stériles et que l'activité de reproduction se soit concentrée sur quelques individus ? Darwin (Darwin, 1859) reconnaissait que les fourmis représentaient à ses yeux « une difficulté toute particulière, qui [lui] parut tout d'abord insurmontable, voire fatale à l'ensemble de [sa] théorie » puis avait suggéré une sélection familiale des individus. L'explication la plus communément admise est que l'avantage en regard de la sélection naturelle que les insectes peuvent tirer de la vie en société provient de la mise en commun de leurs efforts pour assurer la survie de leur descendance et la propagation de leur patrimoine génétique. La *sociobiologie* (Hamilton, 1964; Wilson, 1975; Jaisson, 1993), avec le principe de la *sélection de parentèle* (*kin selection*), a prolongé les pistes laissées par Darwin pour expliquer l'apparition de la socialité chez les insectes.

La règle de Hamilton (Chapuisat and Keller, 1997) exprime le rapport entre le coût d'un *acte altruiste* et le bénéfice que peut en retirer son auteur. Si le degré de parenté entre deux individus est r, le coût de l'acte altruiste c et le bénéfice pour l'individu qui en profite b, alors l'acte altruiste sera favorisé lorsque :

$$br - c > 0 \tag{1.1}$$

Les bénéfices et coûts sont mesurés en nombre de descendants. Ainsi, pour les humains, deux sœurs ont en moyenne la moitié de leur génome en commun (pour chaque couple de chromosomes, la mère en fournit un parmi les deux qu'elle possède, de même pour le père). Pour qu'une des sœurs puisse trouver un avantage à aider sa sœur, il faut donc que son sacrifice (l'abandon de sa fécondité) double la fécondité (le succès reproductif) de sa parente.

Les hyménoptères sont *haplodiploïdes* : les femelles sont *diploïdes* (2 jeux de chromosomes) et les mâles *haploïdes* (1 jeux de chromosomes). Cette caractéristique permet de rendre l'investissement altruiste rentable. En effet, deux sœurs sont génétiquement plus proches l'une de l'autre (3/4) que de leur frère (1/4) comme il est indiqué dans la figure 1.1.

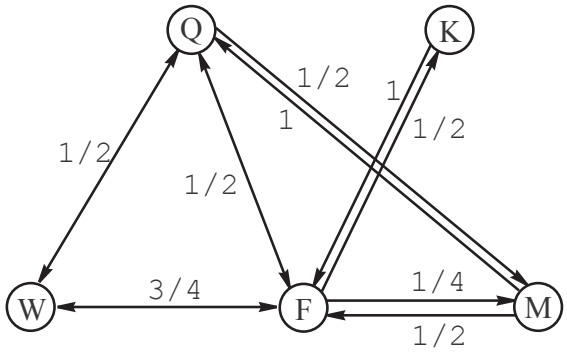


Fig. 1.1 – Distances génétiques chez les fourmis (tiré de (Hölldobler and Wilson, 1990)). Q correspond à la reine fondatrice, K au mâle fondateur, W à une ouvrière, F à une future reine et M à un futur mâle.

La règle de Hamilton ne garantit cependant pas l'abandon de la fécondité pour une partie de la population, elle explique que l'apparition du « gène altruiste » puisse perdurer et se propager.

On peut remarquer que le terme d'*altruisme*, signifiant « amour d'autrui », peut aller très loin chez les insectes puisque certains espèces tropicales d'Asie (du groupe *Camponotus saundersi*) vont jusqu'au suicide pour protéger leur colonie (Hölldobler and Wilson, 1990).

Bien que la question de l'origine de la socialité chez les fourmis soit passionnante et qu'elle fut génératrice de nombreuses polémiques (voir (Jaisson, 1993)), nous n'en dirons pas plus ici car à notre connaissance aucun système artificiel ne s'est inspiré de ces théories.

1.3 L'intelligence collective des fourmis

De par la grande diversité des écosystèmes colonisés (des forêts vierges aux déserts), les fourmis offrent une grande diversité de comportements et de morphologies. L'étude

précise de leur comportement (l'*ethologie*) est souvent limitée aux espèces les moins populeuses pour des raisons pratiques évidentes d'étude en laboratoire.

Cette diversité exubérante est une mine d'inspiration fascinante pour les systèmes informatiques. C'est ainsi que les capacités des fourmis en matière de coopération, de communication, de compétition et d'apprentissage, entre autres, peuvent être mises à profit pour la conception de robots ou d'algorithmes de résolution de problèmes. La suite de cette section présente les principales caractéristiques des fourmis que l'on pourra retrouver dans des systèmes informatiques.

1.3.1 La communication

Les insectes sociaux en général, et les fourmis en particulier, ont développé des *mécanismes de communication* très élaborés (voir (Vander Meer et al., 1998) pour les insectes sociaux et (Brossut, 1996) pour les animaux en général). Il a été défini douze types de réponse mettant en œuvre une forme de communication (Hölldobler and Wilson, 1990) :

1. l'alarme ;
2. l'attraction simple ;
3. le recrutement (pour une source de nourriture ou un site de nidification) ;
4. l'entretien et la mue ;
5. la trophalaxie (échange de liquides) ;
6. l'échange d'aliments solides ;
7. les effets de groupe (augmentation ou inhibition d'une activité) ;
8. la reconnaissance des apparentés ou de caste ;
9. la détermination de caste ;
10. la compétition pour la reproduction ;
11. le marquage du territoire et du nid ;
12. la reproduction (différenciation du sexe, de l'espèce, de la colonie...).

La communication chimique est de loin la plus présente chez les fourmis. Les *phéromones* (mélange d'hydrocarbures) sont à la base de la communication de nombreuses espèces. La *chémoréception* présente les avantages suivants (Bonabeau and Theraulaz, 1994, chapitre 3) :

- la diversité des molécules pouvant intervenir permet de fournir des informations qualitatives ;
- la stabilité du signal pour une molécule peu volatile permet d'assurer une certaine permanence.

Par contre, les principaux inconvénients de la communication chimique sont les suivants :

- elle n'offre que peu d'informations sur la direction ;
- sa propagation est relativement lente et elle est peu adaptée pour la transmission de messages urgents ou pour l'intégration de deux stimulations successives sous une forme temporelle.

Les ouvrières sont par exemple capables de déposer des traces chimiques sur le trajet qu'elles empruntent pour ramener de la nourriture. Au delà du fait que ce marquage leur permet de retrouver leur chemin jusqu'à la fourmilière pour ce qui est du retour et jusqu'à la source de nourriture pour ce qui est d'exploiter une source abondante, cela leur permet de transmettre à leur congénères l'emplacement de l'aubaine.

La communication chimique est aussi mise à l'œuvre pour déclencher des alarmes quand le nid est attaqué et ainsi mobiliser un grand nombre d'individus pour défendre la fourmilière.

Ces deux mécanismes font partie des comportements de *recrutement*. De plus, plusieurs phéromones peuvent être utilisées et avec des concentrations différentes, constituant ainsi une sorte de langage chimique. Les principales manifestations du recrutement sont la recherche de nourriture, la construction du nid, la défense de la colonie et la migration vers de nouveaux sites de nidification.

Bien que peu répandue, certaines espèces ont développé une forme de communication acoustique soit en utilisant un grattoir ou en utilisant les vibrations du sol. Les mouvements peuvent aussi servir de canal de communication : certaines fourmis tisserandes se livrent à une sorte de danse pour recruter des ouvrières. On trouve aussi des ouvrières qui transportent d'autres ouvrières pour leur indiquer le nouvel emplacement du nid (Hölldobler and Wilson, 1996). La communication tactile entre aussi en jeu dans de nombreux rituels d'invitation et de recrutement. Enfin la communication visuelle est assez difficile à mettre en évidence mais certaines espèces semblent utiliser ce canal pour déclencher des mouvements collectifs notamment lors de l'attaque de proies.

La communication entre les individus peut se faire directement ou indirectement. L'utilisation des phéromones est majoritairement une forme indirecte puisque l'échange d'information se fait grâce au support du sol[2]. Quand deux individus interagissent indirectement en modifiant l'environnement on parle de *stigmergie*. Ce terme a été introduit par Grassé (Grassé, 1959) à propos des mécanismes collectifs de contruction du nid chez les termites.

Les différentes applications informatiques qui découlent des capacités de communication des fourmis se retrouvent par exemple en optimisation combinatoire où la coopération stigmergétique s'applique parfaitement à la recherche du plus court chemin dans un graphe. Ces applications seront détaillées dans le chapitre suivant.

1.3.2 La division du travail

Une des caractéristiques particulièrement intéressante est la capacité des sociétés d'insectes à se partager le travail. Les tâches que doivent accomplir les ouvrières sont en effet multiples :

- la recherche de nourriture ;
- la défense du nid ;
- l'entretien et la construction du nid ;
- l'entretien des larves et leur approvisionnement en nourriture.

[2]La communication chimique n'est pas indirecte dans tous le cas, par exemple dans le cas des phéromones de proximité qui ne nécessitent pas de support. Ce cas se retrouve pour les mécanismes d'alarme ou de recrutement à courte distance.

Toutes ces activités, dont l'importance est variable dans le temps et l'espace, doivent être assurées simultanément pour la survie et le développement de la colonie. C'est essentiellement la plasticité de l'organisation déployée par les fourmis qui nous intéresse. Il a été mis en évidence que certains groupes d'individus se spécialisent dynamiquement pour une tâche particulière. Cette dynamique peut être mise en œuvre pour un individu particulier : sa tâche de prédilection varie dans le temps, dans ce cas toutes les ouvrières sont potentiellement capables d'accomplir n'importe quelle tâche. On trouve aussi des spécialisations morphologiques, avec par exemple des variations de taille de un à dix à l'intérieur de la même espèce. Dans ce cas la dynamique est assurée par un contrôle des naissances sur chaque type de morphologie.

Nous présentons dans le chapitre suivant certaines modélisations de cette capacité d'auto-organisation des fourmis qui peuvent générer des applications dans le domaine de la robotique mobile par exemple.

1.3.3 La construction du nid

L'architecture des nids construits par les fourmis est un exemple frappant de structure complexe. L'intérêt pour des modèles pouvant expliquer l'apparition de telles structures provient encore une fois de l'organisation distribuée qui est sous-jacente. Il n'y a pas, a priori, de contrôle centralisé, de coordination de niveau supérieur à l'individu. La structure émerge des interactions inter-individuelles et avec l'environnement. La communication indirecte entre les individus est là encore mise à profit.

Les fourmis tisserandes sont par exemple capables d'unir leurs efforts pour rapprocher des feuilles en formant de véritables ponts puis d'unir les bords des feuilles en utilisant la soie produite par leurs larves (Hölldobler and Wilson, 1996). La construction du nid chez les termites à été étudiée par Deneubourg (Deneubourg, 1977). L'apparition des piliers dans une termitière pouvant être expliquée par l'amplification de multiples fluctuations chaotiques : la struture, modèle d'équilibre des forces par sa stabilité, nait de l'amplification de multiples déséquilibres. La construction des nids de guèpes est aussi un modèle d'action collective où les agents répondent plus particulièrement à des stimuli issus de certaines configurations dans la structure (Bonabeau et al., 1999).

1.3.4 La quête de nourriture

La recherche de la nourriture (le *fourragement*) est une activité souvent plus dispersée spatialement que la construction du nid et qui peut aussi être mise en œuvre de façon très différente suivant les espèces de fourmis. Les stratégies de recherche de nourriture sont en effet extrèmement diversifiées. Par exemple à cause des différences de régime alimentaire : certaines espèces peuvent être spécialisées sur un unique type d'aliment. On peut aussi trouver des mécanismes très élaborés comme la culture de champignon ou l'élevage de pucerons. La recherche de nourriture est une activité risquée (les fourrageuses de *Cataglyphis bicolor* ont par exemple une espérance de vie de 6.1 jours (Hölldobler and Wilson, 1990)) mais souvent efficace (la quantité de nourriture ramenée au nid au cours d'une vie représente de 15 à 20 fois le poids d'un individu).

La communication peut avoir un impact important, en particulier pour les mécanismes de recrutement dont le principal intérêt collectif est de rassembler les ouvrières sur les sources de nourriture rentables. D'un point de vue plus général, la communication mise en œuvre pour la recherche de nourriture peut être considérée comme une forme de mémoire collective quand elle s'appuie sur la modification de l'environnement telle que l'utilisation des phéromones.

Nous verrons au chapitre 7 que les statégies de fourragement les plus simples sont aussi une source d'inspiration intéressante pour des systèmes artificiels.

1.4 Capacités individuelles

Les capacités individuelles des fourmis peuvent servir de modèle à des systèmes artificiels tant leur adaptation à leur environnement peut être efficace. Nous citons par exemple les points suivants :

- individuellement, une fourmi possède certaines capacités d'apprentissage, et notamment quand elle se déplace autour du nid. Les expériences de Schatz et ses collègues montrent que les fourmis du genre *Cataglyphis* sont capables d'apprendre visuellement des routes familières pour se déplacer entre un site alimentaire et leur nid (Schatz et al., 1999) ;
- du point de vue physique, certaines espèces ont des capacités étonnantes comme les fourmis *Gigantiops destructor* capables de faire des bonds impressionants et dotées de capacités visuelles inhabituelles ce qui les a rendues difficiles à observer (Chagné et al., 1999).

La plupart des caractéristiques qui intéressent l'informatique sont cependant collectives. Les caractéristiques individuelles ne sont évidemment pas une particularité des fourmis mais de tous les organismes vivants ayant un souci de survie.

1.5 Les processus d'auto-organisation chez les insectes sociaux

Les théories rassemblées sous le terme d'auto-organisation ont originellement été développées en physique ou en chimie pour décrire l'émergence de phénomènes macroscopiques à partir d'interactions et de processus définis au niveau microscopique. L'auto-organisation se prête bien à l'étude des insectes sociaux qui montrent des comportements collectifs complexes issus de comportements individuels simples. On peut regrouper les processus d'auto-organisation chez les insectes sociaux en quatre groupes tant leur diversité est importante :

- le recrutement et l'exploitation collective de sources de nourriture : le fourragement met à jour des stratégies qui permettent aux insectes une grande adaptation à leur milieu ;
- la division du travail et l'organisation des rôles sociaux : à l'intérieur d'une même société, on peut observer différentes castes spécialisées dans un certain nombre de tâches (élevage du couvain, recherche de nourriture, construction du nid, ...) ;

- l'organisation de l'environnement : la construction du nid est un symbole de l'organisation distribuée des insectes. Le nid est construit sans que les insectes soient dirigés, ils répondent à un certain nombre de stimuli provenant de leur environnement ;
- la reconnaissance inter-individuelle : chaque fourmi est capable d'identifier ses congénères tout en participant elle même à l'identité de sa colonie (l'échange d'aliments entre les individus d'une même colonie, la *trophalaxie*, est un exemple d'acte altruiste permettant en plus d'homogénéiser l'identité de la colonie, l'identité coloniale). Les explications du mécanisme de reconnaissance ne sont pas encore parfaitement établies. Cependant, il s'avère qu'il y ait à la fois une composante génétique et une compasante acquise par apprentissage. Un réseau de neurones a par exemple été utilisé pour reproduire ce mécanisme d'apprentissage puis de différenciation entre les composés chimiques, dans le cas des termites (Bagnères et al., 1998).

Ces processus d'auto-organisation sont à l'origine de ce que l'on dénomme l'intelligence collective. On parle d'intelligence collective quand un groupe social peut résoudre un problème dans un cas où un agent isolé en serait incapable (Bonabeau and Theraulaz, 1994). Nous illustrons dans le chapitre suivant quelques uns des développements informatiques pouvant se rattacher à ces notions d'auto-organisation et d'intelligence collective.

1.6 Conclusion

Dans ce chapitre nous avons survolé les principales caractéristiques des fourmis trouvant des développements artificiels. Nous verrons que ces développements ne visent pas obligatoirement des problèmes industriels. Par exemple en vie artificielle, de nombreux chercheurs s'intéressent au fonctionnement et à l'apparition de l'intelligence collective et utilisent l'informatique et la modélisation mathématique comme outils de prospection. L'objet du chapitre suivant est de présenter un certain nombre de travaux que l'on peut rassembler sous la notion assez large de fourmis artificielles.

Chapitre 2

Les fourmis artificielles

Ce chapitre présente les principaux algorithmes développés autour du thème des fourmis artificielles. Nous passons en revue certains travaux en robotique, en optimisation et en vie artificielle avant de décrire brièvement les domaines pouvant être considérés comme voisins des fourmis artificielles.

2.1 Introduction

Le domaine des fourmis artificielles existe-t-il ? Il est peut-être encore trop tôt pour pouvoir réellement parler du domaine des fourmis artificielles, même si la première conférence y étant consacrée s'est déroulée à Bruxelles en 1998 (ANT'S 98 (Dorigo, 1998))[1]. On retrouve donc majoritairement les communications consacrées aux fourmis artificielles dans les conférences consacrées aux algorithmes évolutionnaires (Männer and Manderick, 1992; Voigt et al., 1996; Eiben et al., 1998a; IEEE, 1997; Prieditis and Russell, 1995), à la vie artificielle (Varela and Bourgine, 1991; Husbands and Harvey, 1997; Floreano et al., 1999) et à la simulation des comportements adaptatifs des animaux (Meyer and Wilson, 1990; Cliff et al., 1994; Pfeifer et al., 1998). Les revues et journaux pouvant accueillir des publications sur le thème des fourmis articielles sont dans le même courant et plus globalement en intelligence artificielle (Adaptive Behavior, Artificial Life, BioSystems, Journal of Complex Systems, Journal of Artificial Intelligence Research, Evolutionary Computation). En ce qui concerne les autres sources de publication, on trouve dans les revues centrées sur un domaine particulier des applications des fourmis artificielles (Journal of Operation Research Society, ...). Depuis peu, certains ouvrages y consacrent un ou plusieurs chapitres complets (Bonabeau and Theraulaz, 1994; Bonabeau et al., 1999; Corne et al., 1999) ou encore des numéros spéciaux (Future Generation Computer Systems journal).

À la place du domaine des fourmis artificielles, il est plus opportun de parler d'intelligence collective artificielle ou d'intelligence en essaim (*Swarm Intelligence*) (Bonabeau and Theraulaz, 1994; Bonabeau et al., 1999).

[1] la deuxième a eu lieu en septembre 2000 : ANT'S 2000 (Dorigo et al., 2000).

On peut tenter de regrouper les travaux portant sur les fourmis artificielles en trois catégories. Dans les sections suivantes de ce chapitre, nous présentons tout d'abord un certain nombre d'exemples d'applications ou de modèles pouvant se rapporter à la robotique. Ensuite nous présentons les travaux se rapportant à l'optimisation, actuellement en plein développement. Enfin nous décrivons quelques expériences menées en vie artificielle. Nous présentons également quelques points d'entrée sur des notions voisines ou relatives aux fourmis artificielles.

2.2 Robotique collective

Une présentation exhaustive des travaux s'inspirant des insectes sociaux et de leurs comportements collectifs pour la résolution de problèmes de robotique dépasse largement le cadre donné à ce tour d'horizon. Bien que le sujet de ce travail ne soit pas centré sur la robotique, il nous est apparu important de ne pas négliger cette perspective. En effet, le travail présenté par la suite, à savoir la résolution de problèmes de classification non supervisée et l'optimisation, n'impliquent pas, a priori, de préoccupations concernant la robotique. Il y a cependant un certain nombre de raisons justifiant de présenter ici les travaux de robotique collective :

1. la robotique, en tant que science de la construction de robots, et l'informatique, en tant que science de la résolution de problèmes combinatoires, sont deux disciplines en interaction permanente : l'informatique est utilisée pour mettre au point des robots grâce à ses capacités de simulation et les robots nécessitent la programmation d'algorithme d'apprentissage ;
2. nous verrons dans la suite de ce chapitre que les insectes sociaux ont inspiré les roboticiens et les informatiens pour des raisons très similaires, les premières applications ayant été développées pour des problèmes de robotique.

Avant de considérer les travaux menés en robotique collective, il faut rappeler que les insectes sont une source d'inspiration intéressante en ce qui concerne la conception de robots dits solitaires. Ces travaux n'utilisent pas les capacités collectives des colonies d'insectes mais s'inspirent plutôt des capacités des insectes en matière de perception ou de mobilité. On trouve par exemple des robots capables d'utiliser la lumière polarisée pour se diriger (Lambrinos et al., 1997) comme l'utilisent certaines espèces de fourmis (Hölldobler and Wilson, 1990). L'utilisation des phéromones a aussi été mise en œuvre sur un robot. Russel et ses collègues (Russell et al., 1994) ont ainsi construit un robot capable de se diriger en suivant des traces de camphre. L'utilisation de la chaleur comme information environnementale a aussi été envisagée. Les capacités motrices des insectes ont inspiré de nombreux robots hexapodes, utilisant des réseaux de neurones artificiels pour en contrôler le comportement (Quinn and Espenschield, 1993, par exemple). La vision des fourmis et leurs capacités de navigation sont aussi étudiées pour la construction de robots autonomes. Nous avons développé en collaboration avec le LEPA (Laboratoire d'Ethologie et Psychologie Animale) de Toulouse un robot reproduisant les stratégies de navigation des fourmis (Françoise, 1999; Françoise et al., 1999; Duvaux, 2000).

Du point de vue de l'architecture utilisée pour construire des robots autonomes reproduisant les comportements des insectes, la plupart des systèmes implémentés fonctionnent sur le même principe, introduit par Brooks (Brooks, 1986), à savoir l'architecture de subsomption. Contrairement aux approches basées sur le raisonnement artificiel, l'architecture de subsomption décompose le comportement du robot en primitives hiérarchisées de réponses à des stimuli provenant de l'environnement ou des autres robots. Cette architecture se prête bien à la reproduction des comportements rencontrés dans la nature puisqu'elle évite de faire des suppositions sur le mode de raisonnement des animaux.

Nous renvoyons le lecteur intéressé par la conception de robots mobiles et autonomes à (Kaelbling, 1993) où sont présentés les différentes approches de l'apprentissage dans des environnements complexes et à (Kortenkamp et al., 1998) où sont présentés un certain nombre de cas réels concernant les problèmes de navigation, de vision et d'architecture.

Les fourmis, du point de vue des roboticiens, sont de petites créatures relativement simples aux capacités de communication réduites mais capables d'accomplir des tâches complexes. On peut notamment s'attendre à ce qu'une communauté de robots coopératifs soit plus performante (en termes de vitesse et d'efficacité) que le même nombre de robots agissants de manière individuelle. Les raisons suivantes peuvent être évoquées pour justifier l'utilisation d'une colonie de robots (Arkin and Balch, 1998; Arkin, 1998) :

- l'action est distribuée : plusieurs robots peuvent être à plusieurs endroits en même temps ;
- le parallélisme est inhérent : plusieurs robots peuvent accomplir en même temps plusieurs actions identiques ou non ;
- la division du travail : certains problèmes se prêtent bien à être décomposés et profitent d'une spécialisation des robots ;
- la simplicité : les agents d'une équipe de robots peuvent être plus simples qu'un seul robot accomplissant seul la tâche.

De ce dernier point, on peut tirer quelques corollaires :

- un coût réduit : que cela soit du point de vue du matériel que de l'effort de développement, construire plusieurs robots identiques peut être moins coûteux que de construire un robot ayant les mêmes capacités que la colonie de robots ;
- une fiabilité augmentée : un robot simple a des chances d'être plus fiable qu'un robot compliqué.

La simplicité du robot n'est pas la seule raison à prendre en compte pour considérer que la fiabilité du groupe est supérieure, le traitement parallèle et le nombre de robots font que la défection de quelques uns d'entre eux ne met pas en péril l'accomplissement de la tâche.

Quelques inconvénients peuvent cependant apparaître :

- les robots peuvent interférer les uns avec les autres, que cela soit en se gênant physiquement dans leurs déplacements ou dans leurs mécanismes de communication ;
- le coût de communication peut être élevé en énergie ou en traitement et des problèmes de fiabilité peuvent apparaître ;

– la coopération entre les robots peut se transformer en compétition et faire chuter les performances si les robots ne comprennent pas les intentions de leurs congénères.

Les caractéristiques d'un groupe de robots coopératifs sont principalement les suivantes :

– il n'y a pas de contrôle centralisé ;
– il n'y a pas de communication globale (seulement des interactions locales), ce qui présente comme avantage d'être économique du point de vue de l'énergie dépensée pour la communication et ce qui limite la quantité d'informations à traiter par chaque robot. L'inconvénient majeur est qu'il faut prévoir des stratégies de déplacement permettant d'éviter de perdre des robots dans la nature.

Les principaux problèmes traités en robotique collective sont les suivants (Arkin, 1998) :

– le fourragement ou la quête de nourriture (*foraging*) : il s'agit de ramasser un ensemble d'objets aléatoirement distribués dans l'environnement et de les rassembler en un point central. Ce type d'application est parfaitement illustré par la collecte de minerais dans un environnement inadapté à l'homme comme peut l'être une planête étrangère ou de haut fonds marins ;
– la réparation (*consuming*) : les robots doivent accomplir un certain nombre d'opérations sur des objects fixes dans l'environnement (par exemple l'utilisation de robots démineurs) ;
– le nettoyage (*grazing*) : les robots doivent couvrir de façon adéquate une zone de leur environnement. Les principales applications sont des opérations de surveillance, de sauvetage et de nettoyage ;
– le regroupement (*flocking, formation*) : les robots doivent se disposer dans leur espace suivant une certaine géométrie se conservant quand le groupe se déplace ;
– le transport collectif : les robots doivent se rassembler autour de l'objet à déplacer, ce qui peut être considéré comme une forme de fourragement. Les applications peuvent être le déplacement de boites ou le transport de palettes ;

Ces applications sont directement déduites de comportements des fourmis réelles :

– la recherche de nourriture est une activité centrale des fourmis, la multitude des comportements offre une source d'inspiration presque inépuisable pour les problèmes de robotique apparentés à la collecte de minerai ;
– les fourmis pratiquent le partage des tâches et divisent le travail de façon très courante et ceci de façon diversifiée : le polyethisme temporel qui désigne la capacité de certaines fourmis à changer d'activité en fonction de leur age, le polymorphisme des ouvrières désigne les diversités morphologiques qui peuvent apparaitre au sein d'une même colonie permettant à différentes castes d'être spécialisées pour des tâches variées et enfin l'activité d'une même fourmi peut varier en fonction de ses expériences passées ;
– la création de cimetières et le rangements des œufs et larves sont deux exemples de la capacité des fourmis à organiser leur environnement de façon distribuée ;
– le transport coopératif de proies est une illustration de la capacité des fourmis à unir leurs efforts pour accomplir une tâche qui ne pourrait pas être menée à bien par des individus isolés.

Le développement de robots autonomes passe par une phase de simulation mais il est très difficile de prendre en compte tous les paramètres physiques qui interviennent pour le déplacement et la perception des robots. C'est pour cela que les expérimentations réelles sont particulièrement intéressantes pour évaluer la validité d'une méthode.

Les sections suivantes décrivent un certain nombre d'applications en robotique collective pour la résolution de problèmes plus ou moins réels.

2.2.1 Le ramassage d'objets

Goss et Deneubourg (Goss and Deneubourg, 1991) montrent par des simulations comment un groupe de robots simples peut naviguer dans un espace inconnu, sans carte, reconnaissance visuelle ou points de repère. Les robots ont une perception inférieure à la taille de leur espace et ils peuvent rencontrer des obstacles. Cette méthode, moins efficace que des systèmes de guidage plus complexes, présente des avantages de simplicité, de fiabilité et de flexibilité.

Le problème traité est de faire ramasser des objets dispersés dans une surface carrée et de les ramener au point central. Trois modèles de robots sont présentés : les deux premiers correspondent au fourragement individuel des fourmis. Le dernier modèle utilise des marquages temporaires indépendants de la découverte des objets.

Les trois modèles de robots ont les caractéristiques suivantes en commun :
- chaque robot est capable d'éviter les obstacles ;
- la distance de détection des objets est réduite : un objet n'est détecté que quand le robot est placé sur la même case ;
- la capacité de transport d'un robot est limitée à un seul objet ;
- les limites de l'espace de recherche sont considérées comme des obstacles.

Le premier type de robot (dit Robot Basique : RB) présente les caractéristiques suivantes :
- il n'y a pas d'interaction avec les autres robots ;
- la recherche d'objet est aléatoire ;
- le retour au nid est direct : le robot reçoit toujours le signal du nid sur l'espace de recherche ;
- le retour au nid s'effectue quand le robot a trouvé un objet ou quand il abandonne sa recherche ;
- lors du retour au nid, le robot dépose son objet et repart dans une direction aléatoire mais à une distance du nid équivalente à celle de sa prise antérieure.

Le deuxième type de robot présenté (Robot avec Mémoire : RM) possède certaines capacités supplémentaires dont la principale est la mémorisation de l'emplacement de sa dernière découverte. Il y a alors deux façons de procéder :

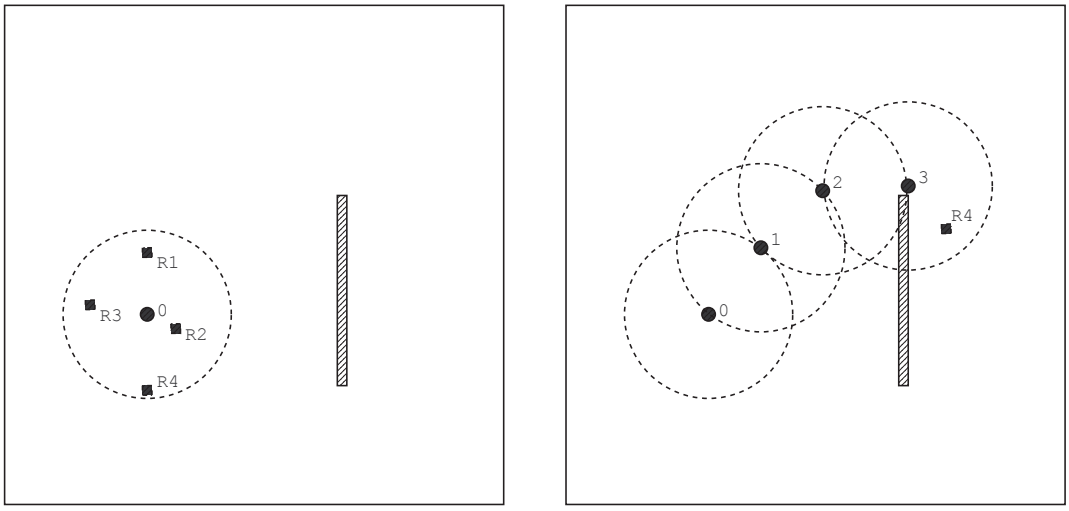
1. déposer un signal sur le site de découverte. Puis retourner à ce site par triangularisation avec le signal émis par le nid ;
2. mémoriser l'angle et la distance par rapport au nid.

Le robot retourne au nid après un certain temps de recherche infructueuse. Il est programmé pour avoir un poids ($0 < w < 1$) associé à sa capacité de mémorisation. Pour des valeurs proches de 1, le robot a une mémoire à court terme et ne se rappelle que

de la découverte la plus récente. Pour des valeurs plus proches de 0, l'influence des découvertes antérieures est plus grande et la mémoire du robot est plus importante. Ce type de robot s'inspire du comportement de fourragement de certaines espèces de fourmis dont on donnera un exemple détaillé dans le chapitre 7 où nous nous intéresserons aux fourmis de l'espèce *Pachycondyla apicalis*.

Le dernier modèle de robot (Robot Fabricant de Chaînes : RFC) s'éloigne des inspiration biomimétiques pour s'adapter à des situations où le signal du nid est faible. Les robots doivent compter les uns sur les autres pour trouver leur chemin. Leur principales caractéristiques sont les suivantes :

- les signaux sont de portée limitée ;
- la recherche est aléatoire ;
- ils possèdent une estimation de la distance à l'origine du signal ;
- quand un robot arrive à la limite de sa perception d'un signal, il cherche un autre signal à capter. Si un nouveau point d'émission a été trouvé, le robot continue sa prospection locale, sinon il se transforme en émetteur avec un numéro d'une unité supérieure à celui qu'il reçoit ;
- Pour retourner au nid, le robot « remonte » les numéros des émetteurs en sens décroissant.



(a) (b)

FIG. 2.1 – (a) Quatre robots fabricant de chaîne dans le rayon d'émission du point central. (b) Les robots se sont déplacés. R1, R2 et R3 se sont transformés en émetteurs (balises). R4 peut explorer derrière l'obstacle grâce au signal du 3.

En comparaison des deux premiers modèles, les robots RFC ne peuvent pas se perdre, sauf en cas de défaillance d'un robot-émetteur, et leur exploration n'est pas limitée par la topologie de leur espace de recherche qui pourrait gêner le retour au nid des robots de type RB et RM car ceux ci reviennent en ligne droite. Leur exploration est plutôt limitée par leur nombre et donc par la longueur des chaînes qu'ils peuvent construire.

Pour traiter le cas où un robot ne peut savoir à l'avance s'il percevra un signal ou non au pas suivant, celui-ci est capable de faire un pas en arrière s'il perd le contact. l'inconvénient de fabriquer une chaîne est que les robots constituant la chaîne deviennent inopérants pour la recherche d'objets. De plus, quand un robot arrive en fin de zone, il devient émetteur ce qui fait que le dernier robot mobile qui sort de la zone devient émetteur et plus aucun robot ne cherche.

Un certain nombre d'ajustements sont proposés :

- les robots chercheurs peuvent abandonner avec une certaine probabilité ;
- la mémorisation des robots émetteurs suivis pour revenir au nid permet de réutiliser ces émetteurs pour retourner sur le site de découverte ;
- les robots émetteurs possèdent un compteur de passages qui décroît avec le temps et qui est augmenté à chaque passage d'un robot chargé d'un objet. En dessous d'un certain seuil, et quand aucun émetteur de numéro supérieur n'est perçu, l'émetteur rentre au nid.

Le résultat de la simulation montre que les robots à mémoire (RM) se spécialisent sur des zones distinctes. Le nombre moyen de pas pour une découverte donne l'avantage aux robots à mémoire (RM) et surtout en petite équipe (10) par rapport aux plus grandes (50, 200). Viennent ensuite les robots fabricants de chaînes (RFC) en grande population. Quand des obstacles sont disposés, seuls les RFC peuvent capturer 75% des objets et ils le font légèrement moins bien que sans obstacles.

Robots démineurs

Les micro-robots développés par McLurkin (McLurkin,) au MIT sont conçus pour résoudre un problème similaire au ramassage d'objets. Un de leurs principaux avantages est leur taille puisqu'ils peuvent tenir dans un cube de 2,6 cm de côté. Voici leurs caractéristiques :

- 3 moteurs (2 pour le déplacement et 1 pour les mandibules) ;
- 17 capteurs (2 détecteurs IR, 2 capteurs de lumière et des détecteurs de collision et de toucher pour la mandibule) ;
- 2 émetteurs IR, les communications sont omnidirectionnelles et de portée limitée.

À l'aide de la définition d'ensemble de couples stimulus-réponse, un certain nombre de comportements collectifs sont expérimentés :

- le jeu du chat et de la souris (*tag game*) ;
- le rassemblement et la dispersion des robots ;
- le rassemblement en relais ;
- la poursuite (à petite et moyenne distance) ;
- mesure de la densité de robots (chaque robot compte les robots qu'il perçoit) ;
- déplacement en essaim (les robots utilisent des bornes inférieures et supérieures de la densité de robots les entourant).

L'application de ces robots est la destruction de mines anti-personnel qui peut être effectuée de différentes manières :

- chaque robot cherche une mine. Une fois celle-ci trouvée, le robot se place à proximité et émet un signal indiquant sa découverte. Quand tous les robots sont en attente, un signal provenant de la base les fait s'autodétruire. Le principal

inconvénient de cette méthode est que les robots deviennent évidemment inutilisables ;
- Les robots rassemblent les mines en un lieu commun afin d'en faciliter la destruction ou le désarmement. Cette méthode est difficile à mettre en œuvre car le transport des mines est délicat ;
- Les robots déposent des charges explosives sur les mines qu'ils trouvent et se mettent à l'abri quand le signal d'explosion des charges est émis ;

2.2.2 Le nettoyage

Le nettoyage et la surveillance d'une zone par une colonie de robots sont des tâches assez similaires. Contrairement à la recherche de minerai où l'efficacité de la collecte importait, il s'agit ici de parcourir une zone le plus complètement possible. Pour une opération de nettoyage, la poussière présente sur le sol peut servir d'indicateur ou de mémoire aux robots (Wagner and Bruckstein, 1995). Dans (Wagner et al., 1999), les auteurs assimilent la zone à couvrir à un graphe. Les nœuds du graphe correspondent à une zone ou une pièce et les arcs correspondent aux connexions entre chaque zone. Chaque nœud du graphe doit être surveillé régulièrement et uniformément. Trois méthodes sont proposées et pour chacune d'elles, une borne supérieure du temps de couverture est fournie en tenant compte des erreurs des capteurs. Une mémoire partagée, assimilable à des traces de phéromones, est stockée sur chaque nœud ou chaque arc du graphe et sert au choix de direction de chaque robot.

2.2.3 Le partage du travail

Theraulaz et ses collègues présentent un modèle de division du travail dans les colonies d'insectes basé sur un seuil de réponse fixe (Bonabeau et al., 1998c) ou variable (Theraulaz et al., 1998; Bonabeau et al., 1998b). Les seuils de réponse correspondent aux chances de réagir à des stimuli associés à des tâches.

Les premiers modèles basés sur des seuils fixes avaient quelques limitations :

1. ils ne prennent pas en compte la mise en place de l'allocation de tâches (polyethisme temporel). Les individus sont pré-affectés ;
2. il n'y a pas de robustesse dans la spécialisation ;
3. les modèles ne sont valides que pour une faible durée ;
4. ces modèles ne sont pas validés par les récentes expérimentations sur les abeilles montrant que le vieillissement et/ou l'apprentissage joue un rôle dans l'affectation des tâches.

Les individus de seuil bas effectuent les tâches à un plus faible stimulus que les individus à seuil élevé. Dans le cas des seuils adaptatifs, parmi les individus travailleurs, l'accomplissement d'une tâche donnée induit une réduction du seuil de réponse correspondant. Ce processus de renforcement combiné conduit à l'émergence de travailleurs spécialisés, ce qui signifie qu'ils sont plus réceptifs à des stimuli correspondants à certaines tâches, alors qu'initialement, tous les individus sont identiques.

La probabilité pour qu'un agent i accomplisse la tâche j sachant que l'intensité du stimulus correspondant à cette tâche perçu par l'agent est s_j est donnée par :

$$P_{\theta_{ij}}(s_j) = \frac{s_j^2}{s_j^2 + \theta_{ij}^2} \tag{2.1}$$

où θ_{ij} correspond au seuil de réponse de l'agent i pour la tâche j. À chaque fois qu'un agent effectue une tâche, le seuil de réponse à cette tâche est diminué alors que les seuils des autres tâches sont augmentés, le tout proportionellement au temps mis pour accomplir cette tâche (Δ_t) :

$$\theta_{ij} \leftarrow \theta_{ij} - x_{ij}\xi\Delta_t + (1 - x_{ij})\varphi\Delta_t \tag{2.2}$$

où x_{ij} représente la fraction de temps passé par l'agent i pour la tâche j, ξ et φ sont deux paramètres de l'apprentissage.

Les dynamiques de la spécialisation sur des tâches résultant du modèle sont étudiées par les auteurs et des observations sont faites sur ce qui doit être observé quand des spécialistes d'une tâche donnée sont retirés de la colonie et sont réintroduits après une durée variable : la colonie ne retrouve pas le même état précédant la perturbation. Plus la durée d'absence est élevée, plus la différence d'état de la colonie est marquée.

Bien que ces travaux ne concernent pas obligatoirement la robotique, c'est en robotique collective qu'ils trouvent leurs applications les plus immédiates. On peut par exemple citer les travaux de Krieger et Billeter (Krieger and Billeter, 1999) où des robots Khepera implémentent ce système d'allocation de tâche pour la récolte de graines afin de maintenir un certain niveau d'énergie du nid (avec cependant des seuils de réponse différents pour chaque robot mais fixes dans le temps).

2.2.4 Le transport collectif

De nombreux travaux se sont attachés à montrer que l'on pouvait faire coopérer plusieurs robots sans qu'il y ait de communication directe entre eux pour leur faire accomplir des tâches où la coordination de leurs efforts est nécessaire. Le problème type en robotique est de pousser des boîtes trop lourdes pour un seul robot. Cela est très similaire au problème que peuvent rencontrer les fourmis quand elle doivent ramener au nid une proie trop grosse pour une fourmi[2]. Il a par exemple été montré que la capacité de transport d'un groupe de fourmis pouvait être parfois bien supérieure à la somme des capacités de transport individuelles.

Les premiers travaux sont de Kube et Zhang (Kube and Zhang, 1992) (voir aussi (Kube and Bonabeau, 1999) pour un exposé plus récent). Les robots possèdent quelques primitives de réponse à leur perception limitée de leur environnement proche, notamment en ce qui concerne leur but (s'approcher de la boîte), leurs collègues (suivre un autre robot) et les obstacles (s'en éloigner). Comme pour les fourmis, la principale difficulté rencontrée est une situation de bloquage : la somme des forces appliquées par

[2]Certaines fourmis découpent la proie pour la transporter, ce n'est évidemment pas cette technique qui nous intéresse ici.

les robots s'annule ou alors la boîte est bloquée par un obstacle. La stratégie d'une fourmi est dans un premier temps de changer son alignement par rapport à sa proie, puis en cas d'échec elle se repositionne et change de prise.

2.2.5 Le rassemblement d'objets

Le rassemblement d'objets se distingue du ramassage d'objets dans le sens où les objets n'ont pas à être ramenés à un point précis représentant souvent le nid. Le rassemblement d'objets consiste à créer des groupes d'objets (*clusters*).

Beckers et ses collègues ont présenté des robots capables de pousser des palets (*puks*) grâce à une pelle en forme de « C » (Beckers et al., 1994). Les robots sont équipés d'un capteur obligeant le robot à faire demi-tour lorsqu'il pousse plus de 3 palets. Tant qu'un obstacle n'est pas détecté et que la pelle n'est pas pleine, le robot avance en ligne droite. Quand un obstacle est rencontré le robot change de direction aléatoirement. Si la pelle contient plus de trois palets, le robot recule pour libérer les palets et change de direction aléatoirement. Enfin, pour chaque expérience, les palets sont initiallement disposés régulièrement dans une arène de forme carrée. Les résultats obtenus montrent trois phases :

1. de nombreux groupes de palets formés de trois objets se forment rapidement puis leur taille augmente, rendant leur disparition plus difficile ;
2. quelques groupes de taille importante subsistent dont les palets ne peuvent être retirés que si le robot se dirige sur le groupe avec un angle adapté ;
3. la troisième phase, la plus longue, mène à la formation d'un seul groupe de palets.

Nous avons reproduit ce comportement en simulation avec des robots uniquement capables de pousser les objets (la pelle n'a plus une forme de « C ») et limités à quatre directions (est, ouest, nord et sud). Tels que nous les avons modélisés, les robots ne distinguent pas la différence entre une paroi de l'arène et un groupe d'objets trop important pour être poussés. Les résultats obtenus sont légèrement différents : on retrouve bien trois phases dans le processus de regroupement mais la dernière phase aboutit au rassemblement des palets dans les quatre coins de l'arène. Il se forme donc quatre groupes de palets au lieu d'un seul. Ceci peut s'expliquer par la différence de forme de la pelle ainsi que le nombre limité de direction que peut prendre un robot. Dans les expériences de Beckers, quand un robot rencontre un obstacle et qu'il pivote, les objets contenus dans sa pelle y restent bloqués ce qui évite la formation de groupes sur les bords. Melhuish (Melhuish, 1999) a proposé plusieurs modifications de la stratégie de Beckers pour que les robots forment un groupe d'objets contre les parois de l'arène. Il a notamment proposé de réduire la capacité des robots à détecter les parois de l'arène, ce qui s'avère être assez similaire à notre simulation.

La simulation nous a permis de tester certaines limites du modèle : on peut donner aux robots une capacité infinie, ce qui est difficile à réaliser en pratique, ce qui signifie que les changements de direction interviennent uniquement quand un robot rencontre un bord de l'arène ou un autre robot. Contrairement à ce que l'on pouvait attendre, les objets sont rassemblés moins rapidement que dans l'expérience où ils étaient de

capacité limitée. On constate en effet que la couverture de l'arène est moins efficace, la direction des robots ne variant que rarement au milieu de la surface.

Un certain nombre d'extensions ont été proposées aux travaux de Beckers. Par exemple en considérant deux types de palets (des rouges et des jaunes) et en ajoutant une règle obligeant les robots à opérer une marche arrière avant de déposer un palet jaune (Melhuish et al., 1998). Cela permet d'obtenir un comportement de tri en plus du rassemblement des objets. Enfin, les deux voies d'expérimentation, sur des robots réels et par simulation, sont complétées par un modèle probabiliste permettant d'obtenir des résultats beaucoup plus rapidement (Martinoli et al., 1999).

2.2.6 Autres problèmes

Nous avons présenté les problématiques les plus souvent rencontrées en robotique collective, on trouve cependant d'autres expérimentations :

- l'auto-organisation spatiale des robots à été étudiée dans (Ünsal and Bay,) où les robots se placent les uns par rapport aux autres formant plusieurs configurations ;
- les capacités de construction des insectes sociaux ont été exploitées en robotique, notamment par l'auto-assemblage de structures (voir un panorama dans (Bonabeau et al., 1999)).

2.3 Optimisation combinatoire

Les fourmis résolvent de nombreux problèmes liés à leur survie. De là à estimer que dans certains cas elles résolvent un problème d'optimisation, il n'y a qu'un pas. Dorigo et ses collègues l'ont franchi en transposant la capacité des fourmis à trouver le plus court chemin entre une source de nourriture et leur nid à un problème classique en optimisation, le problème du voyageur de commerce[3]. Cette section présente les différents travaux s'inspirant des stratégies de recherche de nourriture des fourmis pour résoudre des problèmes d'optimisation. Nous avons conçu un autre exemple de ce type de modélisation, toujours pour résoudre des problèmes d'optimisation, en s'inspirant d'une espèce de fourmis particulières, les *Pachycondyla apicalis* (chapitre 7).

2.3.1 Le Problème du Voyageur de Commerce

Définition du problème

Le Problème du Voyageur de Commerce (PVC), ou *Traveling Salesman Problem (TSP)*, est un classique du genre. Rappelons tout de même sa formulation générale :

Définition 2.1 *Problème du voyageur de commerce. Un voyageur de commerce doit visiter un ensemble* $\{v_1, \ldots, v_n\}$ *de n villes dont on connaît les distances respectives* $d(v_i, v_j), \forall (i,j) \in \{1, \ldots, n\}^2$. *Le problème consiste à trouver la permutation* σ *telle*

[3]Marco Dorigo maintient un site internet dédié à l'optimisation par des fourmis artificielles :`http ://iridia.ulb.ac.be/`*mathttf*`mdorigo/ACO/ACO.html`

que la séquence $s = (v_{\sigma(1)}, \ldots, v_{\sigma(n)})$ *minimise la distance totale* $D(\sigma)$ *parcourue par le voyageur :*

$$D(\sigma) = \sum_{i=1}^{n-1} d(v_{\sigma(i)}, v_{\sigma(i+1)}) + d(v_{\sigma(n)}, v_{\sigma(1)}) \tag{2.3}$$

L'espace de recherche est l'ensemble des combinaisons possibles des n villes, soit au total $n!$ combinaisons. Ce problème, NP-difficile (Garey and Johnson, 1979), peut être aussi considéré comme la recherche d'un circuit hamiltonien de longueur minimale dans un graphe complet pouvant être anti-symétrique dans le cas général ($\exists(i,j)$ tel que $d(v_i, v_j) \neq d(v_j, v_i)$).

Inspiration biologique

Comment résoudre ce type de problème en s'inspirant des fourmis ? Tout d'abord, les fourmis ont-elles ce type de problème à résoudre dans la nature ? Pour ce qui est du PVC, qui consiste à minimiser un effort en terme de distance à parcourir, les fourmis peuvent rencontrer ce type de problème dans le cas où elles se déplacent entre une source de nourriture et leur nid. Un certain nombre de travaux ont montré que les fourmis réelles étaient confrontées à ce type de problème, mais qui plus est, elles sont capables d'y apporter une réponse (un exposé de cette source d'inspiration a par exemple été proposé dans (Bonabeau et al., 1999)). Les développements informatiques que nous allons présenter sont issus de ces constatations.

Cette section, concernant le PVC va nous permettre d'introduire le principe de base, nous présenterons ensuite un certain nombre d'améliorations de ces principes. Enfin, nous donnerons quelques exemples d'application à d'autres types de problèmes combinatoires.

Le PVC est l'un des premiers problèmes à avoir suscité « l'utilisation » des fourmis pour résoudre des problèmes d'optimisation combinatoire. Les premiers travaux ont été menés au début des années 1990 par Alberto Colorni, Marco Dorigo et Vittorio Maniezzo (Colorni et al., 1991; Dorigo et al., 1991; Dorigo, 1992).

L'article « *Distributed Optimization by Ant Colonies* » (Colorni et al., 1991) est une des premières parutions mettant en œuvre le « Ant System » (AS par la suite). AS peut être présenté, d'une manière générale, comme un algorithme d'optimisation stochastique distribué. Plus exactement, il est défini comme une combinaison de calcul distribué, de récompense retardée et d'heuristique gloutonne (Dorigo et al., 1991). Ce système s'inspire du comportement des fourmis qui déposent des traces de phéromones sur leurs lieux de passage afin de faciliter leur recherche de nourriture.

Le point de départ de cette série d'algorithmes se base sur l'observation des fourmis qui construisent des chemins entre une source de nourriture et leur nid. Les fourmis sont capables de déposer sur le sol une certaine quantité d'une substance chimique volatiles (les *phéromones*) qu'elles peuvent détecter ensuite. Les fourmis se déplacent au hasard, en cela, on peut dire qu'elles sont aveugles, mais sont attirées par les chemins de phéromones déposées par d'autres fourmis. Ainsi, plus les fourmis empruntent un chemin, plus il y aura de fourmis attirées par cet itinéraire. Le comportement collectif qui émerge est ainsi décrit comme un comportement autocatalytique (allelomimesis).

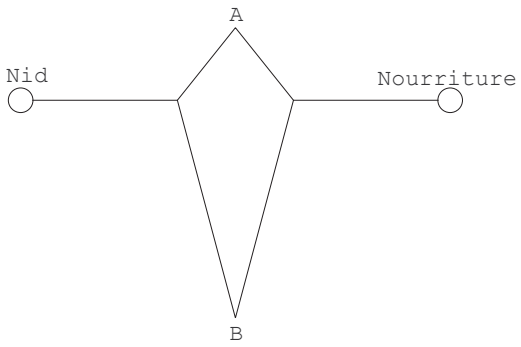


FIG. 2.2 – Contournement d'un obstacle par une colonie de fourmis.

Dans (Colorni et al., 1991; Dorigo and Gambardella, 1997b), la recherche du plus court chemin par une colonie de fourmis est illustrée par l'apparition d'un obstacle sur un chemin entre la source de nourriture et le nid. Sur la figure 2.2, la présence d'un obstacle sur le chemin contraint les fourmis à en faire le tour par l'un des deux chemins (A ou B). Quand les fourmis commencent à arriver par la gauche du dessin, en moyenne, la moitié des fourmis choisissent le plus long chemin (A) et l'autre moitié le plus court (B). Les fourmis déposant toutes des phéromones, le chemin B sera plus marqué que A pour un temps donné. Comme les fourmis suivent en probabilité le chemin le plus marqué, le phénomène s'amplifie et le chemin B devient majoritairement suivi par les ouvrières. Des expérimentations sur des fourmis d'Argentine ont été effectuées dans (Goss et al., 1989) où les chemins sont modifiés après une durée donnée de simulation, permettant ainsi de tester les capacités dynamiques de réponse du système. Enfin, l'aspect probabiliste du déplacement des fourmis assure qu'elles seront toujours à la recherche d'une meilleure solution puisque que même quand les fourmis choisissent majoritairement le chemin B, la probabilité de choisir A ne devient pas nulle. De plus, les phéromones étant des substances chimiques volatiles , elles s'évaporent avec le temps, ce qui permet aux fourmis de continuer l'exploration de leur environnement.

2.3.2 Principe général : l'algorithme Ant System

Du côté des fourmis artificielles, quelques modifications sont apportées aux capacités des fourmis décrites précédemment :

- elles possèdent une mémoire ;
- elles ne sont pas totalement aveugles ;
- le temps est discret.

Dans (Colorni et al., 1991) sont introduits trois algorithmes qui mettent à profit ce comportement collectif. Ils sont appliqués au PVC. De ces trois algorithmes, on retiendra celui qui a donné naissance à l'algorithme AS (Dorigo et al., 1996).

Voici la modélisation du comportement des fourmis qui est proposée. Les fourmis sont placées sur les sommets du graphe (i.e. sur chaque ville). Elles se déplacent d'un sommet à l'autre en empruntant les arêtes du graphe. On note par $b_i(t)$ le nombre de

fourmis dans la ville i à l'instant t et soit $m = \sum_{i=1}^{n} b_i(t)$ le nombre total de fourmis. Chaque agent-fourmi possède les caractéristiques suivantes :

- la fourmi dépose une trace de phéromones sur l'arête (i, j) quand elle se déplace de la ville i à la ville j ;
- elle choisit la ville de destination suivant une probabilité qui dépend de la distance entre cette ville et sa position et de la quantité de phéromones présente sur l'arête (règle de transition) ;
- afin de ne passer qu'une seule fois par chaque ville, la fourmi ne peut se rendre sur une ville qu'elle a déjà traversée, c'est pour cela que la fourmi doit être dotée d'une mémoire.

Pour éviter qu'une fourmi ne revienne sur ses pas, elle conserve la liste des villes qu'elle a déjà traversées. Cette liste, nommée liste-tabou est remise à zéro chaque fois que la fourmi a terminé un tour. La liste-tabou constitue la mémoire de la fourmi.

Les traces de phéromones sont modélisées par les variables $\tau_{ij}(t)$ qui donnent l'intensité de la trace sur le chemin (i, j) à l'instant t. La probabilité de transition du sommet i vers le sommet j par la fourmi k est donnée par :

$$p_{ij} = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\nu_{ij}]^{\beta}}{\sum_{l \notin L_k(i)} [\tau_{il}(t)]^{\alpha} \cdot [\nu_{il}]^{\beta}} & \text{si } j \notin L_k(i) \\ 0 & \text{sinon} \end{cases} \tag{2.4}$$

où $L_k(i)$ représente la liste-tabou de la fourmi k située sur le sommet i et ν_{ij} représente une mesure de visibilité qui correspond à l'inverse de la distance entre les villes i et j. α et β sont deux paramètres permettant de moduler l'importance relative des phéromones et de la visibilité.

La mise à jour des phéromones est effectuée une fois que toutes les fourmis sont passées par toutes les villes :

$$\tau_{ij} \leftarrow \tau_{ij}(1 - \rho) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2.5}$$

où ρ est un coefficient représentant l'évaporation des traces de phéromones. $\Delta\tau_{ij}^{k}$ représente le renforcement de l'arc (i, j) pour la fourmi k :

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L^k} & \text{si la fourmi } k \text{ est passée par l'arc } (i, j) \\ 0 & \text{sinon} \end{cases} \tag{2.6}$$

avec Q une constante et L^k la longueur du chemin parcouru par la fourmi k.

L'algorithme 2.1 donne la structure générale de AS pour le PVC (noté AS-TSP).

Algorithme 2.1: Algorithme AS-TSP
AS-TSP()

(1) Initialisation : $\tau_{ij} \leftarrow \tau_0 \forall (i,j) \in \{1,\ldots,n\}^2$, placer aléatoirement chaque fourmi sur une ville
(2) **pour** $t = 1$ à $t = t_{\max}$ **faire**
(3) **pour** chaque fourmi k **faire**
(4) Construire un chemin $T^k(t)$ avec la règle de transition 2.4
(5) Calculer la longueur $L^k(t)$ de ce chemin
(6) **finpour**
(7) Soient T^+ le meilleur chemin trouvé et L^+ la longueur correspondante
(8) Mettre à jour les traces de phéromones suivant la règle 2.5
(9) **finpour**
(10) **retourner** T^+ et L^+

La valeur initiale des τ_{ij} est τ_0. Concernant le nombre de fourmis, il est raisonnablement proposé d'utiliser autant de fourmis que de villes ($m = n$).

La suite de cette section présente un certain nombre d'extensions proposées autour de AS.

2.3.3 Ant System et l'apprentissage par renforcement

Dans (Gambardella and Dorigo, 1995), les auteurs généralisent AS à une famille plus large d'algorithmes : Ant-Q. AS peut en effet être interprété comme un cas particulier d'une méthode d'apprentissage par renforcement. Voici quelques notations introduites :

- la valeur $AQ(i,j)$ est introduite pour chaque arc du graphe des villes pour le PVC. Elle décrit l'intérêt de faire un déplacement en j quand la fourmi est en i,
- $HE(i,j)$ est une valeur heuristique associée à l'arc (i,j),
- à l'agent k, on associe la liste $J_k(i)$ représentant les villes qu'il reste à parcourir après la ville i.

La loi de transition d'états est la suivante : quand l'agent est en i, il choisit la ville j de la façon suivante :

$$j = \begin{cases} \arg\left[\max_{l \in J_k(i)} \left\{[AQ(i,l)]^\alpha \cdot [HE(i,l)]^\beta\right\}\right] & \text{si } q \leqq_0 \\ J & \text{sinon} \end{cases} \tag{2.7}$$

α et β sont des paramètres qui pondèrent l'importance relative de $AQ(i,j)$ et de $HE(i,j)$. q est une valeur choisie de manière uniformément aléatoire dans $[0,1]$. q_0 est un seuil ($q_0 \in [0,1]$) constituant aussi un paramètre de la méthode. J est une valeur aléatoire choisie suivant une distribution de probabilité dépendant de $AQ(i,J)$ et $HE(i,J)$ où $J \in J_k(i)$.

Les valeurs AQ sont mises à jour avec la loi suivante :

$$AQ(i,j) \leftarrow (1-\rho)AQ(i,j) + \rho(\Delta AQ(i,j) + \gamma \max_{l \in J_k(i)} \{AQ(i,l)\} \tag{2.8}$$

La fonction de distribution de probabilité pour J et ΔAQ sont deux paramètres structurels de Ant-Q. Trois règles de transition ont été testées :

- **pseudo-aléatoire** : J est choisie de manière uniformément aléatoire parmi les villes de $J_k(i)$. Cette règle ressemble à la règle de choix d'action pseudo-aléatoire du Q-learning ;
- **règle proportionnelle pseudo aléatoire** : J est déterminé suivant la distribution de probabilité suivante :

$$p_k(i,j) = \begin{cases} \frac{[AQ(i,j)]^\alpha \cdot [HE(i,j)]^\beta}{\sum_{l \in J_k(i)} [AQ(i,u)]^\alpha \cdot [HE(i,l)]^\beta} & \text{si } j \in J_k(i) \\ 0 & \text{sinon} \end{cases} \tag{2.9}$$

- **règle proportionnelle aléatoire** : Même formule que 2.9 avec en plus $q_0 = 0$, ce qui est alors équivalent à l'algorithme AS.

Les tests réalisés donnent la deuxième règle comme plus performante.

Deux types de renforcement différés ont été étudiés :

- **meilleur-globalement** :

$$\Delta AQ(i,j) = \begin{cases} \frac{W}{L^{k_{gb}}} & \text{si } (i,j) \in \text{tour effectué par } k \\ 0 & \text{sinon} \end{cases} \tag{2.10}$$

  W est un paramètre fixé à 10, k_{gb} est l'agent qui a fait le meilleur tour depuis le début de l'essai (*global-best*) et $L^{k_{gb}}$ est la longueur de ce tour. Cette méthode ne renforce que les arcs appartenant au meilleur tour trouvé jusqu'alors ;
- **meilleur-itération** :

$$\Delta AQ(i,j) = \begin{cases} \frac{W}{L^{k_{ib}}} & \text{si } (i,j) \in \text{tour effectué par } k \\ 0 & \text{sinon} \end{cases} \tag{2.11}$$

  où k_{ib} est l'agent qui a fait le meilleur tour à l'itération courante (*iteration-best*).

Les tests réalisés donnent des performances équivalentes pour les deux types de renforcement. On peut tout de même noter un léger avantage pour **meilleur-itération** en ce qui concerne le temps de découverte et son indépendance par rapport au paramètre γ. AS, par contre, utilise tous les agents pour renforcer $AQ(i,j)$ et tous les arcs sont renforcés. Les tests donnent Ant-Q plus performant que AS.

Des tests ont été menés pour montrer que tous les agents ne convergent pas vers un chemin unique et que l'exploration continue.

Le facteur « γ-branching » est introduit :

$$\delta_i = \max_{l \neq i} \{AQ(i,l)\} - \min_{l \neq i} \{AQ(i,l)\} \tag{2.12}$$

Le facteur γ-branching du noeud i est donné par le nombre d'arcs qui partent de i tel que la valeur AQ associée soit supérieure à $\lambda\delta_i + \min_{l \neq i} \{AQ(i,l)\}$ Le facteur γ-branching moyen donne une indication de la dimension de l'espace de recherche. L'expérimentation montre que pour différentes valeurs de λ, le facteur moyen diminue au cours du temps.

Une autre série de tests montre que la fonction HE est surtout utile au début de la recherche, c'est-à-dire quand les valeurs AQ n'ont pas encore été apprises.

Enfin, ANT-Q est comparé à des méthodes heuristiques : Elastic-Net, Simulated Annealing, Self Organizing Map et Farthest Insertion, ainsi qu'à des versions améliorées (avec une recherche locale : 2-opt, 3-opt). ANT-Q est l'algorithme le plus performant. Il faut noter que quand le 2-opt et le 3-opt sont appliqués aux résultats obtenus par ANT-Q, le résultat n'est pas amélioré.

La complexité de ANT-Q, de l'odre de mn^2, rend impossible son utilisation pour des problèmes de grande taille. L'avantage de ANT-Q réside dans sa falculté à résoudre le PVC anti-symétrique sans augmenter sa complexité.

2.3.4 $\mathcal{MAX} - \mathcal{MIN}$ Ant System

Dans (Stützle and Hoos, 1997b) est introduit l'utilisation de valeurs $\tau_{\max}$ et $\tau_{\min}$ pour l'intensité de la trace de phéromone. Ceci permet d'éviter que certains chemins soient trop favorisés. De plus, si la recherche ne progresse plus, un mécanisme de « *smoothing* » permet de remettre à flot certains arcs. On peut aussi retenir l'utilisation de listes candidates pour accélérer la recherche et l'utilisation d'une recherche locale pour améliorer les solutions. Cette recherche locale est appliquée pour toutes les fourmis ou pour la meilleure de l'itération seulement. Retenons les points principaux de la méthode, notée MMAS-TSP :

- la mise à jour des phéromones est globale ;
- les phéromones sont bornées : $\tau_{ij} \in [\tau_{min}, \tau_{max}]$;
- elles sont initialisées à τ_{max}.

2.3.5 $\mathbf{AS}_{rank}$

Dans (Bullnheimer et al., 1997b) on trouve une version élitiste de AS :

- les fourmis sont rangées par ordre décroissant des L^k,
- la mise à jour des phéromones tient compte du rang des σ meilleures fourmis :

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \frac{\sigma}{L^+} + \sum_{k=1}^{\sigma-1} \Delta\tau_{ij}^k \tag{2.13}$$

$$\Delta\tau_{ij}^k = \begin{cases} \frac{(\sigma-k)}{L^k} & \text{si } (i,j) \in T^k \\ 0 & \text{sinon} \end{cases} \tag{2.14}$$

2.3.6 Ant Colony System

ACS (*Ant Colony System*) est issu des variantes proposées pour AS (Dorigo and Gambardella, 1997b) :

- la règle de transition entre les villes est la suivante : la fourmi placée en i choisit la ville j telle que :

$$j = \begin{cases} \arg\max_{l \in J_k(i)} \left\{ [\tau_{il}(t)] \cdot [\nu_{il}]^\beta \right\} & \text{si } q \leq q_0 \\ J & \text{sinon} \end{cases} \tag{2.15}$$

q est un réel aléatoirement tiré dans $[0, 1]$, q_0 est un paramètre ($q_0 \in [0, 1]$) et J est une ville choisie aléatoirement suivant la probabilité :

$$p_{iJ}^k(t) = \frac{[\tau_{iJ}(t)] \cdot [\nu_{iJ}]^\beta}{\sum_{l \in J_k(i)} [\tau_{il}(t)] \cdot [\nu_{il}]^\beta} \tag{2.16}$$

où β est un paramètre servant à moduler la prise en compte des phéromones par rapport à la visibilité ;

- des listes de villes candidates[4] sont utilisées pour accélerer le processus de construction d'un chemin ;
- une heuristique locale est utilisée pour améliorer les solutions générées par les fourmis (2-opt ou 3-opt) ;
- la mise à jour des phéromones n'est faite qu'à partir du meilleur chemin généré ;
- une règle de mise à jour locale des phéromones est utilisée à chaque transition d'une fourmi.

Les résultats obtenus par ACS sur le PVC sont les meilleurs obtenus par les heuristiques à base de fourmis sans toutefois dépasser les meilleures heuristiques dédiées à ce problème.

2.3.7 L'heuristique ACO

Toutes les variantes que nous venons d'exposer ont été récemment regroupées sous une description plus large : l'heuristique ACO (*Ant Colony Optimization*), afin de faciliter le rapprochement des méthodes entre elles et de se soustraire aux spécificités du PVC (Stützle and Dorigo, 1999b; Dorigo and Di Caro, 1999b; Dorigo and Di Caro, 1999a; Dorigo et al., 1999). Dans cet effort de généralisation, on peut noter l'introduction (hasardeuse) d'un processus « reine » visant à coordonner et superviser le travail des fourmis (Taillard, 1999).

2.3.8 L'assignement Quadratique

Définition 2.2 *Le problème de l'assignement quadratique (PAQ) (Quadratic Assignment Problem : QAP). Soit deux matrices $n \times n$, $A = (a_{ij})$ et $B = (b_{ij})$ on cherche la permutation σ minimisant la quantité :*

$$C(\sigma) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\sigma(i)\sigma(j)} \tag{2.17}$$

Ce problème peut s'illustrer de la façon suivante : on désire placer n unités de production sur n emplacements dont on connait les distances (matrice A), on connait aussi les flux entre chaque unité de production (matrice B). L'objectif est de minimiser le coût total des transferts entre unités de production ($C(\sigma)$).

Un certain nombre de systèmes à base de fourmis artificielles ont été proposés pour résoudre ce problème NP-difficile :

[4] Si la taille des listes candidates est cl, pour une ville i, la liste est composée des cl villes les plus proche de i.

– AS-QAP (Maniezzo et al., 1994). Les phéromones τ_{ij} représentent l'attirance d'une unité de production i pour un emplacement j : $\sigma(i) = j$. L'heuristique correspondant à la visibilité pour le PVC est calculée de la façon suivante :

$$\nu_{ij} = \sum_{k=1}^{n} a_{ik} b_{jk} \tag{2.18}$$

Mis à part ces particularités, AS-QAP se comporte comme AS-TSP, c'est-à-dire de façon constructive : chaque fourmi construit de façon itérative une permutation des n unités de production (formule 2.4). et les phéromones sont mises à jour en utilisant le coût ($C(\sigma)$) des permutations construites (formule 2.5). Les résultats obtenus par AS-QAP sont similaires à ceux obtenus par d'autres méthodes (algorithmes génétique, recuit simulé, recherche tabou) sans toutefois se démarquer véritablement ;

– HAS-QAP (Gambardella et al., 1997; Gambardella et al., 1999b). Cette méthode est assez différente des autres approches à base de fourmis : les phéromones ne servent plus à construire une permutation mais à en modifier une existante. À l'initialisation, chaque fourmi k correspond à une permutation σ_k. Chaque déplacement d'une fourmi correspond à R modifications de sa permutation (R est un paramètre) : deux index i et j sont choisis dans $\{1, \ldots, n\}$ puis $\sigma_k(i)$ et $\sigma_k(j)$ sont échangés. Le choix de i est uniformément aléatoire, par contre le choix de j fait intervenir les phéromones : avec une probabilité q_0 (comme pour ACS), j est choisi de façon à maximiser la quantité $\tau_{i\sigma_k(j)} + \tau_{j\sigma_k(i)}$, sinon (probabilité $(1 - q_0)$) j est choisi avec la probabilité :

$$p_{ij}^{k} = \frac{\tau_{i\sigma_k(j)} + \tau_{j\sigma_k(i)}}{\sum_{l=1, l\neq i}^{n} \tau_{i\sigma_k(l)} + \tau_{l\sigma_k(i)}} \tag{2.19}$$

Chaque permutation ainsi obtenue est transformée par une procédure de recherche locale qui examine toutes les permutations possibles de deux éléments de σ_k dans un ordre aléatoire et qui retient à chaque fois celle qui améliore $C(\sigma_k)$. La mise à jour des phéromones suit la même méthode que ACS. Enfin, des procédures d'intensification et de diversication sont utilisées pour augmenter la recherche dans le voisinage de la meilleure permutation trouvée et éviter une stagnation de la recherche. Du point de vue des tests réalisés, HAS-QAP donne toujours la meilleure solution pour les problèmes réels par rapport aux meilleures heuristiques connues alors que cette tendance est inversée pour les problèmes artificiels (les matrices A et B suivent les mêmes lois de distribution) ;

– d'autres travaux ont donné des résultats satisfaisants pour le PAQ : MMAS-QAP (Stützle and Hoos, 1997c) basé sur AS-QAP et similaire à MMAS-TSP (section 2.3.4), ANTS-QAP[5] (Maniezzo, 1998), aussi basé sur AS-QAP, utilise des bornes inférieures sur le coût d'une solution partielle pour estimer l'intérêt d'une affectation. La mise à jour des phéromones dans ANTS-QAP ne fait pas apparaître d'évaporation. Enfin, l'algorithme FANT-QAP (Taillard, 1998) s'inspire de MMAS-QAP mais en n'utilisant qu'une seule fourmi et en mettant à

[5]ANTS est l'acronyme de *Approximate Nondeterministic Tree Search.*

jour les phéromones avec la solution courante et la meilleure rencontrée depuis la première itération ;
– Mis à part les différences sur l'utilisation des phéromones et de leur mise à jour, la plupart des méthodes se différencient aussi par l'opérateur de recherche locale utilisé, ce qui est souvent déterminant pour la qualité des résultats obtenus. Une recherche tabou a par exemple été utilisée dans (Roux et al., 1999) comme opérateur de recherche locale (ANTabu).

Un certain nombre de publications tentent récemment de comparer les différentes approches du PAQ par les fourmis (voir par exemple (Stützle and Dorigo, 1999a; Stützle and Hoos, 2000)).

2.3.9 Détection de graphes hamiltoniens

Wagner et Bruckstein proposent de traiter le problème de la détermination de circuits hamitoniens dans un graphe (Wagner and Bruckstein, 1999). L'objectif est de déterminer s'il existe un circuit passant par tous les sommets du graphe. Le principe est le suivant : pour chaque sommet on stocke deux valeurs entières μ et τ correspondant respectivement au nombre de passages de la fourmi sur ce sommet et au numéro de l'itération correspondant à ce passage. Chaque fourmi se déplace d'un sommet à l'autre en suivant les règles suivantes (VAW : *Vertex Ant Walk*) :

1. choisir le sommet suivant v, voisin du sommet où se trouve la fourmi (noté u), qui a les plus petites valeurs de μ et τ[6] ;
2. mettre à jour les valeurs μ et τ de u :
   – $\mu(u) \leftarrow \mu(u) + 1$ (nombre de passages),
   – $\tau(u) \leftarrow t$ (temps).
3. augmenter le temps : $t \leftarrow t + 1$;
4. aller en v.

Ce type de règle assure, comme le démontrent les auteurs, que si un cycle est trouvé, il sera répété indéfiniment. Cela n'assure pas que la méthode converge ni qu'elle converge uniquement sur des circuits hamiltoniens. Les expérimentations sont menées sur des graphes de 100 à 300 sommets de connexités diverses et donnent des résultats encourageants. Il semble qu'une seule fourmi soit utilisée. Le choix du sommet suivant est déterministe et les auteurs proposent d'utiliser une règle probabiliste en fonction des valeurs de μ sans donner de résultats. Des résultats plus détaillés sont donnés dans (Wagner et al., 2000).

2.3.10 Routage dans les réseaux non commutés

Les travaux de Di Caro et Dorigo (Di Caro and Dorigo, 1997; Di Caro and Dorigo, 1998b; Di Caro and Dorigo, 1998a) présentent l'utilisation d'agents-fourmis pour le routage de données dans un réseau. Il s'agit d'optimiser la quantité (*throughput*)

[6] a priori, on recherche d'abord le plus petit μ des sommets voisins puis parmi ces sommets, on prend celui qui a le plus petit τ.

et la qualité (*average packet delay*) des informations transmises sur un réseau de type internet.

L'algorithme présenté est adaptatif dans le sens où il dépend du traffic : les informations utilisées pour établir le routage sont réactualisées constamment. Il se distingue des algorithmes statiques où des tables de routage sont fixées une fois pour toutes, le trajet d'un paquet dans ce cas ne dépend que de son nœud source et de son nœud destination.

Le réseau peut être modélisé par un graphe de N nœuds dont les arcs sont pondérés par une largeur de bande (*bandwidth*) en bit/s et un délai de transmission. Chaque nœud possède un tampon d'entrée. Quand un paquet arrive sur un nœud, il est dirigé sur un nœud voisin en fonction de son nœud de destination et de la table de routage du nœud sur lequel il se trouve.

Le routage dynamique est un problème intrinsèquement distribué. Les décisions de routage ne peuvent se faire que d'un point de vue local et en approximant l'état courant et futur du réseau.

L'algorithme AntNet se compose de deux groupes d'agents : appelés « Forward ant » (F) et « Backward ant » (B) :

- notons par $F_{s \to d}$ la fourmi qui se déplace du nœud s au nœud d. Cette fourmi accompagne le paquet de données. Sur le chemin, chaque nœud k visité ainsi que le temps pour y parvenir sont mémorisés par la fourmi dans la structure $S_{s \to d}(k)$. Chaque agent choisi aléatoirement son prochain nœud proportionnellement à l'intérêt des nœuds voisins relativement au nœud de destination, ou alors, selon une faible probabilité, la fourmi choisit aléatoirement le nœud suivant de façon équiproblable, ceci assurant une certaine exploration. Si le nœud choisi a déjà été visité, la fourmi choisit uniformément un autre nœud ;
- quand le nœud de destination est atteint, l'agent $F_{s \to d}$ génère un agent $B_{d \to s}$ en lui transmettant toute sa mémoire. La fourmi $B_{d \to s}$ retourne donc au nœud de départ d. A chaque nœud k traversé, $B_{d \to s}$ met à jour les structures de données :
  1. La table de routage : T contenant P_{in} la probabilité de choisir le nœud n comme nœud suivant quand i est le nœud de destination,
  2. Une liste $Trip_k(\mu_i, \sigma_i^2)$ des valeurs moyennes estimées et des variances associées du temps de voyage du nœud courant k vers tous les nœuds du réseau.

  Les agents F sont de même priorité que les paquets de données alors que les agents B sont de priorité supérieure afin de propager rapidement l'état du réseau. Les probabilités de transitions sont renforcées dans la direction d'arrivée de B en fonction des valeurs mémorisées dans T_k et $Trip_k(\mu_i, \sigma_i^2)$.

L'expérimentation a été faite en comparant AntNet avec des algorithmes classiques du problème de routage :

- OSPF (*official Internet routing algorithm*) : les tables de routage sont construites à partir du calcul des plus courts chemins ;
- BF (*asynchronous distributed Bellman-Ford algorithm*) ;
- SPF : *link-state algorithm with dynamic metric for link-costs evaluations* ;
- SPF_1F : *SPF with flooding limited to the first neighboors*. Le coût des nœuds

éloignés sont tous mis à la même valeur ;
– Daemon : algorithme idéal. À chaque instant on connaît la charge des files d'attente de tous les nœuds du réseau. On calcule pour chaque lien un coût qui dépend de la capacité du lien, de la quantité de données à transmettre sur ce lien et de la quantité moyenne de données en attente sur ce lien.

AntNet est le plus performant ou de même niveau de performances parmi les algorithmes testés. D'une manière générale AntNet est robuste. Ses performances sont principalement affectées par la fréquence de lancement des agents et par leur repartition spatiale. En effet, à partir d'un certain taux de lancement, les tables de routage ont tendance à vibrer ou à converger trop rapidement vers une configuration qui dégrade les performances. Pour la répartition spatiale, les performances de AntNet sont aussi dégradées si on fait apparaître un goulet d'étranglement dans le réseau. Enfin, on peut remarquer que le temps de réaction de l'algorithme correspond au temps de retour de la fourmi B après le passage de la fourmi F. Si la mise à jour de la fourmi B ne correspond plus à la situation du réseau, cela n'affecte que le choix pour le nœud destination. Ceci rend le système robuste face à de courtes fluctuations du traffic.

Le même type de problème est abordé dans (Bonabeau et al., 1998a; Heusse et al., 1998). Un problème similaire de routage entre des satellites de télécommunication a été proposé (Sigel et al., 2000). Les fourmis n'accompagnent cependant pas chaque paquet mais sont lancées régulièrement à travers le réseau. La fréquence de mise à jour des tables de routage ne dépend pas dans ce cas du traffic réel. Enfin, des travaux mettant en œuvre des algorithmes à base de fourmis artificielles ont aussi été menés sur le problème du routage dans les réseaux commutés (Schoonderwoerd et al., 1997).

2.3.11 Hybridation avec un algorithme génétique

Dans (Abbatista et al., 1995; Abbattista and Dalbis, 1996), AS est utilisé pour accélérer le processus de recherche d'un l'Algorithme Génétique (AG) [7]. Les tests sont effectués sur le PVC à titre de comparaison. Il s'agit de combiner l'aspect coopératif de AS avec l'aspect évolutif de l'AG. Deux méthodes d'hybridation sont possibles :

1. chaque individu de l'AG correspond à un jeu de paramètres de AS. L'évaluation d'un individu de l'AG est effectuée en exécutant AS paramétré par cet individu ;
2. la deuxième méthode intègre AS à l'AG en tant qu'heuristique de recherche locale : à chaque génération, une certaine proportion de solutions trouvées par AS est intégrée à la population de solutions manipulées par l'AG[8].

Les tests réalisés sont cependant assez pauvres puisqu'une seule instance du PVC à 50 villes est utilisée.

La recherche du meilleur jeux de paramètres pour ACS-TSP a été proposée de façon beaucoup plus précise dans (Botee and Bonabeau, 1999) ainsi que pour les problèmes de routage (White et al., 1998). D'une façon assez proche, on pourrait disposer de plusieurs colonies de fourmis (Kawamura et al., 1998) en compétition entre elles.

[7] voir les chapitres 5 et 6 pour une présentation des AG.
[8] l'AG est implémenté de façon classique pour le PVC.

2.3.12 Parallélisation de ACO

Plusieurs propositions ont été faites pour paralléliser les algorithmes ACO. Par exemple dans (Roux et al., 1999; Talbi et al., 1999), chaque fourmi est affectée à un processeur. Un processus maître est utilisé pour initialiser et synchroniser les fourmis, stocker la matrice des phéromones et la meilleure solution trouvée. Chaque processeur esclave prend en charge le travail d'une fourmi, à savoir la construction d'une solution et la recherche tabou pour cette solution. Le maître reçoit les solutions de toutes les fourmis et met à jour les phéromones en conséquence.

Dans (Bullnheimer et al., 1997c), deux versions parallèles de AS-TSP sont évaluées. La première implémentation (synchrone) répartit les fourmis sur des processeurs différents et chaque processeur contruit ainsi un chemin pour le PVC. Ce modèle induit un volume de communications important ce qui nuit à l'intérêt de la parallélisation. La seconde méthode (partiellement asynchrone) laisse chaque fourmi effectuer plusieurs itérations avant que les résultats ne soient renvoyés au processus maître. Malheureusement, les résultats présentés sont obtenus par simulation.

Dans (Stützle, 1998b), le lancement de plusieurs instances indépendantes de MMAS-TSP est d'abord proposé. Du point de vue de l'effort de parallélisation, c'est évidemment le plus simple. Puis, faisant remarquer que pour la plupart des problèmes, la recherche locale est la plus coûteuse en temps de calcul, l'auteur propose trois niveaux de distribution : le premier correspond au maître dans les précédents travaux cités, le deuxième correspond aux esclaves-fourmis qui construisent un chemin dans le cas du PVC et le troisième niveau correspond à la recherche locale. Chaque niveau communique avec plusieurs processus de niveau inférieur. L'apport principal est donc de paralléliser l'étape de recherche locale pour chaque processus-fourmi.

2.3.13 Récapitulatif des problèmes combinatoires traités avec des fourmis

Le tableau 2.1 présente pour chaque problème combinatoire les références correspondant à des travaux inspirés des fourmis. Bien que ce tableau soit par définition dépassé, nous avons tenté d'y faire figurer le maximum de références bibliographiques. De plus il donne un aperçu de la vitalité de ce domaine. Les problèmes sont approximativement classés par ordre d'apparition.

Problème	Références
Travelling Salesman	(Colorni et al., 1991; Dorigo et al., 1991; Dorigo, 1992; Colorni et al., 1992; Colorni et al., 1995; Gambardella and Dorigo, 1995; Dorigo and Gambardella, 1996; Dorigo et al., 1996; Gambardella and Dorigo, 1996; Bullnheimer et al., 1997b; Dorigo and Gambardella, 1997b; Dorigo and Gambardella, 1997a; Stützle and Hoos, 1997b; Stützle and Hoos, 1997c; Stützle and Hoos, 1997a; Bullnheimer et al., 1999c; Stützle and Dorigo, 1999b)
Quadratic Assignement	(Maniezzo et al., 1994; Gambardella and Dorigo, 1995; Gambardella et al., 1997; Taillard and Gambardella, 1997; Maniezzo, 1998; Maniezzo and Carbonaro, 1998; Taillard, 1998; Roux et al., 1999; Gambardella et al., 1999b; Maniezzo and Colorni, 1999; Stützle and Dorigo, 1999a)
Jobshop Scheduling	(Colorni et al., 1994)
Bin Packing	(Bilchev and Parmee, 1996b)
Vehicle Routing	(Bullnheimer et al., 1997a; Bullnheimer et al., 1999b; Bullnheimer et al., 1999a; Bullnheimer, 1999; Gambardella et al., 1999a)
Network Routing	(Di Caro and Dorigo, 1997; Schoonderwoerd et al., 1997; Di Caro and Dorigo, 1998b; Di Caro and Dorigo, 1998a; Di Caro and Dorigo, 1998; Di Caro and Dorigo, 1998; Bonabeau et al., 1998a; Heusse et al., 1998; White et al., 1998)
Bus Driver Scheduling	(Forsyth and Wren, 1997)
Sequential Ordering	(Gambardella and Dorigo, 1997)
Graph Coloring	(Costa and Hertz, 1997)
Frequency Assignement	(Maniezzo and Carbonaro, 1998; Maniezzo and Carbonaro, 2000)
Shortest common supersequence	(Michel and Middendorf, 1998; Michel and Middendorf, 1999)
Flowshop Scheduling	(Stützle, 1998a; Laügt et al., 2000; T'Kindt et al., 2000)
Multi-objectif	(Mariano and Morales, 1999)
Virtual Wave Length Path routing	(Navarro Varela and Sinclair, 1999)
Multiple Knapsack	(Leguizamón and Michalewicz, 1999)
Single Machine Total Tardiness	(Bauer et al., 1999)
Recognizing Hamiltonian Graphs	(Wagner and Bruckstein, 1999)
Total Weighted Tardiness	(den Besten et al., 2000)
Constraint Satisfaction	(Pimon and Solnon, 2000; Solnon, 2000b; Solnon, 2000a)
Dynamic graph search	(Wagner et al., 2000)

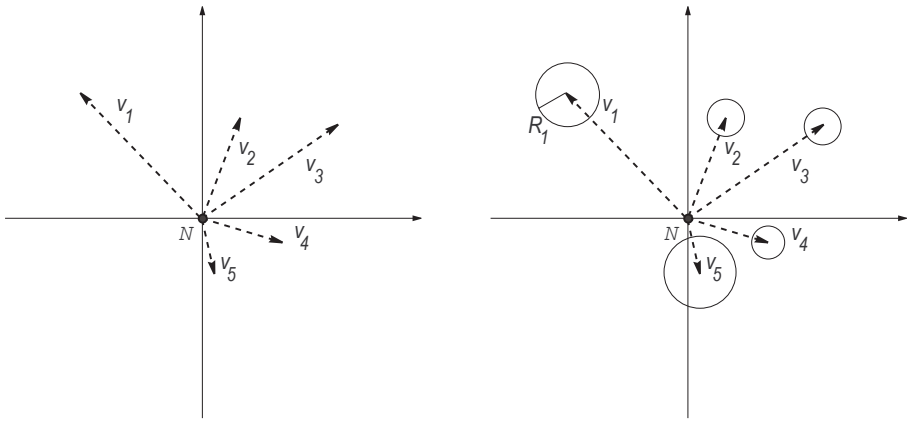
TAB. 2.1 – Problèmes combinatoires traités par des fourmis.

2.4 Optimisation numérique

Les articles présentés dans cette section placent les fourmis dans un espace à variables réelles. La difficulté de définir le mouvement des agents fourmis dans ce genre d'espace explique le nombre moins important de publications relatives à ce domaine d'étude.

Dans (Bilchev and Parmee, 1995; Bilchev and Parmee, 1996a), les auteurs décrivent un système de calcul dynamique qui permet de traiter des problèmes d'optimisation dans des espaces définis sur ce type d'espace de recherche. Il s'agit de minimiser une fonction f à n variables réelles. La métaphore de la colonie de fourmis proposée est appliquée à un problème de conception en ingénierie qui est fortement contraint.

La difficulté d'aborder des problèmes à variables réelles par une population d'agents fourmis provient du caractère non discret de la structure de description du problème. Le système proposé par les auteurs (appelé *Ant Colony Metaphor*, ACM par la suite) se compose d'un ensemble fini de vecteurs représentant des directions qui partent d'un point central représentant le nid. Ces vecteurs évoluent dans le temps suivant les performances des fourmis. La figure 2.3.a montre ces vecteurs représentant des directions intéressantes pour les fourmis à la sortie du nid.



(a) (b)

Fig. 2.3 – Optimisation numérique : ACM. (a) Directions de recherche à la sortie du nid. (b) Rayons de recherche.

Chaque fourmi choisit une direction pour sortir du nid (vecteur v_k). Une fois qu'elle a atteint la position indiquée par le vecteur v_k, elle tente d'améliorer l'optimum courant en générant un point de $\mathbb{R}^n$ dans le voisinage de v_k. La taille de ce voisinage est déterminée par la valeur R. La figure 2.3.b illustre graphiquement les rayons de recherche qui peuvent être différents pour chaque direction v_k.

L'algorithme 2.2 utilise la structure de données $A(t)$ qui représente le nid et son voisinage.

Avant que l'algorithme soit lancé, il faut placer le nid. Ceci peut être effectué par un algorithme génétique ou une méthode similaire. ACM est alors utilisé comme algo-

Algorithme 2.2: Optimisation numérique par une colonie de fourmis.
ACM()

(1) $t \leftarrow 0$
(2) Initialiser $A(t)$
(3) Évaluer $A(t)$
(4) Évaporer $A(t)$
(5) **tantque** la condition d'arrêt est fausse **faire**
(6) $\quad t \leftarrow t + 1$
(7) $\quad$ Déposer_phéromones $A(t)$
(8) $\quad$ Déplacer_fourmis $A(t)$
(9) $\quad$ Évaluer$A(t)$
(10) $\quad$ Évaporer $A(t)$
(11) **fintantque**
(12) **retourner** la meilleure solution trouvée

rithme de recherche locale fine. Il faut ensuite définir un rayon R qui détermine la taille du sous espace de recherche considéré à chaque cycle. Initialiser $A(t)$ envoie les fourmis dans différentes directions et à un rayon inférieur à R. Evaluer $A(t)$ est un appel à la fonction objectif pour toutes les fourmis. Déposer_phéromones $A(t)$ permet de déposer les phéromones sur les directions sélectionnées par les fourmis relativement à la performance de chaque direction. Déplacer_fourmis $A(t)$ envoie les fourmis en sélectionnant les directions aléatoirement et proportionnellement à la quantité de phéromones. La fourmi est alors placée sur la meilleure position trouvée par une fourmi ayant déjà empreinté cette direction. Evaporer $A(t)$ réduit le marquage des phéromones. Le pas local est déterminé par la formule suivante :

$$\Delta(t, R) = R \times (1 - r^{(1-t/T)^b}) \tag{2.20}$$

où r est nombre aléatoire dans $[0, 1]$, T est le nombre maximal d'itérations de l'algorithme et b est un paramètre fixant le degré de non-uniformité. La fonction $\Delta(t, R)$ renvoie ainsi un réel dans $[0, R]$ qui est en probabilité plus proche de 0 quand t se rapproche de T.

Pour améliorer le modèle, la fourmi peut créer une direction basée sur deux directions existantes. Ce mécanisme s'apparente au croisement pour les algorithmes génétiques. La description de ACM est cependant obscure sur ce dernier point, ce qui rend difficile son implémentation.

Wodrich (Wodrich, 1996) a proposé une méthode assez similaire. Il utilise cependant deux types de fourmis : des agents de recherche globaux sont introduits afin de remplacer l'algorithme génétique. De plus, il introduit une notion d'âge pour chaque direction ce qui lui permet d'éliminer celles qui ne permettent pas d'améliorer la meilleure solution et de différencier le rayon R pour chaque direction.

2.5 La classification

Le problème de la classification est tout à fait adapté à une résolution distribuée. Les fourmis ont en effet ce genre de problème à résoudre quand leur nid est dérangé et qu'elle doivent rassembler leurs oeufs en fonction de leur état de développement. On peut notamment trouver des travaux portant sur la classification non supervisée (Lumer and Faieta, 1994) ou sur le partitionnement de graphes (Kuntz et al., 1997). Ces travaux seront plus largement développés au chapitre 3, traitant de classification non supervisée.

2.6 La Vie artificielle

De nombreux travaux en vie artificielle utilisent les fourmis comme support. Certaines études s'intéressent à l'apparition de phénomènes complexes, similaires aux phénomènes collectifs naturels. L'évolution de programmes informatiques est souvent utilisée pour illustrer l'apparition de comportements adaptatifs. Par exemple, dans (Jefferson et al., 1992) le système Genesys manipule des programmes-fourmis sous la forme de réseaux de neurones artificiels ou d'automates à états finis. Les gènes de chaque organisme sont représentés sous la forme de chaînes binaires codant soit les poids d'un réseau de neurones soit la table de transition d'un automate fini. Ce système parvient en quelques centaines de générations à faire évoluer une population de 64000 individus (initialisés au hasard) vers la capacité à suivre un chemin incomplet, assimilable aux traces de phéromones. D'une façon similaire, des programmes Lisp et la programmation génétique ont été utilisés pour faire évoluer une colonie de fourmis évaluée sur la quantité de nourriture ramenée au nid (Koza, 1992).

La question de l'apparition de comportements complexes est étudiée dans (Collins and Jefferson, 1992) avec le projet AntFarm qui tente de montrer l'apparition des phéromones dans le comportement de fourragement de fourmis pilotées par un réseau de neurones artificiels. Les auteurs n'observent cependant pas que l'algorithme génétique utilisé fasse évoluer le système vers une coopération dans le fourragement par l'utilisation de phéromones. Ils attribuent cette « non apparition » à l'évaporation des phéromones qui rend inadaptés les comportements uniquement basés sur la densité en phéromones. De plus, le placement de la nourriture est statique ce qui peut rendre l'évolution du système vers la coopération inutile. Dans (Kawamura et al., 1999), la même expérience est menée : un réseau de neurones pilote le comportement de fourragement d'une colonie de fourmis et un algorithme génétique est utilisé pour faire évoluer une population de réseaux de neurones. Les auteurs concluent cependant à l'émergence de la communication par les phéromones. Cela est peut-être dû à la fonction d'évaluation de l'AG : pour chaque réseau de neurones correspondant à un individu de l'AG, son évaluation est obtenue en mettant en compétition une colonie de fourmi pilotée par ce réseau avec une autre colonie. La principale critique que l'on peut formuler à l'encontre de ce type d'expériences est que le réseau de neurones possède une architecture fixe et qui prévoit une sortie « utilisation de phéromones ». Ainsi, les résultats ne prouvent pas réellement que l'utilisation des phéromones émerge mais plus simplement

qu'elle confère à la colonie qui utilise cette sortie un avantage dans la compétition pour la recherche de nourriture. De toutes les façons, si on se tourne vers les vraies fourmis, on constate qu'il y a des espèces qui n'utilisent pas de phéromones pour le fouragement (*Pachycondyla apicalis* par exemple (Fresneau, 1994)). Cela signifie-t-il que ces espèces sont « arriérée » et que leur survie ne tient qu'à la probabilité de mutation de leur stratégie de fourragement ? Ou alors cela signifie que leur stratégie est suffisante pour leur régime alimentaire ? Ces questions appellent à ce que les systèmes artificiels soient peaufinés et expérimentés plus intensivement.

Citons pour finir le projet Manta (Drogoul et al., 1991) où l'émergence de la spécialisation est étudiée.

2.7 Domaines voisins

2.7.1 Les systèmes multi-agents

Les systèmes multi-agents (SMA) permettent de modéliser les systèmes complexes où plusieurs entités (les agents) communiquent entre elles et agissent sur leur environement. Ce type de modèle se prète bien à la modélisation d'une colonie de fourmis et permet de classifier les fourmis virtuelles (ou logicielles) par rapport à d'autres types d'agents.

Les systèmes multi-agents sont proches de ce que l'on appelle l'intelligence artificielle distribuée. Ce type d'architectures distribuées prend de plus en plus d'importance dans la résolution de problèmes complexes mais aussi dans la simulation de systèmes.

Du point de vue de la simulation, les SMA permettent de construire des modèles réduits afin d'expérimenter la pertinence de certains modèles, de conduire des tests irréalisables en vraie grandeur... Les applications sont nombreuses notamments dans les sciences sociales et économiques. Par exemple, les SMA peuvent être utilisés pour étudier les phénomènes sociaux humains tels que le commerce, la migration, la formation de groupes, le combat, l'interaction avec l'environnement, la transmission de la culture, la propagation des maladies ou encore la dynamique des populations (Epstein and Axtell, 1996).

Du point de vue de la résolution de problèmes, on peut expliquer la nécessité de distribuer les actions et l'intelligence sur différentes entités pour les raisons suivantes :

- le problème à traiter peut être physiquement distribué comme les problèmes de transport ou de déplacement de véhicules ;
- le problème est fonctionnellement distribué comme la construction d'une voiture où plusieurs experts sont nécessaires sans que personne ne soit capable de maitriser toute la fabrication ;
- le développement des réseaux induit que l'information est le plus souvent répartie, il faut donc mettre au point des protocoles de communication, d'interrogation et de navigation dans ces réseaux ;
- le problème à traiter est dynamique ce qui nécessite une architecture souple et adaptative.

Nous reprenons ici dans l'ouvrage de Ferber (Ferber, 1995) quelques définitions :

Définition 2.3 *On appelle agent une entité physique ou virtuelle*

1. *qui est capable d'agir dans un environnement ;*
2. *qui peut communiquer directement avec d'autres agents ;*
3. *qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser) ;*
4. *qui possède des ressources propres ;*
5. *qui est capable de percevoir (mais de manière limitée) son environnement ;*
6. *qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune) ;*
7. *qui possède des compétences et offre des services ;*
8. *qui peut éventuellement se reproduire ;*
9. *dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétence dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.*

À partir de cette définition, on peut définir un système multi-agent :

Définition 2.4 *On appelle système multi-agent (SMA), un système composé des éléments suivants :*

1. *un environnement E c'est-à-dire un espace disposant généralement d'une métrique ;*
2. *un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus créés, détruits et modifiés par les agents ;*
3. *un ensemble A d'agents, qui sont des objets particuliers ($A \in O$), lesquels représentent les entités actives du système ;*
4. *un ensemble de relations R qui unissent les objets (et donc les agents) entre eux ;*
5. *un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O ;*
6. *des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.*

Il y a principalement deux types d'agents :
- les agents cognitifs, qui sont assez perfectionnés, tant du point de vue de la communication que des règles de décision qu'ils mettent en action ;
- les agents réactifs, qui par opposition sont assez simples et peu « intelligents » pris individuellement.

Ces définitions s'appliquent bien à la modélisation d'une fourmi (un agent) ou d'une colonie entière (un SMA). Les fourmis, artificielles ou non, sont en général considérées comme des agents réactifs bien que la limite entre réactif et cognitif soit parfois difficile à déterminer. Il est rare en effet que l'on modélise trés finement les capacités cognitives individuelles des fourmis.

2.7.2 La vie artificielle

La problématique de la vie artificielle a été définie par LANGTON (Langton, 1989) :

> « Artificial Life is the study of man-made systems that exhibit behaviors characteristic of natural living systems. »

La vie artificielle étend le champ d'étude de la biologie à l'étude de la vie telle qu'elle pourrait être. Les principaux thèmes de recherche en vie artificielle sont :

- l'analyse de la dynamique des phénomènes complexes à l'aide d'automates cellulaires ou d'équations différencielles non linéaires ;
- l'évolution de populations par l'utilisation d'algorithmes évolutionnaires ;
- la réalisation de créatures artificielles (les animats) capables de survivre dans des univers nécessitant une certaine adaptation ;
- l'étude des phénomènes collectifs issus de l'interaction d'un ensemble d'agents réactifs.

Le deuxième thème, même s'il n'est pas directement lié aux fourmis, n'est pas sans relation puisqu'il s'agit de population, concept proche de la notion de colonie. C'est évidemment dans le dernier thème que les fourmis artificielles trouvent leur place. Ce dernier thème est maintenant décrit comme l'étude de l'intelligence collective.

2.7.3 L'intelligence collective

Les systèmes artificiels s'inspirent de la nature pour l'intelligence que l'on peut y percevoir. Il serait présomptueux de vouloir donner ici une définition de l'intelligence mais nous pouvons donner une définition correspondant aux préocupations des systèmes artificiels inspirés des sociétés naturelles. Ainsi, selon (Bonabeau and Theraulaz, 1994), « un agent est dit "intelligent" (ou "cognitif") si et seulement si dans ses interactions avec son environnement, il réalise des actions-perceptions bouclées de sorte à satisfaire une contrainte de type "viabilité" ». Ici, l'intelligence, c'est la survie.

En fait, l'intelligence, en tant que notion manipulée en intelligence artificielle distribuée, émerge des interactions que les agents entretiennent entre eux et avec leur environnement extérieur (Bonabeau and Theraulaz, 1994). C'est sur les interactions que l'intelligence collective est basée.

La notion d'émergence est souvent employée quand on parle d'intelligence collective. Bien que de nombreux travaux issus du domaine de la vie artificielle traitent cette notion, il est assez difficile d'en donner une définition précise. La notion d'émergence est cependant assez intuitive. On dit qu'un phénomène est émergent lorsqu'il était impossible d'en prévoir l'apparition avant que le système qui le produit ne soit actionné. Cette définition est en quelque sorte temporelle, on peut lui préférer une définition plus structurelle : si le comportement macroscopique d'un modèle artificiel est inattendu, c'est-à-dire ne pouvant être déduit de la spécification « microscopique » du modèle, alors on parle d'un comportement émergent. On peut définir plusieurs niveaux d'émergence en fonction de la difficulté de prévoir le résultat macroscopique (Assad and Packard, 1991) :

- pas d'émergence : le comportement est imédiatement déductible de l'inspection des spécifications ou règles le générant ;

- faiblement émergent : le comportement est déductible des spécifications du système après son observation ;
- ⋮
- fortement émergent : le comportement peut être déduit en théorie mais extrêmement difficile à élucider ;
- émergence maximale : le comportement ne peut être déduit des spécifications.

Jusqu'ici, on pourrait penser que l'intelligence collective se contente d'être le quatrième thème de recherche que nous avons cité pour la vie artificielle. C'est cependant un thème plus large puisque l'on peut y rapporter un certain nombre d'algorithmes d'optimisation, qui mettent bien en avant une notion d'intelligence collective puiqu'ils sont appliqués à des problèmes complexes mais qui ne partagent rien avec la notion de vie, qu'elle soit artificielle ou non.

2.8 Conclusion

2.8.1 Fallait-il parler de fourmis artificielles ?

Les travaux que nous avons présentés tout au long de ce chapitre sont essentiellement inspirés des fourmis. Pourquoi ne parle-t-on pas de termites artificiels ? La plupart des capacités qui ont été exploitées ne sont pas particulières aux fourmis. Par exemple, les phéromones ne sont pas une exclusivité des fourmis. La principale explication que l'on peut fournir est l'étonnante diversité de comportements efficaces. De plus, les fourmis jouissent d'une sympathie et d'une fascination généralement plus forte que les autres insectes sociaux, peut-être parce qu'il y a peu d'espèces nuisibles.

2.8.2 L'avenir ?

Du point de vue scientifique, les fourmis artificielles vont très certainement se développer :

- pour l'instant, les travaux théoriques sont rares. On peut tout de même trouver quelques éléments de convergence de AS dans (Gutjahr, 2000) : sous certaines hypothèses, AS peut être modélisé comme un processus markovien, ce qui permet de montrer que pour un nombre de fourmis suffisament grand, la probabilité de trouver l'optimum est aussi proche de 1 que l'on désire ;
- la source d'inspiration n'est pas tarie, de nombreux comportements collectifs n'ont pas encore été exploités ;
- l'hybridation d'algorithmes à base de fourmis avec d'autres méthodes, généralistes ou dédiées à un domaine, est une source de travaux pratiquement inépuisable.

Du point de vue industriel, par contre, il n'y a actuellement que très peu d'applications des fourmis artificielles. À notre connaissance, aucune colonie de robots-fourmis n'a été envoyée sur une autre planète du système solaire. Les applications en optimisation sont, nous semble-t-il, les plus aptes à rentrer dans des systèmes opérationels.

Chapitre 3

Le problème de la classification non supervisée

Dans ce chapitre nous présentons le problème de la classification non supervisée que nous allons traiter à l'aide des fourmis artificielles dans le chapitre suivant. Après avoir défini le problème nous présenterons succintement quelques approches existantes pour le résoudre. Nous porterons ensuite notre attention sur les travaux déjà réalisés dans ce domaine et qui s'inspirent du comportement des fourmis réelles.

3.1 Introduction

La classification est un problème central en reconnaissance des formes. Les travaux sont abondants, par exemple pour reconnaissance de l'écriture manuscrite, la reconnaissance de la parole ou encore l'interprétation de photos aériennes ou médicales. Pour chacune de ces problématiques, il s'agit de détecter certaines caractéristiques pour en extraire des informations exploitables par la suite. Par exemple, pour la reconnaissance de l'écriture manuscrite, pour un mot donné il s'agit de classer les formes de chacune des lettres le constituant dans une des 26 classes formant l'alphabet latin. Une fois cette phase de classification effectuée, pour chacune des formes composant le mot on dispose de la lettre correspondante.

D'une façon très générale la reconnaissance des formes peut se diviser en trois phases :

1. acquisition des données,
2. pré-traitement des données,
3. classification des données.

La première phase consiste à transformer les informations à traiter en signaux numériques manipulables par un ordinateur. Pour la reconnaissance d'images satellites, cette phase correspond à l'utilisation d'une caméra permettant de transformer les variables

physiques (l'intensité lumineuse) en données numériques (le niveau de gris). Le prétraitement des données est une phase transitoire visant à mettre en forme les données pour la phase de classification. Il s'agirait par exemple de binariser les images satellites citées en exemple. La phase de classification correspond à l'étape de décision. Pour des images satellites il peut s'agir de décider si la vue satellite contient ou non des bâtiments. Il y a principalement deux approches : l'approche statistique (estimation de densités de probabilité) et l'approche géométrique (analyse structurelle ou syntaxique). Par la suite nous ne considérons que cette dernière phase de classification en supposant que les phases précédentes ont permis d'obtenir un ensemble de données numériques que nous formalisons de la façon suivante.

Nous supposons qu'un ensemble $O = \{o_1, \ldots, o_N\}$ de N données ou objets ont été collectées par un expert du domaine. Chaque objet o_i correspond à un vecteur $\mathbf{x}_i$ de M valeurs numériques $(x_{i1}, \ldots, x_{iM})$ qui correspondent aux M attributs numériques $a_1, \ldots, a_M$. Par la suite nous utilisons indifféremment o_i ou $\mathbf{x}_i$. Le tableau 3.1 donne un exemple de données possibles.

Objets	Attributs		
	a_1	$\ldots$	a_M
o_1	1.32	$\ldots$	3.67
$\vdots$	$\vdots$	$\vdots$	$\vdots$
o_N	8.98	$\ldots$	2.75

TAB. 3.1 – Exemple de données numériques.

3.2 Le problème de la classification

3.2.1 Différentes classifications possibles

Classifier est le processus qui permet de rassembler des objets dans des sous-ensembles tout en conservant un sens dans le contexte d'un problème particulier. Les sous-ensembles obtenus représentent une organisation, ou encore une représentation de l'ensemble des objets. Les relations disponibles entre les objets sont rassemblées dans une *matrice de dissimilarité* dans laquelle les lignes et les colonnes correspondent aux objets. Comme les objets peuvent être représentés par des points dans un espace numérique à M dimensions, la dissimilarité entre deux objets peut être définie comme une distance entre les deux points correspondants. Cette matrice de dissimilarité, que nous noterons D par la suite, est la principale entrée nécessaire à la phase de classification. La figure 3.1 présente les différentes variantes que l'on peut trouver parmi les méthodes de classification (Jain and Dubes, 1988). Voici une rapide description des ces méthodes :

– classification exclusive/non exclusive. Une *classification exclusive* est un partitionnement des objets : un objet n'appartient qu'à une classe et une seule. Au

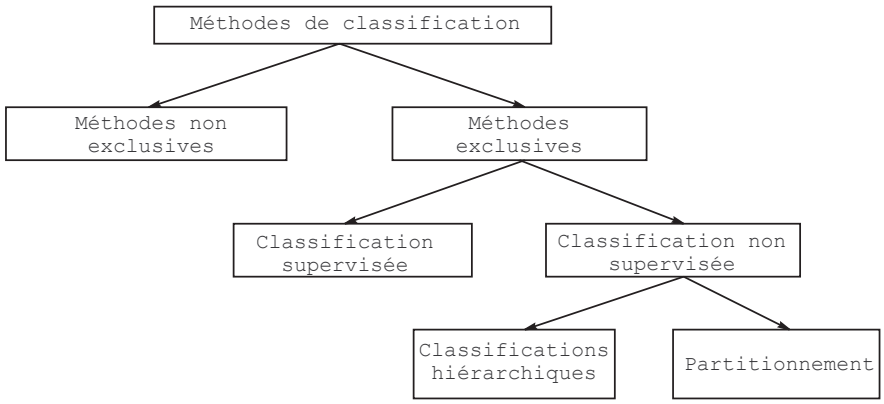


FIG. 3.1 – Les méthodes de classification (d'après (Jain and Dubes, 1988)).

contraire, une classification non exclusive autorise qu'un objet appartienne à plusieurs classes simultanément. Les classes peuvent alors se recouvrir ;

– classification supervisée/non supervisée. Une méthode de *classification non supervisée* (en anglais on parle de *clustering* ou de *unsupervised learning*) n'utilise que la matrice de dissimilarité D. Aucune information sur la classe d'un objet n'est fournie à la méthode (on dit que les objets ne sont pas étiquetés). En classification supervisée, les objets sont étiquetés tout en connaissant leurs dissimilarités. Le problème consiste alors à construire des hyper plans séparant les objets selon leur classe. L'objectif de la classifiaction non supervisée est opposé au cas supervisé : dans le premier cas, il s'agit de découvrir des groupes d'objets alors que dans le deuxième il s'agit d'utiliser les groupes connus pour découvrir ce qui les différencie ou afin de pouvoir classer de nouveaux objets dont la classe n'est pas connue. Prenons l'exemple d'une population d'individus dont on dispose de renseignements tels que l'âge, le sexe, le poids, la taille et le taux de cholestérol. Une méthode de classification non supervisée permettra par exemple de construire deux groupes : un groupe de végératiens et un groupe de carnivores. Une méthode de classification supervisée nécessite de savoir pour chaque individu s'il est végétarien ou non et permettra d'établir des critères permettant de différencier un individu selon ses attributs (âge, sexe, ...) ;
– classification hiérarchique/partitionnement. Une méthode de *classification hiérarchique* construit une séquence de partitions imbriquées, que l'on visualise par exemple par un dendogramme (figure 3.2), alors qu'un *partitionnement* ne construit qu'une seule partition des données.

3.2.2 Le problème étudié

La suite de ce chapitre est consacrée aux méthodes de classification développées pour traiter le problème de la classification non supervisée. De plus, nous ne nous intéresserons qu'au problème du partitionnement en laissant de côté la classification hiérarchique puisque nous ne la traiterons pas avec les fourmis artificielles. Alors que

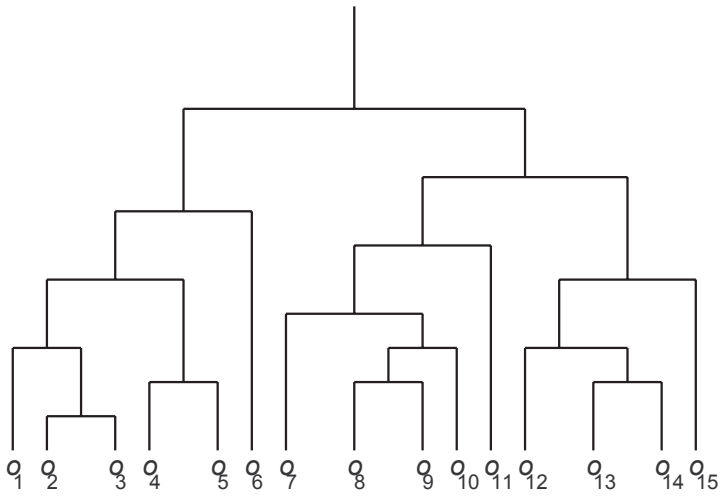


FIG. 3.2 – Exemple de dendogramme.

la classification hiérarchique est surtout utilisée pour construire des taxonomies dans les domaines de la biologie, des sciences comportementales ou sociales, les techniques de partitionnement sont surtout utilisées quand l'obtention d'une partition unique est primordiale, par exemple pour la représentation et la compression de bases de données importantes[1].

Le problème de partitionnement peut être formulé de la façon suivante. Étant donnés N points dans un espace métrique à M dimensions, on recherche une partition de ces points en K classes (ou groupe) de telle sorte que les points d'un même groupe soient plus similaires entre eux qu'avec les points des autres groupes. La valeur de K peut ne pas être connue. Comme pour toute méthode non supervisée, la qualité d'une méthode de partitionnement ne peut être jugée qu'à partir de ses résultats : c'est l'expert du domaine qui peut déterminer la pertinence des résultats obtenus.

3.2.3 Complexité du problème de partitionnement

En supposant que l'on dispose d'un critère fiable pour juger de la qualité d'une partition on peut envisager d'énumérer toutes les partitions possibles d'un ensemble d'objets. Si l'on note par $S(N, K)$ le nombre de partitionnements de N objets en K groupes, par une récurrence sur K, on peut calculer cette quantité (Jain and Dubes, 1988) :

$$S(N,K) = \frac{1}{K!}\sum_{i=1}^{K}(-1)^{K-i}\binom{K}{i}(i)^N \tag{3.1}$$

A titre d'exemple, $S(10,4) = 34\,105$ et $S(19,4) = 11\,259\,666\,000$, l'énumération complète devient en effet rapidement impraticable même pour des problèmes de petite

[1]La représentation d'une classification hiérarchique par un dendogramme est particulièrement inadaptée quand la base de données se compose de plusieurs centaines d'objets.

taille.

3.2.4 Quelques définitions

Etant donné un ensemble $O = \{o_1, \ldots, o_N\}$ de N objets correspondant chacun à un point d'un espace métrique à M dimensions dont les coordonnées sont notées par le vecteur $\mathbf{x}_i = (x_{i1}, \ldots, x_{iM})$ pour l'objet o_i, on peut définir les notions suivantes :

La distance

Rappelons qu'une mesure de distance $d(\cdot)$ doit vérifier les propriétés suivantes :

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \tag{3.2}$$

$$d(\mathbf{x}, \mathbf{y}) \geq 0 \tag{3.3}$$

$$d(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y} \tag{3.4}$$

$$d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}) \tag{3.5}$$

où $\mathbf{x}$, $\mathbf{y}$ et $\mathbf{z}$ sont des vecteurs.

Une distance étant définie, elle est assimilable à une mesure de dissimilarité. Etant donné que nous nous intéressons à des espaces métriques, l'indice de dissimilarité le plus communément utilisé est la métrique de Minkowski :

$$d_r(o_i, o_j) = d_r(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^{M} \omega_k |x_{ik} - x_{jk}|^r \right)^{\frac{1}{r}} \tag{3.6}$$

ω_k est un facteur de pondération que nous considèrerons égal à 1 par la suite. Suivant la valeur de r ($r \geq 1$) on obtient les mesures suivantes :

- $r = 1$: distance de Manhattan ;
- $r = 2$: distance Euclidienne ;
- $r = \infty$: $d_\infty(\mathbf{x}_i, \mathbf{x}_j) = \max_{1 \leq k \leq M} |x_{ik} - x_{jk}|$.

Ces mesures sont souvent utilisées pour des données numériques. Dans le cas de données symboliques, d'autres distances doivent être utilisées. La plus connue étant la distance de Hamming, à l'origine définie pour mesurer la distance entre des chaînes binaires, et qui correspond au nombre de bits différents dans les deux chaînes pour une même position.

Par la suite, nous noterons la mesure $d_2(\cdot)$ par $d(\cdot)$ pour alléger les notations.

L'erreur quadratique

L'erreur quadratique est un critère des plus courants parmi les critères utilisés pour le partitionnement. Considérons qu'une partition de O a été obtenue sous la forme de K classes $c_1, \ldots, c_K$ composées respectivement de $|c_1|, \ldots, |c_K|$ objets. Comme chaque objet appartient à une et une seule classe on a $\sum_{i=1}^{K} |c_i| = N$. Le centre de gravité $\mathbf{g}_i$ de la classe c_i est donné par :

$$\mathbf{g}_i = \frac{1}{|c_i|} \sum_{j=1}^{|c_i|} \mathbf{x}_j^{(i)} \tag{3.7}$$

où $\mathbf{x}_j^{(i)}$ est le j-ième point de la classe c_i. L'erreur quadratique ε_i^2 sur la classe c_i est alors donnée par :

$$\varepsilon_i^2 = \sum_{j=1}^{|c_i|} d^2(\mathbf{x}_j^{(i)}, \mathbf{g}_i) \tag{3.8}$$

L'erreur quadratique (aussi appelée *inertie intraclasse* : $\mathcal{I}_\mathrm{W}$) de la partition est alors :

$$\mathcal{I}_\mathrm{W} = E_q^2 = \sum_{j=1}^{K} \varepsilon_j^2 \tag{3.9}$$

Bien évidemment, ce critère ne peut servir à comparer des partitions ayant un nombre de classe différent. En effet, si $K = N$ alors $E_q = 0$ car $\mathbf{x}_i = \mathbf{g}_i \quad \forall i \in \{1, \ldots, N\}$ et l'erreur quadratique est minimale.

Le vecteur moyen $\mathbf{g}$ de l'ensemble O est donné par :

$$\mathbf{g} = \frac{1}{N} \sum_{i=1}^{K} |c_i| \mathbf{g}_i \tag{3.10}$$

Enfin, l'inertie interclasse, notée $\mathcal{I}_\mathrm{B}$, est donnée par :

$$\mathcal{I}_\mathrm{B} = \sum_{i=1}^{K} |c_i| d^2(\mathbf{g}_i, \mathbf{g}) \tag{3.11}$$

L'inertie totale $\mathcal{I}$:

$$\mathcal{I} = \mathcal{I}_\mathrm{B} + \mathcal{I}_\mathrm{W} \tag{3.12}$$

est stable pour un nombre de classes K fixé.

3.3 Tour d'horizon des méthodes de partitionnement

Les méthodes de résolution du problème de partitionnement peuvent se comporter de deux façons :

- méthode agglomérative : chaque groupe d'objet est constitué par agglomération des objets autour d'un centre (un noyau). Au départ il peut y avoir autant de groupes que d'objets ;
- méthode divisive. La démarche est inverse : on démarre avec un groupe contenant tous les objets puis le partitionnement est construit en divisant successivement ce groupe initial en groupes plus petits.

D'autres critères permettent de différencier les méthodes de classification non supervisée : par exemple si les objets sont traités un par un ou simultanément, ou encore si les attributs sont pris en compte un par un ou tous ensemble.

3.3.1 L'algorithme des centres mobiles

L'algorithme des centres mobiles (K-MEANS ou C-MEANS) est une technique de partitionnement des plus simples qui puisse être en utilisant l'erreur quadratique comme critère d'évaluation d'une partition. Dans un premier temps, les objets sont regroupés autour de K centres arbitraires $\mathbf{c}_1, \ldots, \mathbf{c}_K$ de la manière suivante : la classe c_i associée au centre $\mathbf{c}_i$ est constituée de l'ensemble des points les plus proches de $\mathbf{c}_i$ que de tout autre centre. Géométriquement, cela revient à partager l'espace des points en K zones définies par les plans médiateurs des segments $[\mathbf{c}_i, \mathbf{c}_j]$. La figure 3.3 donne l'exemple d'une partition associée à trois centres dans le plan. Les centres de gravité $\mathbf{g}_1, \ldots, \mathbf{g}_K$

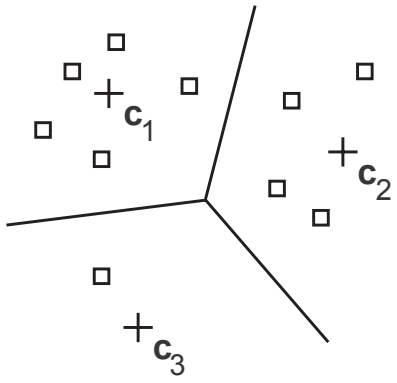


FIG. 3.3 – Exemple de partition obtenue par les centres mobiles.

sont ensuite calculés à partir des classes qui viennent d'être formées. On recommence l'opération en prenant comme centre de classe les centres de gravité trouvés et ainsi de suite jusqu'à ce que les objets ne changent plus de classe. L'algorithme 3.1 résume toutes ces opérations.

Algorithme 3.1: Algorithme des centres mobiles
K-MEANS(Partition P de K classes)

(1) **tantque** l'inertie intraclasse ne s'est pas stabilisée **faire**
(2) Générer une nouvelle partition P' en affectant chaque objet à la classe dont le centre est le plus proche
(3) Calculer les centres de gravité des classes de la nouvelle partition P'
(4) $P \leftarrow P'$
(5) **fintantque**
(6) **retourner** P

Remarques :

– le nombre de classes K fixé au début peut diminuer au cours des itérations : en effet, si une classe n'attire aucun objet, elle sera vide et donc éliminée. Cela peut

représenter un inconvénient si l'on désire obtenir K classes non vide ;
- on peut montrer que d'une itération à l'autre (i.e. d'une partition à l'autre), l'inertie intraclasse $\mathcal{I}_{\mathrm{W}}$ décroît ce qui entraine la convergence de l'algorithme (Jain and Dubes, 1988, page 99) ;
- la complexité de cette méthode est en $\mathcal{O}(NMKT)$ où T est le nombre d'itérations effectuées (Jain and Dubes, 1988, page 100) ;
- le principal inconvénient des centres mobiles est que la partition finale dépend du choix de la partition initiale. Le minimum global n'est pas obligatoirement atteint, on est seulement certain d'obtenir la meilleure partition à partir de la partition de départ choisie (ceci grâce au deuxième point de ces remarques) ;
- de nombreuses variantes peuvent être rencontrées, par exemple au lieu de calculer le centre des classes après avoir affecté tous les objets, le centre peut être recalculé après chaque affectation ;
- la méthode des centres mobiles a été généralisée sous l'appellation de la méthode des nuées dynamiques (Diday and Simon, 1976). Au lieu de définir une classe par un seul point, son centre de gravité, on la définit par un groupe d'objets formant un « noyau ».

Afin de déterminer le nombre de classes en même temps qu'une partition acceptable, Ball et Hall (Ball and Hall, 1965) ont proposé l'algorithme ISODATA. Par rapport aux centres mobiles, ISODATA possède les fonctionalités suivantes :

- une classe peut être détruite si elle ne comporte pas suffisamment d'objets ;
- deux classes peuvent être agglomérées si la distance entre leurs centres est inférieure à un seuil donné ;
- une classe peut être scindée en deux si la dispersion des objets la constituant est supérieure à un seuil donné.

L'algorithme ISODATA est détaillé en annexe A.

Le nombre de méthodes de partitionnement est pratiquement aussi important que le nombre de problèmes posés, aussi nous renvoyons le lecteur aux ouvrages (Bow, 1984) et (Jain and Dubes, 1988) pour un exposé complet des méthodes de partitionnement classiques.

La suite de ce chapitre expose rapidement certaines inspirations biomimétiques pour la résolution du problème de classification non supervisée. Nous exposons tout d'abord l'utilisation des réseaux de neurones artificiels.

3.3.2 Les réseaux de neurones artificiels

L'apprentissage non supervisé peut être traité par les réseaux de neurones artificiels (RNA). Les cellules d'entrée du réseau correspondent chacune à un attribut des objets. Les cellules de sortie donnent la classe de l'objet. La figure 3.4 présente un réseau pouvant accepter des objets à cinq attributs (cellules noires) et peut founir de une à trois classes (cellules blanches). Des poids ω_{kj} sont associés aux connexions entre chaque cellule d'entrée (E_k) et chaque cellule de sortie (S_j). La sortie s_j d'une cellule

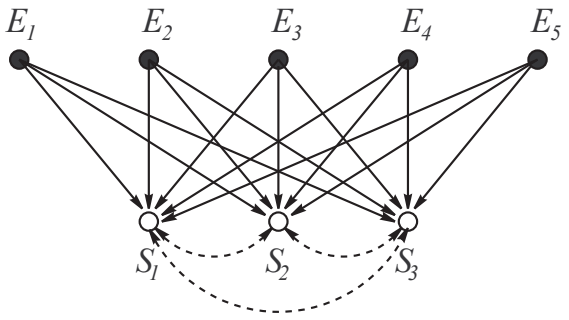


FIG. 3.4 – Réseau de neurone artificiels : 5 cellules d'entrée $(E_1, \ldots, E_5)$ et 3 cellules de sortie $(S_1, \ldots, S_3)$.

S_j est calculée de la façon suivante pour l'objet o_i :

$$s_j(o_i) = \sum_{k=1}^{M} \omega_{kj} x_{ik} \tag{3.13}$$

où x_{ik} est la valeur de l'attribut k de l'objet o_i. Les connexions entre les cellules de sortie servent à sélectionner la sortie la plus élevée comme classe. L'apprentissage des poids du réseau se fait pour chaque donnée. Les poids sont tout d'abord initialisés aléatoirement. Puis pour chaque objet o_i, les poids sont modifiés de la valeur suivante :

$$\Delta\omega_{kj} = LR(x_{ik} - \omega_{kj}) \tag{3.14}$$

Cette étape est répétée jusqu'à ce que les poids se stabilisent ou selon d'autres heuristiques, par exemple en faisant décroitre LR au cours des itérations (Hertz et al., 1993).

Les cartes de KOHONEN (Kohonen, 1988) (*Self Organizing Maps*) sont assez proches du modèle qui vient d'être proposé mais aussi de l'algorithme K-MEANS. Les exemples sont présentés au réseau dans un ordre aléatoire et sont affectés à la cellule dont le représentant $\omega_j = (\omega_{1j}, \ldots, \omega_{Mj})$ est le plus proche, tout comme les objets sont affectés à la classe dont le centre de gravité est le plus proche dans le cas des centres mobiles. Ensuite, les poids sont mis à jour non seulement pour la cellule S_j choisie mais aussi pour les cellules voisines de S_j. Cette structure de voisinage est le plus souvent représentée sous la forme d'une grille à deux dimensions ce qui explique l'analogie avec une carte. La mise à jour des poids est assez proche de la formule 3.14 : pour chaque cellule S_v du voisinage de S_j les poids sont modifiés de la quantité suivante :

$$\Delta\omega_{kv} = h_{vj} LR(x_{ik} - \omega_{kv}) \tag{3.15}$$

où h_{vj} dépend de la proximité des cellules S_v et S_j, par exemple $h_{vj} = e^{-d(v,j)}$. Chaque neurone de la carte a donc tendance à représenter des objets assez proches des objets que représentent les neurones qui lui sont voisins. Au cours des itérations, la taille

du voisinage peut décroitre ainsi que le cœficient LR afin de stabiliser les poids. Les principaux inconvénients de cette méthode sont que la convergence dépend de l'ordre dans lequel les exemples sont présentés ainsi que la difficulté du choix de la taille du voisinage.

3.3.3 Les algorithmes génétiques

Un certain nombre de travaux ont appliqué les algorithmes génétiques à des problèmes de partitionnement. Nous donnons ici à titre d'exemple les travaux présentés dans (von Laszewski, 1991).

Chaque individu de la population représente une partition des objets. Une partition est alors représentée par une chaine de N entiers $(e_1, \ldots, e_N)$ où $e_i \in \{1, \ldots, K\} \forall i$. si $e_i = e_j$ alors les objets o_i et o_j se trouvent dans le même groupe. Si $N = 6$, $(2, 1, 2, 3, 3, 2)$ donne la partition $\{o_2\}, \{o_1, o_3, o_6\}, \{o_4, o_5\}$. L'opérateur de croisement consiste à choisir aléatoirement un groupe d'un parent p_1 et à le recopier dans un parent p_2. Les objets qui appartenaient à ce groupe dans p_2 mais pas dans p_1 sont redistribués aléatoirement parmi les groupes de p_2. La mutation consiste à échanger deux valeurs e_i et e_j dans une partition.

D'autres représentations ont été proposées (Jones and Beltrano, 1991), par exemple en introduisant des séparateurs dans la chaîne d'entiers représentant une partition : par exemple si $N = 6$, $(4, 3, 7, 1, 5, 2, 8, 6)$ donne la partition $\{o_4, o_3\}, \{o_1, o_5, o_2\}, \{o_6\}$ car 7 et 8 représentent des séparateurs. Faulkenauer a fait remarquer que ces représentations prennaient en compte uniquement l'affectation des objets aux classes et propose un codage des solutions incluant les groupes (Falkenauer, 1994). Par exemple, les deux partitions suivantes, $\{o_2\}, \{o_1, o_3, o_6\}, \{o_4, o_5\}$ et $\{o_2, o_3, o_4\}, \{o_1, o_5, o_6\}$, seraient codées par $(2, 1, 2, 3, 3, 2 : 2, 1, 3)$ et $(1, 2, 2, 2, 1, 1 : 1, 2)$ où la première partie (avant les deux points) représente l'affectation des objets aux groupes et où la deuxième représente les groupes présents. Le principal avantage de cette représentation apparaît lorsque l'opérateur de croisement est appliqué : seules les deuxièmes parties sont croisées, les premières parties en subissant les conséquences.

Plusieurs présentations des travaux associant les algorithmes génétiques et les problèmes de classification peuvent être trouvées dans (Cucchiara, 1993; Kettaf, 1997).

3.4 Avec les fourmis

3.4.1 Ce que font les fourmis réelles

Dans la nature, les fourmis offrent un modèle stimulant pour le problème du partionnement. L'exemple du tri collectif du couvain ou de la constitution de cimetières sont les plus marquants. Certains travaux expérimentaux montrent que certaines espèces de fourmis sont capables d'organiser spacialement divers éléments du couvain : les œufs, les larves et les nymphes (Deneubourg et al., 1990; Franks and Sendova-Franks, 1992).

Le modèle de règles utilisé est relativement simple :

– lorsqu'une fourmi rencontre un élément du couvain, la probabilité qu'elle s'en empare est d'autant plus grande que cet élément est isolé ;

– lorsqu'une fourmi transporte un élément du couvain, elle le dépose avec une probabilité d'autant plus grande que la densité d'éléments du même type dans le voisinage est grande.

Pour rassembler en tas un ensemble d'éléments (d'objets) de même type, les probabilités de ramasser un objets (p_p) et de le déposer (p_d) ont été explicitées (Deneubourg et al., 1990) : quand une fourmi ne transporte aucun élément, sa probabilité d'en ramasser un, rencontré sur son chemin, est donnée par :

$$p_p = \left(\frac{k_1}{k_1 + f}\right)^2 \tag{3.16}$$

où k_1 est une constante positive et f correspond à la proportion d'éléments perçus dans le voisinage de la fourmi. Quand il y a peu d'objets dans le voisinage de l'objet convoité par la fourmi, $f \ll k_1$ ce qui signifie que p_p est proche de 1 et l'objet a beaucoup de chance d'être ramassé. Inversement, quand le voisinage est dense en éléments, $f \gg k_1$ et alors p_p est proche de 0. Quand une fourmi chargée d'un objet se déplace, sa probabilité de déposer l'objet est donnée par :

$$p_d = \left(\frac{f}{k_2 + f}\right)^2 \tag{3.17}$$

où k_2 est une constante positive. L'évaluation de f est proposée pour une implantation en robotique : f correspond au nombre d'objets rencontrés durant les T derniers déplacements divisé par le nombre maximum d'objets qui auraient pu être rencontrés. Les résultats obtenus par simulation montrent l'apparition de groupes d'objets, les agents ainsi définis permettent donc de ranger une surface sur laquelle des objets ont été éparpillés. L'adaptation de ce principe à des objets de plusieurs types, par exemple deux : A et B, peut alors se faire en particularisant f : f_A et f_B correspondent à la proportion d'objets de type A et B. Le comportement de rassemblement se transforme alors en tri, ce qui se rapproche de notre problème de partionnement.

En plus de ce mécanisme d'auto-organisation collective, on peut constater que les fourmis peuvent aussi se spécialiser pour un certain type d'éléments par un mécanisme d'apprentissage (Sendova-Franks and Franks, 1992). L'auto-organisation n'est alors plus seulement spaciale mais aussi temporelle. Rappelons aussi que les règles de comportement décrites sont individuelles alors que de nombreuses espèces sont capables de transporter collectivement des charges trop lourdes (par exemple l'espèce *Ectatoma ruidum* (Schatz et al., 1997)).

Ces principes ont trouvé leurs premières applications en robotique collective et de nombreux travaux en découlent (Beckers et al., 1994; Martinoli et al., 1999; Melhuish, 1999).

3.4.2 Les fourmis artificielles

Les travaux précédement exposés supposent que la discrimination entre les différents types d'objets soit effectuée sans difficulté. Hors c'est exactement l'hypothèse inverse qui est posée pour un problème de partitionnement : étant donné un ensemble d'objets,

on désire y percevoir un certain nombre de groupes ayant chacun une cohérence élevée. LUMER et FAIETA ont proposé un algorithme utilisant une mesure de dissimilarité entre les objets (sous la forme d'une distance euclidienne) (Lumer and Faieta, 1994). Les objets qui, rappelons le, correspondent à des points d'un espace numérique à M dimensions sont plongés dans un espace discret de dimension moindre (typiquement de dimension 2). Cet espace discret s'apparente alors à une grille G dont chaque case peut contenir un objet. Les agents se déplacent sur G et percoivent une région R_s de $s \times s$ cases dans leur voisinage. La figure 3.5 donne un exemple de grille avec une fourmi (représentée par $\times$) et son périmètre de détection (en trait épais). Les objets sont représentés par des carrés dont l'intérieur (« invisible » pour la fourmi) représente la classe d'origine.

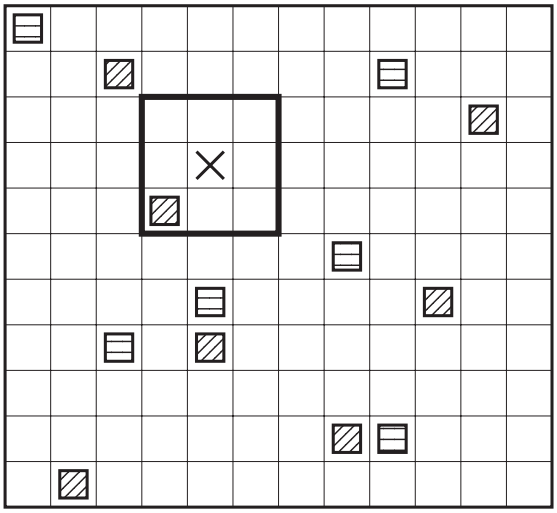


FIG. 3.5 – Grille utilisée dans (Lumer and Faieta, 1994). La fourmi est représentée par $\times$ et son périmètre de détection par un trait épais. Les objets sont représentés par des carrés dont l'intérieur (« invisible » pour la fourmi) représente la classe d'origine.

Les formules 3.16 et 3.17 sont modifiées de la façon suivante :

$$p_p(o_i) = \left(\frac{k_1}{k_1 + f(o_i)} \right)^2 \tag{3.18}$$

$$p_d(o_i) = \begin{cases} 2f(o_i) & \text{si } f(o_i) < k_2 \\ 1 & \text{si } f(o_i) \geq k_2 s \end{cases} \tag{3.19}$$

Comme on peut le remarquer, la fonction de densité locale dépend de l'objet considéré o_i et de sa position sur la grille $r(o_i)$. Elle est calculée de la manière suivante :

$$f(o_i) = \begin{cases} \frac{1}{s^2} \sum_{o_j \in R_s(r(o_i))} 1 - \frac{d(o_i,o_j)}{\alpha} & \text{si } f > 0 \\ 0 & \text{sinon} \end{cases} \tag{3.20}$$

$f(o_i)$ est alors une mesure de la similarité moyenne de l'objet o_i avec les objets o_j présents dans son voisinage. α est un facteur d'échelle déterminant dans quelle mesure

la dissimilarité entre deux objets est prise en compte. L'algorithme 3.2 donne les étapes de la méthode en utilisant A fourmis $\{a_1, \ldots, a_A\}$. Les paramètres ont les valeurs suivantes : $k_1 = 0.1$, $k_2 = 0.15$, $s = 3$, $\alpha = 0.5$ et $T_{\max} = 10^6$.

Algorithme 3.2: Algorithme LF (Lumer and Faieta, 1994)
LF()

(1) Placer aléatoirement les N objets $o_1, \ldots, o_N$ sur la grille G
(2) **pour** $T = 1$ à $T_{\max}$ **faire**
(3) **pour tout** $a_j \in \{a_1, \ldots, a_A\}$ **faire**
(4) **si** la fourmi a_j ne transporte pas d'objet et $r(o_i) = r(a_j)$ **alors**
(5) Calculer $f(o_i)$ et $p_p(o_i)$
(6) La fourmi a_j ramasse l'objet o_i suivant la probabilité $p_p(o_i)$
(7) **sinon**
(8) **si** la fourmi a_j transporte l'objet o_i et la case $r(a_j)$ est vide **alors**
(9) Calculer $f(o_i)$ et $p_d(o_i)$
(10) La fourmi a_j dépose l'objet o_i sur la case $r(a_j)$ avec une probabilité $p_d(o_i)$
(11) **finsi**
(12) **finsi**
(13) Déplacer la fourmi a_j sur une case voisine non occupée par une autre fourmi
(14) **finpour**
(15) **finpour**
(16) **retourner** l'emplacement des objets sur la grille

La figure 3.6 donne un résultat possible de l'exécution de l'algorithme LF sur la grille de la figure 3.5. On peut remarquer que le nombre de groupes d'objets (3) ne correspond pas obligatoirement au nombre de classes original (2).

Des tests ont été effectués sur des données artificielles : 200 points sont générés dans un espace à deux dimensions ($[0, 1] \times [0, 1]$) suivant quatre lois gaussiennes de moyenne variable et d'écart type fixe. La grille utilisée comporte 100×100 cases et 10 fourmis sont utilisées. Les résultats obtenus montrent que généralement l'agorithme obtient plus de classes qu'il n'en existe dans la distribution initiale. Lumer et Faieta ont apporté un certain nombre d'améliorations pour réduire cette tendance :

- chaque fourmi a été dotée d'une vitesse v pouvant varier entre 1 et $v_{\max} = 6$ d'une fourmi à l'autre. Le calcul de $f(o_i)$ a été modifié pour que les fourmis les plus rapides soient moins sensibles à la dissimilarité entre deux objets que les plus lentes ;
- chaque fourmi dispose d'une mémoire à court terme lui permettant de se rappeler de l'emplacement des m derniers objets qu'elle a déposés. A chaque objet ramassé, la fourmi compare les caractéristiques de cet objet aux m objets de sa mémoire et se dirige alors vers le plus similaire. Cette technique permet de réduire le nombre de groupes d'objets équivalents ;

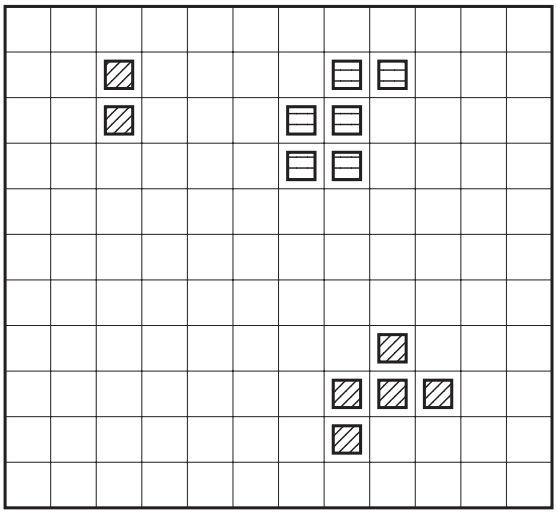


FIG. 3.6 – Résultat possible de l'exécution de l'algorithme LF sur la grille de la figure 3.5.

– comme les objets ont de moins en moins de chance d'être déplacés, un mécanisme de redémarrage a été ajouté : si une fourmi n'a pas effectué d'action depuis un certain temps, elle peut détruire un groupe en ramassant un objet.

Cet algorithme a été expérimenté sur des bases de données de taille importante et les résultats, tant du point de vue de la classification que de l'intérêt de la représentation planaire se sont montrés prometteurs[2]. Les principales critiques concernent le manque de comparaison avec d'autre méthodes de classification ainsi que le temps de calcul relativement important ($T_{\max} = 10^6$). De plus, l'interprétation des résultats devient hasardeuse quand la taille de la base augmente puisque la frontière entre deux groupes d'objets peut être réduite à une case vide alors que ces deux groupes peuvent représenter deux groupes d'objets très distincts.

Ces travaux ont inspiré KUNTZ, LAYZELL et SNYERS (Kuntz et al., 1997; Kuntz et al., 1998) pour résoudre le problème du partitionnement de graphes. À la place des objets manipulés par l'algorithme LF, les sommets d'un graphe sont répartis sur une grille à deux dimensions et les agents-fourmis rassemblent les sommets en fonction de leur connexité dans le graphe. Les applications de ce type d'approche vont de la conception de circuits VLSI à la représentation géométrique de graphes.

On peut trouver une version supervisée de ces algorithmes dans (Bonabeau et al., 1999) se basant sur la capacité des insectes sociaux à suivre des patrons, par exemple mis en œuvre lors de la construction d'une termitière.

Les travaux de LANGHAM et GRANT (Langham and Grant, 1999b; Langham and Grant, 1999a) portent aussi sur le partitionnement de graphes. Le problème consiste à trouver un partitionnement de graphe de façon à minimiser le nombre de coupures entre les partitions. Ce genre de problème est rencontré dans les systèmes de simulation

[2]Lumer, E. and Faieta, B. (1995) Exploratory Database Analysis via Self-Organization. non publié.

où les traitements sont répartis sur plusieurs processeurs. L'objectif est de construire des groupes de traitement (un traitement est représenté par un sommet du graphe) de tailles relativement homogènes afin de minimiser les communications entre processeurs (arcs du graphe). Les sommets du graphe sont répartis sur une grille en tenant compte de leur connectivité. Ils représentent la nourriture que plusieurs colonies de fourmis tenteront de ramasser. Ces colonies de fourmis sont aussi placées sur la grille mais à une position non aléatoire (la répartition des nids est faite en fonction de la dispersion de la nourriture). La stratégie individuelle des fourmis est élaborée par programmation génétique. La capture d'un sommet-proie est décidée de telle sorte que le nombre de coupures occasionnées ne soit pas trop importante. En fonction du nombre de coupures générées, la nourriture possède un certain poids. Si ce poids est trop important, les fourmis peuvent collaborer pour ramener cette nourriture au nid. Un groupe de sommets est constitué des sommets ramenés au même nid. La partition obtenue correspond aux groupes de sommets obtenus. Pour traiter des graphes de grande taille, le graphe est traité en plusieurs étapes : les premières étapes manipulent des groupes de sommets et les dernières étapes traitent les sommets individuellement. Les résultats présentés portent sur des graphes de 286 sommets avec 1046 arcs jusqu'à 4720 sommets avec 13722 arcs. Sur ces exemples la méthode inspirée du fourragement des fourmis donne les meilleurs résultats, concernant la minimisation du nombre de coupure entre les groupes de sommets formés, par rapport à deux autres méthodes classiques.

Alexandrov (Alexandrov, 2000) s'est intéressé à un problème de classification supervisée où l'objectif est de trouver une partition des données en K classes qui minimise le plus grand diamètre rencontré dans la partition (le diamètre d'une classe correspond à la plus grande distance séparant deux objets de la classe). La méthode est agglomérative et les phéromones sont utilisées pour choisir à chaque itération les objets qui seront agglomérés. Malheureusement, le papier est trop court pour qui voudrait se faire une idée plus précise des mécanismes réellement mis en œuvre.

3.5 Conclusion

Dans ce chapitre nous avons défini le problème de classification non supervisé que nous allons traiter dans le chapitre suivant. Nous y avons exposé l'algorithme des centres mobiles (K-means) et l'algorithme LF de Lumer et Faieta (Lumer and Faieta, 1994) basé sur les travaux de Deneubourg et ses collègues (Deneubourg et al., 1990). Ces deux algorithmes sont à la base des développements que nous présentons par la suite, l'algorithme AntClass.

Chapitre 4

L'algorithme AntClass

Nous présentons dans ce chapitre un nouvel algorithme de classification non supervisée. Cet algorithme découvre automatiquement les classes dans des données numériques sans connaître le nombre de classes a priori, sans partition initiale et sans paramétrage délicat. Il utilise les principes exploratoires stochastiques d'une colonie de fourmis avec les principes heuristiques et déterministes de l'algorithme des centres mobiles. Les fourmis se déplacent sur une grille à deux dimensions et peuvent transporter des objets. La saisie ou la dépose d'un objet sur un tas dépend de la similarité entre cet objet et les objets du tas. L'algorithme des centres mobiles est ensuite utilisé pour améliorer la partition obtenue par les fourmis. Nous appliquons cet algorithme sur des bases de données numériques artificielles et réelles.

4.1 Introduction

Comme nous l'avons constaté dans le chapitre précédent, la classification fait partie des problèmes pour lesquels les fourmis suggèrent des heuristiques très intéressantes pour les informaticiens. Une des premières études relatives à ce domaine a été menée par (Deneubourg et al., 1990) où une population d'agents-fourmis se déplacent aléatoirement sur une grille à deux dimensions et sont capables de déplacer des objets dans le but de les rassembler. Cette méthode à été étendue par (Lumer and Faieta, 1994) sur des objets simples puis par (Kuntz et al., 1997) où un problème réel est abordé dans le but de résoudre efficacement un problème d'optimisation. Le chapitre précédent donne le détail de ces algorithmes.

En se basant sur les travaux existants, nous contribuons à l'étude des fourmis classifieuses du point de vue de la découverte de connaissances, avec comme objectif de résoudre des problèmes réels. Dans de tels problèmes, nous considérons qu'un expert du domaine a collecté un ensemble de données et qu'il aimerait se voir proposer une partition de ses données en des classes pertinentes. Ceci suppose de pouvoir découvrir les classes intéressantes sans pouvoir disposer d'une partition de départ et sans connaître

le nombre de classes qui seront nécessaires. De plus, nous désirons traiter les données manquantes qui sont souvent rencontrées dans les problèmes réels. Deux points clés dans ce travail sont de proposer des résultats compréhensibles, puisque nous voulons une approche « découverte de connaissances », et aussi d'éviter un paramétrage complexe car dans la plupart des cas les experts du domaine ne sont pas des informaticiens et ne savent par conséquent pas comment régler les paramètres d'une méthode de classification. Le problème que nous nous proposons de résoudre est le problème de la classification non supervisée, ou partitionnement, qui a été exposé au chapitre précédent.

Comme nous le verrons par la suite, pour atteindre ces objectifs, nous avons développé un nouvel algorithme de classification basé sur les fourmis que nous avons appelé AntClass . Cet algorithme utilise des heuristiques basées sur les fourmis, plus générales que dans les approches antérieures. AntClass introduit une classification hiérarchique dans la population de fourmis artificielles qui seront aussi capables de transporter des tas d'objets. De plus, AntClass inclut une hybridation avec l'algorithme des centres mobiles afin de résoudre les inconvénients inhérents à la classification par les fourmis, par exemple pour accélérer la convergence en éliminant les erreurs « évidentes » de classification.

La suite de ce chapitre est organisée de la façon suivante : la section 4.2 présente les motivations qui nous sont apparues pour proposer une amélioration des travaux existants. La section 4.3 décrit l'algorithme AntClass. La section 4.4 décrit toutes les expérimentations qui ont été effectuées avec AntClass sur des ensembles de données artificielles ou issues de bases réelles. Enfin, la section 4.7 conclut sur les travaux futurs et sur les extensions possibles de AntClass.

4.2 Motivations

Comme il a été montré dans le chapitre 1, les fourmis sont capables d'accomplir collectivement des tâches relativement complexes de classification. Cette source d'inspiration a déjà été explorée à quelques reprises (voir le chapitre 3). Nous présentons dans cette section les motivations qui nous ont poussés à proposer une nouvelle approche.

Pour classer des données, ou partitionner un ensemble d'objets, de nombreux algorithmes déterministes (par exemple les centres mobiles ou Isodata, présentés dans le chapitre précédent) nécessitent qu'une partition initiale soit donnée en entrée. C'est l'inconvénient majeur de ces méthodes : la partition obtenue à partir de cette initialisation risque d'être localement optimale, le seul moyen de contourner ce problème étant de relancer la méthode avec une partition initiale différente. La même remarque peut être faite concernant le nombre de classes : les centres mobiles nécessitent que le nombre de classes soit initialisé, ce qui diminue l'intérêt de la méthode pour un expert cherchant justement à connaître ce nombre de classes.

L'introduction d'une recherche stochastique à la place d'une recherche déterministe peut améliorer les résultats, par exemple avec un algorithme génétique (Jones and Beltrano, 1991; Cucchiara, 1993) ou une population de fourmis. L'utilisation de fourmis artificielles pour la classification, à la place d'un AG par exemple, est pertinent dans le

sens où les fourmis réelles ont ce genre de problèmes à résoudre. Dans un certain sens, le modèle de fourmis artificielles pour la classification est certainement plus proche du problème de la classification que le modèle génétique et nous espérons qu'il sera plus performant du point de vue de la qualité comme de la rapidité.

Nous reprenons les travaux de Lumer et Faieta (Lumer and Faieta, 1994) pour en améliorer certains points (le chapitre précédent en offre une description). Dans cette méthode inspirée des fourmis, les objets sont initialement éparpillés aléatoirement sur une grille à deux dimensions. Les fourmis utilisent une mesure de similarité locale pour prendre leurs décisions. L'intérêt de ce travail est d'introduire les principes de classification par des fourmis dans des problèmes d'analyse des données. Cependant des inconvénients peuvent apparaître :

- chaque case de la grille ne peut accueillir qu'un seul objet à la fois. Ce qui signifie qu'une fourmi peut passer un certain nombre d'itérations à trouver un emplacement de libre à côté du groupe d'objets proches de celui qu'elle transporte ;
- le déplacement de fourmis étant aléatoire, certains objets peuvent ne pas avoir été transportés et restent non classés ;
- la convergence est lente ;
- les résultats obtenus par les fourmis ne sont pas comparés à ceux obtenus par d'autres méthodes ;
- l'interprétation des résultats est visuelle (la grille) et ne facilite pas leur exploitation. L'emplacement des groupes d'objets n'est par exemple pas obligatoirement corrélé à l'emplacement des objets dans l'espace des attributs.

Ce sont ce type de difficultés qui ont guidé la conception de l'algorithme AntClass présenté et étudié dans la suite de ce chapitre.

4.3 L'algorithme AntClass

Cette section décrit l'algorithme de classification non supervisée que nous avons développé à partir des travaux présentés dans le chapitre précédent. Nous reprenons les notations introduites à cette occasion.

4.3.1 Répartition des objets sur une grille

Comme dans les travaux de Lumer et Faieta (algorithme LF) (Lumer and Faieta, 1994), nous faisons évoluer des agents-fourmis sur une grille G où les objets à partitionner sont positionnés. Bien que le positionnement des objets soit initialement aléatoire, cela revient à répartir les points de l'espace numérique à M dimensions dans lequel les objets sont définis vers un espace discret à deux dimensions. Le caractère aléatoire de la disposition initiale n'est pas obligatoire, on pourrait par exemple envisager de placer les objets suivant le résultat obtenu par une analyse factorielle des correspondances (AFC) en discrétisant les coordonnées de chaque objet sur les premiers axes d'inertie (nous ne sommes pas obligés de nous limiter aux deux premiers axes si la grille comporte plus de deux dimensions).

Par rapport à la grille utilisée par l'algorithme LF nous introduisons plusieurs différences :

- la grille G est toroïdale, ce qui signifie que les fourmis passent d'un côté de la grille à l'autre en un seul pas. Cette caractéristique n'est pas présente pour l'algorithme LF et cela représente à notre avis l'inconvénient de provoquer des effets de bord indésirables (par exemple, la répartition des positions explorées par les fourmis peut ne plus être uniforme) ;
- G est de forme carrée et sa taille est déterminée automatiquement en fonction du nombre d'objets à traiter (ce qui n'est pas précisé pour LF). Si N représente le nombre d'objets, G comporte L cases par côté :

$$L = \left\lceil \sqrt{2N} \right\rceil \tag{4.1}$$

  Cette formule permet de s'assurer que le nombre de cases est au moins égal au nombre d'objets[1]. Par contre, si la grille est trop grande, les fourmis vont perdre beaucoup de temps à y chercher les objets. Le fait de déterminer automatiquement la taille de la grille signifie que, quelque soit le nombre d'objets, la probabilité qu'un mouvement aléatoire d'une fourmi lui permette de trouver un objet est indépendante de N. Nous avons expérimentalement remarqué que cette méthode donne de bons résultats ;
- dans LF, chaque case ne peut contenir qu'un seul objet, une classe est alors représentée par un amas d'objets (voir la figure 3.6 page 60). Dans notre cas, plusieurs objets peuvent être placés sur une seule case, ce qui forme un tas. Dans ce cas, une classe correspond à un tas et une partition est donnée par l'ensemble des tas présents sur la grille. La figure 4.1 illustre cette caractéristique. Le principal avantage de cette façon de procéder est qu'il n'est plus nécessaire de définir un voisinage pour l'estimation de la densité $f(o_i)$ d'objets similaires à l'objet o_i (formule 3.20). Dans LF, la similarité des objets est estimée localement (justement, grâce au voisinage) ce qui peut impliquer qu'un amas comportant beaucoup d'objets peut avoir des caractéristiques très différentes à deux de ses extrémités.

4.3.2 Déplacement des fourmis

À l'initialisation, les A fourmis $\{a_1, \ldots, a_A\}$ sont disposées aléatoirement sur la grille en vérifiant qu'une case ne peut accueillir qu'une seule fourmi. À chaque itération de l'algorithme, chaque fourmi a_i se déplace aléatoirement sur la grille à une vitesse $v(a_i)$. Concrètement, si $(x(a_i), y(a_i))$ sont les coordonnées de la fourmi a_i sur G, après un déplacement, la nouvelle position sera dans l'intervalle $([x(a_i) - v(a_i), x(a_i) + v(a_i)], [y(a_i) - v(a_i), y(a_i) + v(a_i)])$ en tenant compte du fait que G est toroïdale. Enfin la direction choisie par une fourmi dépend de sa direction précédente : elle a une probabilité égale à 0.6 de continuer tout droit et de 0.4 de changer de direction[2]. Dans ce cas, elle a une chance sur deux de tourner de 45 degrés à gauche ou à droite.

[1] Il faut au moins que $L^2 \geq N$ pour que chaque objet puisse être seul sur une case à l'initialisation.

[2] Ces deux valeurs ont été choisies arbitrairement et indépendamment de ce qui pourrait se produire dans la réalité.

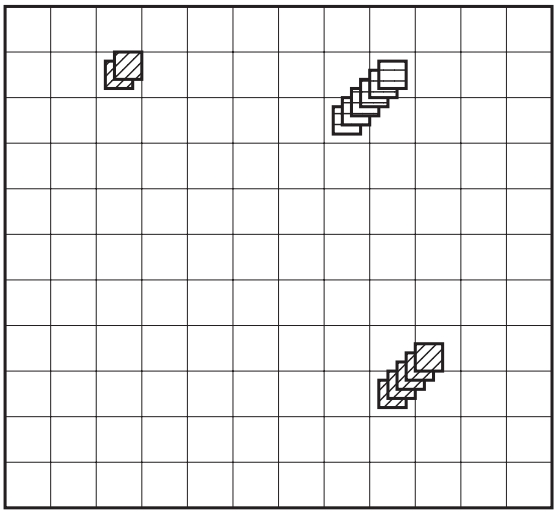


FIG. 4.1 – ANTCLASS permet la construction de tas d'objets sur la grille.

Dans LF, chaque fourmi ne peut transporter qu'un seul objet à la fois. Nous avons généralisé ce point en dotant chaque fourmi d'une capacité de transport $c(a_i)$. Par la suite, nous n'étudions que deux valeurs pour ce paramètre : une capacité $c(a_i) = 1$ et infinie ($c(a_i) = \infty$).

Afin de déterminer les règles que les fourmis vont utiliser sur la grille pour manipuler les objets, il faut leur donner certaines mesures, de dispersion d'un tas ou de la dissimilarité entre deux objets par exemple. Les notations suivantes vont servir par la suite :

– la distance maximale entre deux objets de l'ensemble O :

$$d^*(O) = \max_{(i,j)\in\{1,\ldots,N\}^2} \{d(\mathbf{x}_i, \mathbf{x}_j)\} \tag{4.2}$$

– la distance moyenne entre deux objets de l'ensemble O :

$$\bar{d}(O) = \frac{2}{N(N-1)} \sum_{(i,j)\in\{1,\ldots,N\}^2, i<j} d(\mathbf{x}_i, \mathbf{x}_j) \tag{4.3}$$

– la distance maximale entre les objets d'un tas T_j et son centre de gravité $\mathbf{g}_j$:

$$d_g^*(T_j) = \max_{\mathbf{x}_i\in T_j} \{d(\mathbf{x}_i, \mathbf{g}_j)\} \tag{4.4}$$

– la distance moyenne entre les objets d'un tas T_j et son centre de gravité $\mathbf{g}_j$:

$$\bar{d}_g(T_j) = \frac{1}{|T_j|} \sum_{\mathbf{x}_i\in T_j} d(\mathbf{x}_i, \mathbf{g}_j) \tag{4.5}$$

Ramassage d'objets

Si la fourmi a_i ne transporte pas d'objet et qu'elle se trouve sur une case contenant un objet ou un tas d'objets T_j, elle a une probabilité p_p de ramasser un objet :

$$p_p(T_j) = \begin{cases} 1 & \text{si } |T_j| = 1 \\ \min\left\{\left(\frac{\bar{d}_g(T_j)}{\bar{d}(O)}\right)^{k_1}, 1\right\} & \text{si } |T_j| = 2 \\ 1 - 0.9\left(\frac{\bar{d}_g(T_j)+\varepsilon}{d_g^*(T_j)+\varepsilon}\right)^{k_1} & \text{sinon} \end{cases} \tag{4.6}$$

où ε est une petite valeur positive (10^{-5}) et $|T_j|$ le nombre d'objets dans le tas. La fourmi ramasse les objets jusqu'à ce que sa capacité soit atteinte en choisissant à chaque fois l'objet o_i le plus éloigné du centre de gravité du tas. Si la case ne contient qu'un seul objet ($|T_j| = 1$), il est systématiquement ramassé par la fourmi. Si la case contient un tas de deux objets ($|T_j| = 2$ et $T_j = \{o_u, o_v\}$), la probabilité de ramasser l'un des objets dépend de la distance entre les deux objets et le centre de gravité $\mathbf{g}_j$ ($d(\mathbf{x}_u, \mathbf{g}_j) = d(\mathbf{x}_v, \mathbf{g}_j) = \bar{d}_g(T_j)$) et de la distance moyenne entre tous les objets ($\bar{d}(O)$). Enfin, si le tas se compose de plus de deux objets, p_p est proche de 1 quand la distance moyenne au centre est négligeable devant la distance au centre de l'objet le plus éloigné. k_1 est un paramètre réel positif permettant de contrôler la forme de la densité de $p_p(T_j)$ quand $|T_j| > 2$.

Si la capacité de la fourmi est supérieure à 1, elle ramasse autant d'objets que sa capacité le lui permet.

Dépôt d'objets

Si la fourmi transporte un objet, et qu'elle se trouve sur une case contenant un ou plusieurs objets, sa probabilité de déposer l'objet o_i sur le tas T_j est donnée par :

$$p_d(o_i, T_j) = \begin{cases} 1 & \text{si } d(\mathbf{x}_i, \mathbf{g}_j) \leq d_g^*(T_j) \\ 1 - 0.9 \min\left\{\left(\frac{d(\mathbf{x}_i, \mathbf{g}_j)}{\bar{d}(O)}\right)^{k_2}, 1\right\} & \text{sinon} \end{cases} \tag{4.7}$$

où k_2 est un paramètre réel positif permettant de contrôler la forme de la densité de $p_d(o_i, T_j)$ quand $d(\mathbf{x}_i, \mathbf{g}_j) > d_g^*(T_j)$. Si l'objet o_i transporté par la fourmi est moins éloigné du centre $\mathbf{g}_j$ du tas T_j que l'objet du tas le plus éloigné de ce centre, elle dépose systématiquement o_i. Sinon, plus la distance entre o_i et $\mathbf{g}_j$ ($d(\mathbf{x}_i, \mathbf{g}_j)$) est grande par rapport à la distance moyenne entre les objets de la base ($\bar{d}(O)$), plus la probabilité de déposer sera faible.

Si la capacité de la fourmi est supérieure à 1 et qu'elle transporte plusieurs objets, la probabilité de déposer le tas T_i qu'elle transporte sur le tas T_j est calculée de la même façon que pour un objet unique en remplaçant $\mathbf{x}_i$ par le centre de gravité $\mathbf{g}_i$ des objets transportés.

Patience des fourmis

S'il y a trop de fourmis par rapport au nombre de tas ou d'objets, on peut tomber sur le problème suivant : tous les tas (dans le cas d'une grande capacité de transport

des fourmis) ou tous les objets sont transportés ce qui n'offre plus de possibilité aux fourmis de déposer ce qu'elles transportent. Dans le cas où les fourmis ont une capacité de transport égale à 1, il suffit de s'assurer que le nombre de fourmis est nettement inférieur au nombre d'objets. Par contre quand les fourmis ont une capacité de transport supérieure, le problème peut réapparaître. La solution la plus immédiate est de doter les fourmis d'une certaine patience, qui peut être individuelle, et notée $p(a_i)$. Quand la fourmi a effectué plus de $p(a_i)$ déplacements sans avoir réussi à déposer les objets qu'elle transporte, elle les dépose sur la case où elle se trouve si elle est vide ou l'une de son voisinage dans le cas contraire. Par la suite, cette patience sera utilisée en particulier quand la capacité $c(a_i)$ d'une fourmi est supérieure à 1.

Mémoire des fourmis

Lumer et Faieta ont introduit un mécanisme de mémoire à court terme pour accélérer le processus d'agrégation. Quand un objet o_i est ramassé par une fourmi a_i, il est comparé aux $m(a_i)$ mémoires de la fourmi (où $m(a_i)$ représente la taille de la mémoire de la fourmi a_i). Elle se dirige ensuite vers l'emplacement de l'objet le plus similaire à o_i.

Nous avons adapté cette technique en remplacant la comparaison des objets sur la distance les séparant par la distance entre le centre de gravité du tas transporté par la fourmi et les tas qu'elle a mémorisés, puisque nos fourmis peuvent transporter plusieurs objets. Dans le cas où la fourmi ne transporte qu'un seul objet, il se confond avec le centre de gravité du tas qu'il forme à lui tout seul. La fourmi gère sa mémoire sous la forme d'une liste FIFO où les positions les plus anciennes sont oubliées quand la fourmi mémorise de nouvelles positions. Cette mémorisation est effectuée quand la fourmi dépose un ou plusieurs objets sur un tas.

Concernant la taille de la mémoire à utiliser et en dehors de toute expérimentation, on peut envisager qu'une grande mémoire sera coûteuse à gérer du point de vue du temps de calcul puisqu'à chaque ramassage d'un ou plusieurs objets la fourmi calcule la distance entre le centre de gravité de ce qu'elle vient de ramasser et le centre de gravité de chacun des tas mémorisés. D'un autre côté, si la mémoire est trop petite, comme la fourmi choisit de se diriger vers le tas dont le centre de gravité est le plus proche, son éventail de choix pourrait être trop restreint. Ce qui signifie que son déplacement pourrait être inutile si elle ne dépose rien sur le tas qu'elle a choisi d'atteindre.

La mémoire d'une fourmi est statique dans le sens où le centre du tas dont elle mémorise la position ne varie plus de son point de vue. Si elle revient sur un tas, il est possible qu'il ait été suffisament modifié pour que sa mémoire ne l'ait pas bien orientée, il peut même avoir disparu.

4.3.3 Hybridation avec les centres mobiles

Le partitionnement construit par les fourmis n'est pas obligatoirement optimal au sens de l'inertie intra-classe $\mathcal{I}_w$ (formule 3.9) même si les règles de ramassage et de dépôt des objets tendent à agglomérer les objets à des tas dont le centre est proche. Comme nous l'avons vu dans le chapitre précédent, l'algorithme des centres mobiles (ou plus

couramment K-MEANS : algorithme 3.1) offre une réponse simple à la non-uniformité de la partition construite par les fourmis. Comme les fourmis agissent localement il peut rester un certain nombre d'objets « méritant » d'appartenir à une classe dont le centre est beaucoup plus proche. K-MEANS est alors utilisé pour corriger ce type de défaillance.

Dans des versions précédentes de ANTCLASS nous avions décidé d'affecter les objets isolés (c'est-à-dire se trouvant sur une fourmi ou seul sur la grille) au tas dont le centre de gravité était le plus proche. Cela est intéressant quand une fourmi est interrompue dans une opération qu'elle pouvait accomplir ou pour les objets qui n'auraient pas été visités sur la grille. Ce deuxième point peut être évité en donnant plus d'itérations aux fourmis, certes au détriment du temps de calcul. Cependant, on peut raisonnablement penser qu'un objet dont les paramètres sont trop bruités ait du mal à trouver une place sur un tas, il a donc plus de chances d'être seul sur la grille (si une fourmi impatiente l'a finalement laissé tomber) ou bien transporté par une fourmi au moment de l'interruption. Il peut être alors intéressant qu'il reste isolé afin d'attirer l'attention, l'algorithme K-MEANS ayant beaucoup de chance de le laisser seul dans sa classe.

4.3.4 Algorithmes

LUMER et FAIETA ont utilisé une population d'agents dont les paramètres individuels diffèrent d'un agent à l'autre. Telles que nous les avons exposées, nos fourmis peuvent être hétérogènes quant à leurs paramètres et les fourmis que nous utiliserons au chapitre 7 auront aussi cette propriété. Nous adoptons cependant une autre stratégie qui nous semble plus adaptée au problème de classification non supervisée : le travail des fourmis peut être interrompu pour lancer les centres mobiles puis reprendre la nouvelle partition avec des paramètres différents mais homogènes. Cela permet de fournir après chaque itération des fourmis et des centres mobiles une partition des données ce qui se rapproche de la classification hiérarchique si par la suite de moins en moins de classes sont proposées.

L'algorithme ANTCLASS désigne une succession d'itérations des fourmis et des centres mobiles. Les fourmis ont pour tâche de réduire le nombre de classes et les centres mobiles d'améliorer globalement la partition découverte par les fourmis.

L'algorithme 4.1 donne la structure générale de la méthode de classification par les fourmis (appelé ANTS par la suite).

L'algorithme 4.2 donne le schéma général de ANTCLASS. L'algorithme K-MEANS est initialisé avec la partition obtenue par ANTS.

Le tableau 4.1 résume les paramètres à fixer pour l'algorithme ANTS. Ces paramètres sont à spécifier pour chacune des T_{AntClass} itérations de ANTCLASS.

4.4 Etude expérimentale

4.4.1 Evaluation des résultats

Afin d'évaluer les résultats obtenus par ANTCLASS nous avons utilisé des bases de données numériques artificielles que nous avons générées et des bases réelles issues

Algorithme 4.1: Algorithme ANTS : regroupement des objets par les fourmis. Les indications entre crochets concernent le cas où les fourmis ont une capacité de transport supérieure à 1.

ANTS(Grille G)

(1) **pour** $t = 1$ à T **faire**
(2) **pour** $k = 1$ à A **faire**
(3) Déplacer la fourmi a_k sur une case non occupée par une autre fourmi
(4) **si** il y a un tas d'objets T_j sur la même case que a_k **alors**
(5) **si** la fourmi a_k transporte un objet o_i [un tas d'objets T_i] **alors**
(6) Déposer l'objet o_i [le tas T_i] transporté par la fourmi sur le tas T_j suivant la probabilité $p_d(o_i, T_j)$ [$p_d(T_i, T_j)$] (équation 4.7)
(7) **sinon**
(8) `/* La fourmi ne transporte pas d'objet */` Ramasser l'objet o_i le plus dissimilaire du tas T_j [jusqu'à ce que la capacité $c(a_k)$ de la fourmi soit atteinte ou que le tas soit vide] selon la probabilité $p_p(T_j)$ (équation 4.6)
(9) **finsi**
(10) **finsi**
(11) **finpour**
(12) **finpour retourner** la grille G

du *Machine Learning Repository* (Blake and Merz, 1998). Ces bases de données sont supervisées (pour chaque objet on en connaît la classe) afin de pouvoir évaluer la qualité du partitionnement que nous obtenons. Cette classe n'est bien entendu pas donnée à ANTCLASS.

La question de la validité d'une partition est assez difficile puisque cela peut concerner plusieurs questions simultanément :

– combien de classes sont présentes dans les données ?
– entre deux partitions, laquelle est la plus adaptée à partitionner les données ?

Nous avons utilisé la mesure d'erreur suivante. Si, pour chaque objet o_i on connait sa classe d'origine $c(o_i)$ et la classe obtenue par ANTCLASS $c'(o_i)$, l'erreur de classification E_c est calculée de la façon suivante :

$$E_c = \frac{2}{N(N-1)} \sum_{(i,j)\in\{1,\ldots,N\}^2, i<j} \varepsilon_{ij} \tag{4.8}$$

où :

$$\varepsilon_{ij} = \begin{cases} 0 & \text{si } (c(o_i) = c(o_j) \wedge c'(o_i) = c'(o_j)) \vee (c(o_i) \neq c(o_j) \wedge c'(o_i) \neq c'(o_j)) \\ 1 & \text{sinon} \end{cases} \tag{4.9}$$

Cette erreur considère tous les couples d'objets possibles et augmente à chaque fois que deux objets n'ont pas été classés ensemble par ANTCLASS alors qu'ils faisaient partie de la même classe à l'origine et réciproquement. Cette mesure a l'avantage de tenir compte du nombre de classes trouvées.

Algorithme 4.2: Algorithme AntClass de classification non supervisée par des fourmis et les centres mobiles.

AntClass()

(1) Soit P_0 la partition initiale formée de N classes.
(2) **pour** $t = 1$ à T_{AntClass} **faire**
(3) Initialiser la grille G à partir de la partition P_{t-1} (un tas par classe)
(4) $G' \leftarrow$Ants(G)
(5) Construire la partition P' associée à la grille G'
(6) $P_t \leftarrow$K-means(P')
(7) **finpour**
(8) **retourner** la partition $P_{T_{\text{AntClass}}}$

Paramètre	Description
A	nombre de fourmis
T	nombre de déplacements de chaque fourmi
$c(a_i)$ $\forall i \in \{1, \ldots, A\}$	capacité de transport des fourmis
$m(a_i)$ $\forall i \in \{1, \ldots, A\}$	taille de la mémoire de chaque fourmi
$v(a_i)$ $\forall i \in \{1, \ldots, A\}$	vitesse sur la grille de chaque fourmi
$p(a_i)$ $\forall i \in \{1, \ldots, A\}$	patience de chaque fourmi
k_1, k_2	paramètres de calcul des probabilités p_p et p_d

Tab. 4.1 – Paramètres de l'algorithme Ants.

On peut considérer les deux cas extrèmes suivants :
– le nombre de classes obtenu (K') est égal au nombre d'objets ($K' = N$), dans ce cas :

$$E_c(K' = N) = \sum_{i=1}^{K} \frac{|c_i|(|c_i| - 1)}{2} \tag{4.10}$$

où $|c_i|$ représente le nombre d'objets dans la classe c_i d'origine ;
– on n'a obtenu qu'une seule classe contenant tous les objets ($K' = 1$) :

$$E_c(K' = 1) = \sum_{i=1}^{K-1} \sum_{j=1}^{|c_i|} \sum_{k=i+1}^{K} |c_k| \tag{4.11}$$

4.4.2 Données artificielles

Construction des données

Afin de tester les différents algorithmes, nous avons constitué un jeu de tests formé d'ensembles de données générées en utilisant des lois uniformes ou gaussiennes. L'avantage de cette méthodologie est que l'on connait les caractéristiques de ces données et cela nous permet d'évaluer la capacité d'un algorithme à les retrouver.

Le tableau 4.2 présente pour chaque ensemble, la dimension de l'espace des objets (M), le nombre de classes créées (K), le nombre d'objets par classe ($|c_i|$) ainsi que les lois de probabilité utilisées pour chacune des K classes et chacune des M dimensions. La

Nom	M	K	$\lvert c_i \rvert$	répartition des points
ART1	2	4	100	$(\mathcal{N}(0.2, 0.2), \mathcal{N}(0.2, 0.2))$
			100	$(\mathcal{N}(0.2, 0.2), \mathcal{N}(0.8, 0.2))$
			100	$(\mathcal{N}(0.8, 0.2), \mathcal{N}(0.2, 0.2))$
			100	$(\mathcal{N}(0.8, 0.2), \mathcal{N}(0.8, 0.2))$
ART2	2	2	500	$(\mathcal{N}(0.2, 0.2), \mathcal{N}(0.2, 0.2))$
			500	$(\mathcal{N}(0.8, 0.2), \mathcal{N}(0.8, 0.2))$
ART3	2	4	500	$(\mathcal{N}(0.2, 0.2), \mathcal{N}(0.2, 0.2))$
			50	$(\mathcal{N}(0.2, 0.2), \mathcal{N}(0.8, 0.2))$
			500	$(\mathcal{N}(0.8, 0.2), \mathcal{N}(0.2, 0.2))$
			50	$(\mathcal{N}(0.8, 0.2), \mathcal{N}(0.8, 0.2))$
ART4	2	2	100	$(\mathcal{U}[-1, 1], \mathcal{U}[-10, 10])$
			100	$(\mathcal{U}[2, 3], \mathcal{U}[-10, 10])$
ART5	2	9	100	$(\mathcal{N}(0.2, 0.2), \mathcal{N}(0.2, 0.2))$
			100	$(\mathcal{N}(0.8, 0.2), \mathcal{N}(0.2, 0.2))$
			100	$(\mathcal{N}(1.4, 0.2), \mathcal{N}(0.2, 0.2))$
			$\vdots$	$\vdots$
			100	$(\mathcal{N}(1.4, 0.2), \mathcal{N}(1.4, 0.2))$
ART6	8	4	100	$(\mathcal{N}(0.2, 0.2), \ldots, \mathcal{N}(0.2, 0.2))$
			100	$(\mathcal{N}(0.2, 0.2), \mathcal{N}(0.8, 0.2), \ldots, \mathcal{N}(0.2, 0.2), \mathcal{N}(0.8, 0.2))$
			100	$(\mathcal{N}(0.8, 0.2), \mathcal{N}(0.2, 0.2), \ldots, \mathcal{N}(0.8, 0.2), \mathcal{N}(0.2, 0.2))$
			100	$(\mathcal{N}(0.8, 0.2), \ldots, \mathcal{N}(0.8, 0.2))$
ART7	2	1	100	$(\mathcal{U}[0, 1], \mathcal{U}[0, 1])$
ART8	2	1	1000	$(\mathcal{U}[0, 1], \mathcal{U}[0, 1])$

Tab. 4.2 – Paramètres des données artificielles.

base ART1 est similaire à celle utilisée dans (Lumer and Faieta, 1994). La base ART2 n'est formée que de deux classes mais chaque classe est composée d'un nombre d'objets important (500) par rapport aux autres bases. La base ART3 est similaire à ART1 sauf que les effectifs de chaque classe sont très variables. La base ART4 est composée de deux classes rectangulaires générées par une loi uniforme. La base ART5 comporte un nombre important de classes (9). Les objets de la base ART6 sont définis sur un espace comportant plus de dimensions que les autres bases (8). Enfin, les base ART6 et ART7 sont deux ensembles de points (100 et 1000 points) générés uniformément sur $[0, 1]$. Ces deux dernières bases nous permettent d'évaluer les résultats renvoyés par une méthode quand les données sont aléatoires. Les figures 4.2 et 4.3 donnent les représentations de toutes ces bases artificielles sur deux dimensions.

La suite de cette section présente les résultats obtenus par les différentes méthodes sur le jeu de tests qui vient d'être défini. Avant d'évaluer les résultats que AntClass peut obtenir, nous présentons les résultats obtenus par chacune des méthodes de façon séparée.

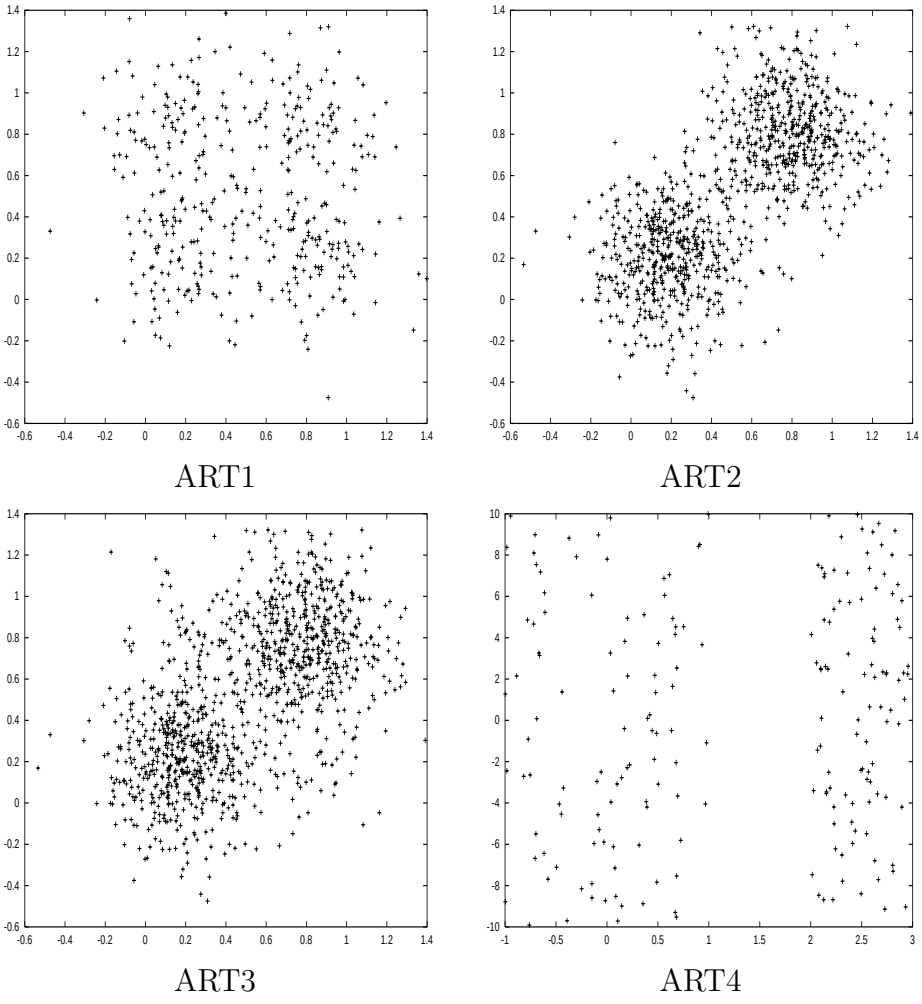


FIG. 4.2 – Bases artificielles ART1, ART2, ART3 et ART4.

Résultats de K-means seul

Le principal inconvénient de K-MEANS est qu'il faut lui fournir une partition de départ de K classes. Si cette partition n'est pas disponible, le plus simple est d'en générer une aléatoirement. C'est cette technique qui nous a permis d'obtenir les résultats donnés dans le tableau 4.3 avec deux valeurs de K (10 et 100). À titre de comparaison, nous fournissons l'erreur obtenue en générant aléatoirement des partitions de 10 et 100 classes (méthodes RAND-10 et RAND-100). Cela nous permet d'évaluer l'apport des K-MEANS sur l'erreur de classification E_c. On constate, comme prévu, que K-MEANS est inadapté pour la découverte du nombre de classes et ceci de façon plus évidente encore quand le nombre de classes de départ est éloigné du nombre de classes originelles. Du point de vue de l'erreur commise, 10-means est meilleur que 100-means, qui à son tour est légèrement meilleur que RAND-10 et RAND-100.

Si on fournit à K-MEANS une partition aléatoire comportant autant de classes que la partition d'origine, dans quelle mesure celle-ci va être découverte ? C'est à cette

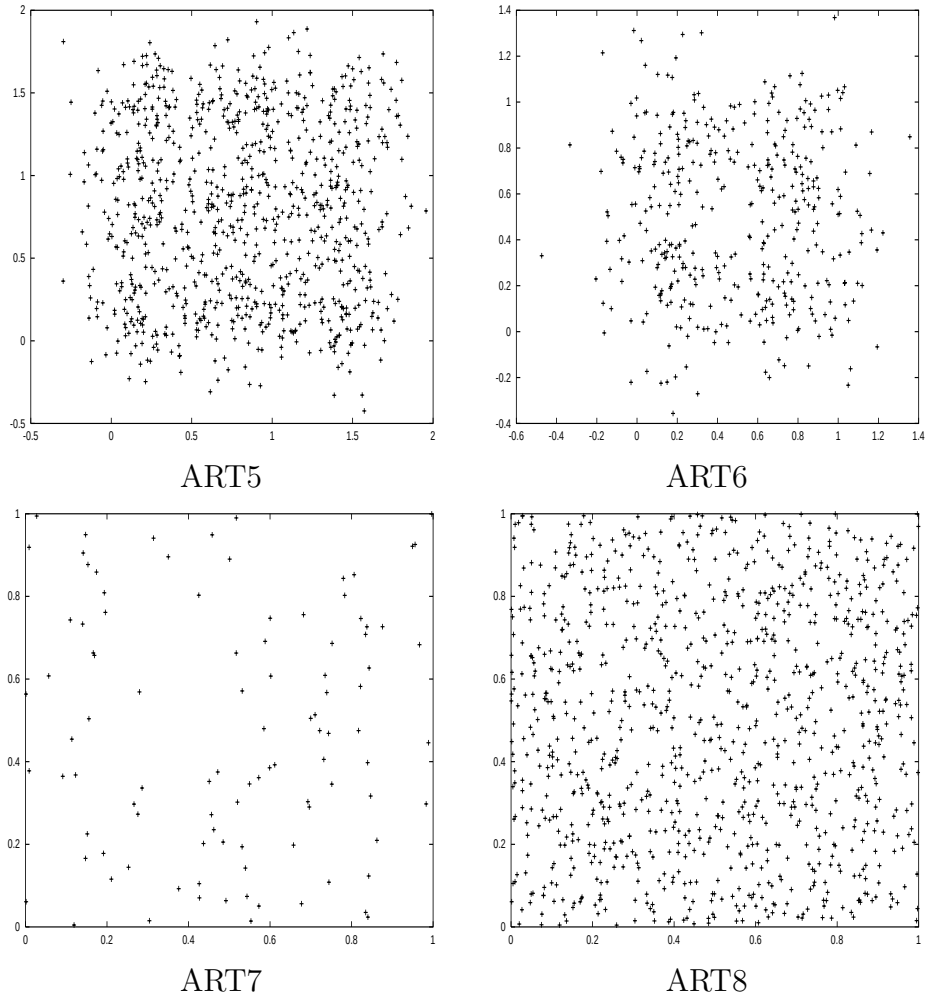


FIG. 4.3 – Bases artificielles ART5, ART6, ART7 et ART8.

question que le tableau 4.4 tente de répondre.

On peut se poser la question de la mesure dans laquelle K-MEANS peut détériorer une partition optimale (au sens de la partition d'origine). Soit l'expérience suivante : la partition de départ fournie à K-MEANS est la partition d'origine (et non plus seulement une partition comportant le bon nombre de classes comme précédemment). Le tableau 4.4 donne les résultats obtenus (K-MEANS$_t$). On remarque que K-MEANS$_t$ modifie de façon importante les partitions des données ART1 et ART3 puisque l'erreur est de 0.11 et 0.23 respectivement. Les résultats sur ART7 et ART8 ne sont pas présentés puisqu'ils n'ont aucun intérêt car tous les objets font partie de la même classe et par conséquent, K-MEANS ne fait rien.

La comparaison de résultats de K-MEANS et K-MEANS$_t$ (tableau 4.4) montre que les résultats obtenus par l'algorithme K-MEANS, en l'initialisant avec la partition d'origine ou une partition aléatoire comportant le bon nombre de classes, sont très similaires.

Base	RAND-10	RAND-100	10-MEANS		100-MEANS	
	E_c $[\sigma_{E_c}]$	E_c $[\sigma_{E_c}]$	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$
ART1	0.30 [0.00]	0.25 [0.00]	8.58 [0.98]	0.18 [0.01]	67.70 [3.65]	0.23 [0.00]
ART2	0.50 [0.00]	0.50 [0.00]	8.52 [0.96]	0.38 [0.01]	62.52 [3.31]	0.48 [0.00]
ART3	0.43 [0.00]	0.42 [0.00]	8.28 [0.96]	0.31 [0.01]	62.08 [3.39]	0.40 [0.00]
ART4	0.50 [0.00]	0.50 [0.00]	6.38 [0.75]	0.32 [0.02]	54.58 [3.07]	0.48 [0.00]
ART5	0.19 [0.00]	0.12 [0.00]	8.82 [0.91]	0.08 [0.01]	72.90 [3.01]	0.10 [0.00]
ART6	0.30 [0.00]	0.25 [0.00]	8.46 [1.08]	0.10 [0.02]	59.04 [3.55]	0.23 [0.00]
ART7	0.90 [0.00]	0.99 [0.00]	7.76 [1.03]	0.87 [0.02]	50.76 [3.98]	0.98 [0.00]
ART8	0.90 [0.00]	0.99 [0.00]	8.78 [0.83]	0.88 [0.01]	66.48 [3.44]	0.98 [0.00]

TAB. 4.3 – Résultats obtenus par 10-MEANS et 100-MEANS sur les données artificielles. RAND-10 et RAND-100 correspondent à une méthode aléatoire générant une partition de 10 et 100 classes. Sur 50 essais, K' représente le nombre moyen de classes obtenues, E_c l'erreur de classification moyenne, $\sigma_{K'}$ et σ_{E_c} représentent les écarts type respectifs.

Base	K-MEANS		K-MEANS$_t$	
	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$	K'	E_c
ART1	3.96 [0.20]	0.12 [0.02]	4	0.11
ART2	2.00 [0.00]	0.04 [0.00]	2	0.04
ART3	3.90 [0.30]	0.22 [0.02]	4	0.23
ART4	2.00 [0.00]	0.01 [0.06]	2	0.00
ART5	7.98 [0.76]	0.09 [0.01]	9	0.07
ART6	3.98 [0.14]	0.01 [0.04]	4	0.00

TAB. 4.4 – Résultats obtenus par K-MEANS sur les données artificielles avec le bon nombre de classes au départ et avec la partition d'origine comme partition de départ (K-MEANS$_t$). Sur 50 essais (pour K-MEANS), K' représente le nombre moyen de classes obtenues, E_c l'erreur de classification moyenne, $\sigma_{K'}$ et σ_{E_c} représentent les écarts type respectifs.

Résultats des fourmis seules : Ants

La figure 4.4 donne le nombre de classes obtenu en moyenne sur 50 essais par les fourmis seules sur les huit ensembles de données artificielles. En ordonnée figurent différentes combinaisons du couple de paramètres (k_1, k_2) : pour chacun, les valeurs 0.1, 0.5, 1.0 et 2.0 sont testées et sont numérotées de 1 à 4 sur la figure. On utilise 20 fourmis de capacité 1, sans mémoire, de patience égale à la largeur de la grille et 10 000 déplacements leur sont donnés. Pour l'ensemble des données artificielles considéré, il est clair que le couple $(k_1, k_2) = (0.1, 0.1)$ fournit le nombre de classes le plus faible ce qui semble être la meilleure initialisation envisageable pour les K-MEANS. Le tableau 4.5 donne les valeurs numériques pour ce jeu de paramètres. Les résultats semblent indiquer que le nombre de classes obtenu dépend du nombre d'objets dans la base : plus la base est importante, plus le nombre de classes obtenu par ANTS est grand (et donc éloigné

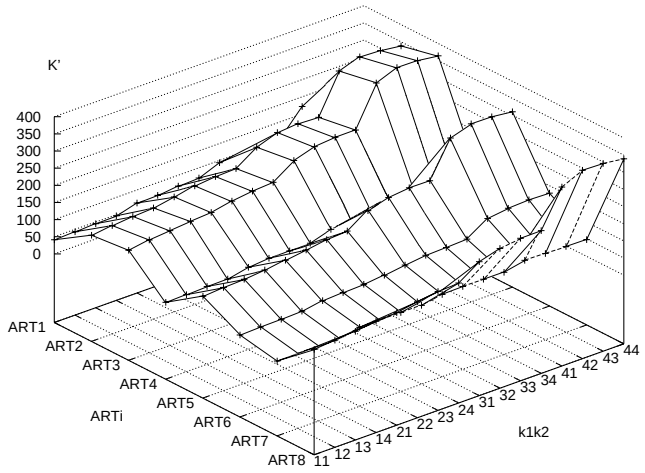


FIG. 4.4 – Moyennes du nombre de classes obtenu par ANTS pour les données artificielles. k1k2 = 11 signifie que $(k_1, k_2) = (0.1, 0.1)$, k1k2 = 12 signifie que $(k_1, k_2) = (0.1, 0.5)$, ..., k1k2 = 44 signifie que $(k_1, k_2) = (2.0, 2.0)$.

du nombre correct de classes). La figure 4.5 donne pour la base ART1 l'évolution du nombre de tas au cours des 10 000 itérations. Le nombre de tas décroit rapidement pendant les 1 000 premières itérations pour ensuite se stabiliser autour de 50 tas. Que se passe-t-il pendant ce temps ? La figure 4.6 donne l'évolution de la distance moyenne des objets par rapport au centre du tas auquel ils sont rattachés. La distance moyenne est nulle pour la première itération puisque il y a autant de tas que d'objets. Un maximum de 0.15 est atteint aux environs de 1 000 itérations pour ensuite décroitre lentement vers 0.08. Ceci nous montre que lorsque le nombre de tas ne varie presque plus, les fourmis améliorent la classification.

Résultats de AntClass avec $T_{\text{AntClass}} = 1$

Le tableau 4.6 donne les résultats que l'on obtient quand K-MEANS est appliqué sur la partition trouvée par ANTS. La première constatation est que le nombre de classes n'a pas été amélioré par rapport aux fourmis seules, ce qui est normal. Quand à l'erreur de classification, l'amélioration est très légère.

Résultats de AntClass avec $T_{\text{AntClass}} = 2$

Dans ce cas, deux itérations du couple ANTS+K-MEANS sont effectuées. Le tableau 4.7 présente les résultats obtenus par ANTCLASS avec deux paramétrages différents (ANTCLASS1 et ANTCLASS2), les paramètres en commun étant les suivants :

- $T_{\text{AntClass}} = 2$
- $A = 20$ (nombre de fourmis)
- pour la première itération de ANTS, $\forall i \in \{1, \ldots, A\}$:
  - $m(a_i) = 0$ (taille de la mémoire)

Base	ANTS, $k_1 = 0.1$, $k_2 = 0.1$	
	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$
ART1	40.82 [3.71]	0.25 [0.00]
ART2	109.26 [5.84]	0.49 [0.00]
ART3	120.68 [6.62]	0.41 [0.00]
ART4	23.78 [3.43]	0.41 [0.02]
ART5	96.68 [6.09]	0.12 [0.00]
ART6	38.82 [4.92]	0.24 [0.01]
ART7	17.78 [3.88]	0.76 [0.11]
ART8	106.98 [5.61]	0.99 [0.00]

TAB. 4.5 – Résultats obtenus par les fourmis seules sur les données artificielles. Sur 50 essais, K' représente le nombre moyen de classes obtenues, E_c l'erreur de classification moyenne, $\sigma_{K'}$ et σ_{E_c} représentent les écarts type respectifs.

Base	ANTS+K-MEANS	
	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$
ART1	42.00 [4.35]	0.23 [0.00]
ART2	107.34 [5.70]	0.49 [0.00]
ART3	118.26 [5.86]	0.41 [0.00]
ART4	23.68 [4.12]	0.45 [0.01]
ART5	95.84 [5.86]	0.10 [0.00]
ART6	38.02 [4.14]	0.22 [0.00]
ART7	18.70 [3.89]	0.93 [0.02]
ART8	105.80 [5.76]	0.99 [0.00]

TAB. 4.6 – Résultats obtenus par ANTCLASS avec $T_{\text{AntClass}} = 1$ sur les données artificielles. Sur 50 essais, K' représente le nombre moyen de classes obtenues, E_c l'erreur de classification moyenne, $\sigma_{K'}$ et σ_{E_c} représentent les écarts type respectifs.

  - $c(a_i) = 1$ (capacité de transport)
  - $v(a_i) \in \{1, 2, 3\}$ (vitesse)
  - $p(a_i) = L$ (patience = largeur de la grille)
- pour la deuxième itération de ANTS, $\forall i \in \{1, \ldots, A\}$:
  - $m(a_i) = 3$
  - $c(a_i) = \infty$
  - $v(a_i) \in \{1, 2, 3\}$
  - $p(a_i) = L$

Les paramètres qui différent sont indiqués dans le tableau 4.7 (T ,k_1 et k_2). ANTCLASS2 est plus apte à fournir un nombre de classes proche de celui de la partition d'origine. Pour ANTCLASS2, les fourmis de capacité infinie disposent de 1000 itérations supplémentaires par rapport à ANTCLASS1. De plus, pour la deuxième itération du couple ANTS+K-MEANS les valeurs de k_1 et k_2 (notées k_{12} et k_{22}) diminuent les probabilités de déposer et de ramasser des tas.

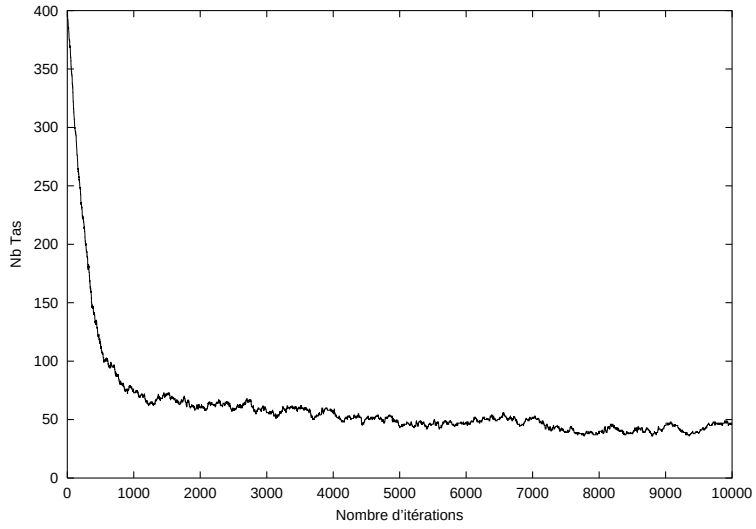


FIG. 4.5 – Evolution du nombre de tas manipulés par ANTS pour la base ART1.

La courbe de la figure 4.7 donne l'évolution du nombre de tas au cours des itérations pour la base ART1 et le paramétrage ANTCLASS2. On constate la stagnation du nombre de tas pendant la première itération de ANTCLASS puis la forte diminution de celui-ci pendant la deuxième itération.

La courbe de la figure 4.8 donne l'évolution de la distance moyenne des objets par rapport au centre du tas auquel ils sont rattachés pour la base ART1 et le paramétrage ANTCLASS2. Cette courbe nous confirme que la capacité infinie des fourmis ne fait pas décroître la distance moyenne : les tas sont agglomérés entre eux et le nouveau centre de gravité qui en résulte est le plus souvent plus éloigné des objets composant le tas qu'auparavant.

L'image de la figure 4.9 donne la fréquence d'association des objets de la base ART1 entre eux. On retrouve nettement la séparation des quatre classes de la partition d'origine (les objets sont fournis à ANTCLASS dans leur ordre d'appartenance aux classes d'origine).

Le nombre de classes obtenu par ANTCLASS semble fortement lié au nombre d'objets dans la base. Par exemple, la base ART2, ne comportant que deux classes, est composée d'un milliers d'objets. ANTCLASS2 trouve en moyenne 12.32 classes alors qu'avec le même jeu de paramètres, 4.22 classes sont trouvées en moyenne pour la base ART1 qui se compose de 400 objets et comporte 4 classes. La courbe 4.10 donne l'évolution du nombre de classes quand on augmente le nombre d'itérations T_2 pour la base ART2.

4.4.3 Données réelles

Le tableau 4.8 présente les bases de données issues de (Blake and Merz, 1998) que nous avons utilisées. Les paramètres utilisés pour ANTCLASS sont les suivants :

– $T_{\text{AntClass}} = 2$

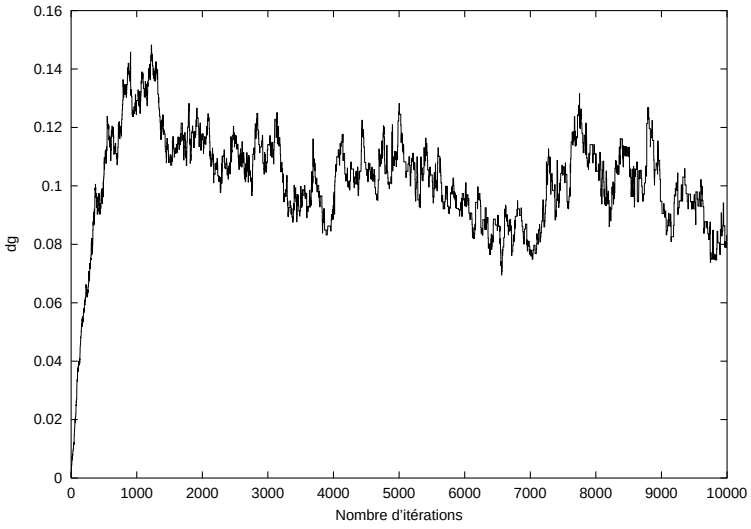


FIG. 4.6 – Evolution de la distance moyenne (dg) des objets par rapport au centre du tas auquel ils sont rattachés pour l'algorithme ANTS et la base ART1.

- pour la première itération de ANTS, :
  - $A = 40$ (nombre de fourmis)
  - $T = 10\,000$ (Nombre d'itérations)
  - $m(a_i) = 0 \quad \forall i \in \{1, \ldots, A\}$ (taille de la mémoire)
  - $c(a_i) = 1 \quad \forall i \in \{1, \ldots, A\}$ (capacité de transport)
  - $v(a_i) \in \{1, 2, 3\} \quad \forall i \in \{1, \ldots, A\}$ (vitesse)
  - $p(a_i) = 1\,000 \quad \forall i \in \{1, \ldots, A\}$ (patience = largeur de la grille)
  - $k_1 = 1$
  - $k_2 = 1$
- pour la deuxième itération de ANTS :
  - $A = 20$
  - $T = 10\,000$
  - $m(a_i) = 5 \quad \forall i \in \{1, \ldots, A\}$
  - $c(a_i) = \infty \quad \forall i \in \{1, \ldots, A\}$
  - $v(a_i) = 1 \quad \forall i \in \{1, \ldots, A\}$
  - $p(a_i) = 1\,000 \quad \forall i \in \{1, \ldots, A\}$
  - $k_1 = 2$
  - $k_2 = 2$

Les Résultats obtenus sur ces bases sont donnés dans le tableau 4.9. Pour la base IRIS, ANTCLASS obtient en moyenne légèrement trop de classes mais cependant, le plus fréquemment (18 essais sur 50), trois classes sont obtenues, totalisant une erreur de 0.15. Pour la base WINE, comportant à l'origine trois classes, ANTCLASS obtient très souvent trop de classes. Ceci peut être expliqué par des objets se retrouvant seuls et formant donc une classe suplémentaire. Mais ceci ne suffit pas à expliquer une erreur de classification importante. L'analyse des partitions obtenues nous montre en effet que les objets sont souvent mélangés dans une même classe. La base GLASS contient 7 classes

Base	AntClass1 $T_1 = 10\,000, k_{11} = 0.1, k_{21} = 0.1$ $T_2 = 2\,000, k_{12} = 1, k_{22} = 1$		AntClass2 $T_1 = 10\,000, k_{11} = 0.1, k_{21} = 0.1$ $T_2 = 3\,000, k_{12} = 2, k_{22} = 2$	
	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$
ART1	7.46 [1.58]	0.17 [0.02]	4.22 [1.15]	0.15 [0.05]
ART2	24.92 [4.16]	0.45 [0.01]	12.32 [2.01]	0.41 [0.01]
ART3	31.54 [3.81]	0.39 [0.00]	14.66 [2.68]	0.35 [0.01]
ART4	3.12 [1.03]	0.14 [0.11]	1.68 [0.84]	0.29 [0.23]
ART5	24.16 [3.27]	0.09 [0.00]	11.36 [1.94]	0.08 [0.01]
ART6	6.90 [1.80]	0.08 [0.04]	3.74 [1.38]	0.11 [0.13]
ART7	1.80 [0.89]	0.31 [0.28]	1.38 [0.60]	0.17 [0.24]
ART8	27.38 [3.65]	0.96 [0.01]	13.06 [2.18]	0.92 [0.01]

Tab. 4.7 – Résultats obtenus par AntClass sur les données artificielles. Sur 50 essais, K' représente le nombre moyen de classes obtenues, E_c l'erreur de classification moyenne, $\sigma_{K'}$ et σ_{E_c} représentent les écarts type respectifs.

Nom	M	K	N
IRIS	4	3	150
WINE	12	3	178
GLASS	9	7	214
PIMA	8	2	798
SOYBEAN	35	4	47
THYROID	5	3	215

Tab. 4.8 – Bases de données réelles utilisées (Blake and Merz, 1998).

mais en fait la classe 4 est vide ce qui signifie que AntClass est assez proche du nombre de classes effectivement présent. C'est en effet 6 classes qui sont le plus fréquemment trouvées. Le nombre de classes trouvées par AntClass pour la base PIMA est trop élevé, ceci est probablement dû à la taille de base qui est la plus importante du jeux de tests utilisé. Nous avions constaté ce même phénomène pour les données artificielles, ce qui signifie qu'il faudrait donner plus de temps au fourmis pour voir le nombre de classes diminuer. Pour la base SOYBEAN il semble que le phénomène inverse (il n'y a que 47 objets) soit à l'origine du faible nombre de classes découvertes par AntClass. En ce qui concerne la base THYROID, on peut constater que AntClass n'obtient pas le nombre de classes voulu (c'est-à-dire trois). Par contre l'erreur de classification est plus faible. Pour illustrer cet aspect, la figure 4.11 donne l'erreur moyenne trouvée en fonction du nombre de classe trouvé.

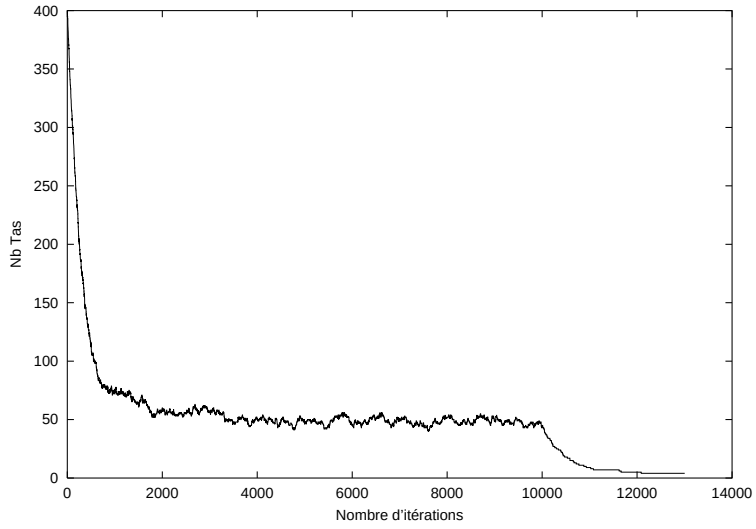


FIG. 4.7 – Evolution du nombre de tas manipulés par ANTCLASS pour la base ART1.

Base	10-MEANS		K-MEANS$_t$		ANTCLASS	
	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$	K'	E_c	K' $[\sigma_{K'}]$	E_c $[\sigma_{E_c}]$
IRIS	7.12 [1.11]	0.18 [0.03]	3.00	0.13	3.52 [1.39]	0.19 [0.08]
WINE	9.64 [0.52]	0.27 [0.01]	3.00	0.28	6.46 [2.10]	0.51 [0.11]
GLASS	9.44 [0.70]	0.29 [0.02]	6.00	0.32	5.60 [2.01]	0.40 [0.06]
PIMA	9.90 [0.36]	0.50 [0.01]	2.00	0.44	6.10 [1.84]	0.47 [0.02]
SOYBEAN	8.82 [0.97]	0.13 [0.02]	4.00	0.00	1.60 [0.49]	0.54 [0.17]
THYROID	9.56 [0.57]	0.42 [0.02]	3.00	0.18	5.84 [1.33]	0.22 [0.09]

TAB. 4.9 – Résultats sur les bases de données réelles.

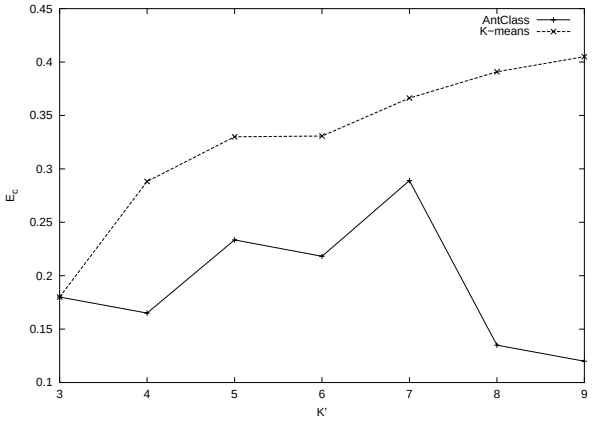


FIG. 4.11 – Erreurs obtenues (E_c) par ANTCLASS et K-MEANS en fonction du nombre de classes trouvées (K').

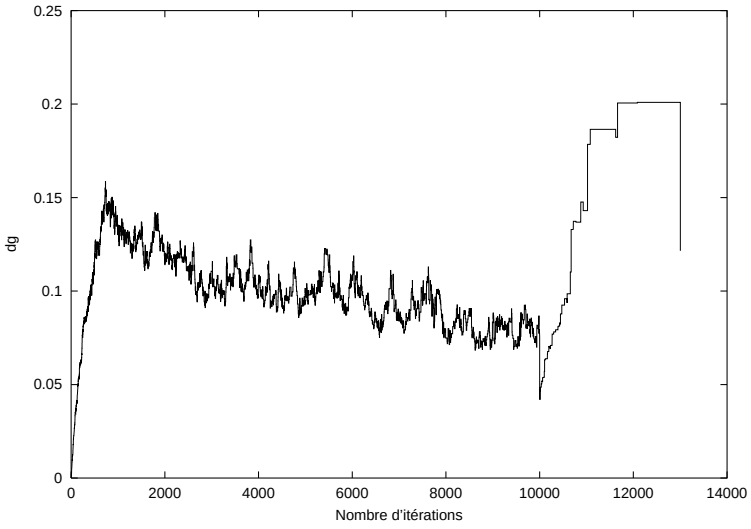


FIG. 4.8 – Evolution de la distance moyenne des objets par rapport au centre du tas auquel ils sont rattachés pour l'algorithme ANTCLASS et la base ART1.

On constate que dans ce cas, ANTCLASS est nettement plus efficace pour initialiser K-MEANS que l'initialisation aléatoire pratiquée quand K-MEANS est utilisé seul. L'intérêt de la coopération entre les deux algorithmes est ici bien illustrée.

Base	10-MEANS	ANTCLASS
IRIS	0.05	1.63
WINE	0.23	3.51
GLASS	0.22	3.12
PIMA	1.69	12.87
SOYBEAN	0.03	0.68
THYROID	0.15	2.37

TAB. 4.10 – Temps d'exécution de 10-MEANS et ANTCLASS en secondes (Les deux algorithmes ont été programmés en C et les tests ont été réalisés sur une machine équipé d'un processeur Celeron à 400 MHz avec Windows NT 4.0 comme système d'exploitation).

Du point de vue du temps d'exécution, le tableau 4.10 donne les temps de calcul pour 10-MEANS et ANTCLASS sur les différentes bases. Les durées d'exécution moyennes de ANTCLASS sont évidemment supérieures à ceux de 10-MEANS puisque ANTCLASS incorpore deux itérations de K-MEANS.

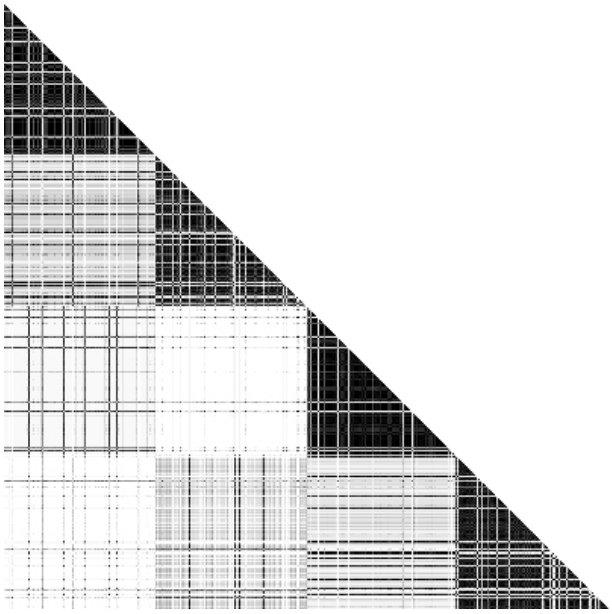


FIG. 4.9 – Fréquence d'association des objets de la base ART1 entre eux. Chaque point de coordonnée (i, j) sur l'image a un niveau de gris proportionnel au nombre d'exécutions de ANTCLASS ayant classé ensemble les objets o_i et o_j. Un pixel noir (valeur 0) signifie que les deux objets ont toujours été classés ensemble.

4.5 Discussion

Un certain nombre d'approches issues de l'apprentissage peuvent être comparées à ANTCLASS. TORRE (Torre, 1999) a par exemple développé un algorithme où les objets sont déplacés d'un groupe à l'autre sur un anneau circulaire et restent dans un groupe quand la distance maximale entre chaque objet du groupe et le nouvel arrivant n'excède pas une valeur donnée. Les objets sont appelés des Amis et un groupe d'Amis homogène atteint le stade de Vraizamis. Ce système converge vers une position d'équilibre où les groupes ne varient plus. Le point commun avec ANTCLASS est que le choix d'un objet à déplacer est aléatoire mais d'autant plus probable que l'objet est seul dans son groupe, de plus, au départ il y a autant de groupes que d'objets. Par contre, pour ANTCLASS la rencontre des objets et des groupes est dépendante du déplacement des fourmis et de la répartition des groupes sur la grille alors que pour les Amis les rencontres se font à chaque déplacement d'un Ami sur l'anneau. L'inconvénient des Amis est qu'il faut leur donner une distance maximale, servant de seuil à l'agglomération d'un objet à un groupe, et qui sert donc de paramètre pour le nombre de groupes à découvrir. Pour ANTCLASS c'est plutôt le nombre d'itérations qui remplit ce rôle : plus on est patient, moins on obtient de groupes. On peut retrouver le principe des Vraizamis dans des travaux antérieurs, menés par KUNTZ et SNYERS (Kuntz and Snyers, 1994), où le problème du partitionnement de graphes était traité. Plusieurs espèces d'animats évoluent sur les sommets d'un graphe et chaque agent possède un paramètre de satisfaction dépendant de l'espèce majoritairement présente

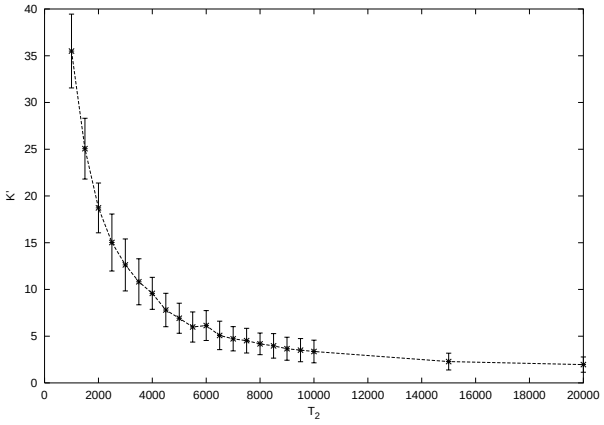


FIG. 4.10 – Variation du nombre de classes en fonction de T_2 pour la base ART2. Les écarts type sont représentés par des barres verticales.

sur le sommet l'accueillant. L'approche est beaucoup plus orientée vers la vie artificielle puisque les animats peuvent se reproduire s'ils sont satisfaits et mourir s'ils ont atteint un âge limite. La partition du graphe est obtenue en considérant les sommets ayant la même espèce majoritaire.

L'utilisation simultanée de plusieurs méthodes de classification a fait l'objet de différents travaux. Dans (Wemmert et al., 1999) les auteurs décrivent un système de collaboration entre plusieurs méthodes de classification non supervisées. Chaque méthode est utilisée sur l'ensemble des données et les résultats sont combinés en résolvant les conflits rencontrés entre les partitions deux à deux. On peut considérer que AntClass fait aussi collaborer deux méthodes : la classification par population de fourmis et les centres mobiles. L'approche est cependant séquentielle ce qui évite d'avoir des conflits à résoudre.

4.6 Évolutions, perspectives

L'approche que nous avons adoptée a été d'améliorer un algorithme existant. La principale évolution que nous pouvons proposer est d'abandonner la grille comme support des objets à classer. En effet, de nombreux paramètres y sont rattachés et il n'est pas toujours évident d'en mesurer l'impact sur les résultats :

- la taille de la grille, même si elle est déterminée automatiquement, peut par exemple ralentir l'agglomération des objets : quand il ne reste que très peu de tas, les fourmis ne font plus beaucoup de rencontres. Il serait sûrement plus efficace de modéliser le passage d'une fourmi d'un tas à l'autre plutôt que de simuler le déplacement des fourmis ou encore de redimensionner la grille à chaque itération de Ants ;
- la vitesse des fourmis ainsi que leur capacité de mémorisation sont des idées qui ont été reprises de l'algorithme LF. Pour Ants, l'objectif de ces deux paramètres est d'atténuer la difficulté des fourmis à trouver les tas d'objets (contrairement à

LF où la grille ne pouvait accueillir qu'un seul objet par case). Il reste à évaluer dans quelles mesures les valeurs de ces paramètres influencent les résultats.

Le tri collectif du couvain n'est pas la seule source d'inspiration que l'on peut retirer des fourmis pour résoudre des problèmes de classification. La recherche de nourriture, par exemple, peut être considérée comme une compétition entre plusieurs colonies pour le contrôle de certaines zones de chasse. On peut imaginer une méthode de classification supervisée où l'on fixe au départ le nombre de colonies/classes, puis en simulant l'activité de fourragement de chaque colonie, les objets à classer représentant des proies, on peut s'attendre à une répartition spatiale de chaque colonie comme réponse à la compétition.

Les mécanismes de reconnaissance inter-individuels et la constitution de l'odeur coloniale peuvent aussi s'apparenter à des mécanismes de classification. Différencier ses congénères de ses ennemis tout en modifiant constamment son propre schéma d'identification peut s'apparenter à la détermination des objets qui font partie d'une même classe des autres objets.

4.7 Conclusion

Nous avons montré dans ce chapitre que les fourmis sont une source d'inspiration pertinente pour résoudre des problèmes de classification non supervisée. Les objets d'une base de données numériques sont manipulés par les agents-fourmis d'une façon analogue au tri du couvain chez les fourmis réelles. Cela nous permet notamment de proposer une heuristique déterminant le nombre de classes automatiquement.

Les bases de données que l'on peut avoir besoin de partitionner ne sont pas toujours composées d'objets ayant des attributs uniquement numériques. Les futurs travaux sur l'utilisation des fourmis en classification devraient traiter le cas de données symboliques.

L'algorithme AntClass a donné lieu à l'encadrement d'un projet de fin d'études à l'E3i (Steinberg, 1998). Un certain nombre de communications ou publications ont aussi ponctué les différentes étapes de cette étude (Steinberg et al., 1998b; Steinberg et al., 1998a; Monmarché et al., 1999b; Monmarché et al., 1999c; Monmarché et al., 1999b; Monmarché et al., 1999a; Monmarché, 1999).

Chapitre 5

Le problème d'optimisation

Nous introduisons dans ce chapitre le problème de l'optimisation globale qui va nous préoccuper dans la suite de ce document. Une rapide présentation du problème et des méthodes de résolution existantes est proposée.

5.1 Introduction

L'optimisation est un sujet central en informatique. De nombreux problèmes peuvent être formulés sous la forme d'un problème d'optimisation. Les problèmes d'apprentissage, par exemple l'apprentissage des chaînes de Markov cachées ou des réseaux de neurones, nécessitent une phase d'optimisation. Le problème de classification abordé au chapitre précédent peut aussi se formuler comme un problème d'optimisation : minimiser la distance intra-classe et maximiser la distance inter-classe.

Ce chapitre de transition présente tout d'abord le type de problème qui sera abordé dans la suite de ce document. Nous nous intéresserons ensuite aux méthodes de résolution existantes et nous en développerons quelques unes qui serviront de référence aux études présentées dans les chapitres suivants.

5.2 Définition du problème

Le problème peut se formuler de la façon suivante :

Définition 5.1 *Problème d'optimisation globale. Étant donnée une fonction f définie sur un espace de recherche $\mathcal{S}$ (l'ensemble des solutions) et à valeurs dans $\mathbb{R}$, on cherche la solution s^* telle que*

$$f(s^*) = \min_{s \in \mathcal{S}} \{f(s)\} \tag{5.1}$$

s^* est la solution au problème d'optimisation globale et $f(s^*)$ (noté f^*) est alors appelé optimum global de f sur $\mathcal{S}$.

Remarques :

- f est appelée la fonction objectif ;
- s^* n'est pas obligatoirement unique, notons alors par $\mathcal{S}^*$ l'ensemble des solutions au problème d'optimisation globale. Dans la plupart des cas la découverte d'un seul élément de $\mathcal{S}^*$ suffit pour répondre au problème ;
- par la suite, par optimisation, nous supposerons qu'il s'agit en fait de minimisation. En effet, maximiser $f(s)$ revient à minimiser $-f(s)$ ce qui permet de ne perdre aucune généralité ;
- des variantes peuvent apparaître. Par exemple si la fonction f est variable dans le temps, on parlera d'optimisation dynamique, s'il existe des contraintes (d'égalité ou d'inégalité) sur les solutions de $\mathcal{S}$ on parlera d'optimisation avec contraintes, enfin si le problème consiste à optimiser simultanément plusieurs fonctions objectifs, on parlera d'optimisation multicritère.

5.3 Méthodes de résolution exactes

Les méthodes de résolution exactes sont des méthodes qui garantissent l'optimalité de leur réponse en un temps fini. On parle de méthodes exactes par opposition aux méthodes approchées présentées dans la section suivante. L'efficacité de ces méthodes est malheureusement dépendante d'un certain nombre d'hypothèses concernant le problème. Par exemple dans le cas de fonctions numériques à variables réelles, les conditions d'optimalité que l'on peut nécessiter sont que la fonction soit continuement différentiable. Les méthodes exactes s'appliquent en particulier quand le problème est convexe. Dans le cas des problèmes à variables discrètes, les procédure par séparation et évaluation (PSE) assurent de trouver un optimum global mais les temps de calcul peuvent être rédhibitoires. Parmis les méthodes exactes, on peut citer les suivantes :

- la programmation linéaire ;
- la programmation quadratique ;
- la programmation dynamique ;
- les méthodes de descente (gradient, gradient conjugé, Newton, ...) ;
- ...

La littérature dans ce domaine est abondante aussi nous renvoyons le lecteur à une introduction aux méthodes d'optimisation classiques par exemple dans (Culioli, 1994).

5.4 Méthodes de résolution heuristiques et biomimétiques

5.4.1 Motivations

Les difficultés que peuvent rencontrer les méthodes exactes peuvent être de plusieurs types (Schwefel, 1998) :

- non différentiabilité de la fonction objectif ;
- discontinuités (de la fonction objectif, de l'espace de recherche) ;

- bruit, erreurs d'arrondi ;
- multimodalité (espace non convexe) ;
- optima non stationnaires ;
- variation du nombre de variables ;
- l'espace de recherche est discret/mixte ;
- problèmes combinatoires ;
- nombre important de variables et de contraintes (grands problèmes) ;
- plusieurs critères non coopératifs.

C'est à cause de ce genre de déconvenues qu'un certain nombre de méthodes classiques ne peuvent être utilisées. Les méthodes analytiques deviennent en effet inopérantes, l'énumération complète est trop laborieuse ou encore les méthodes de gradient deviennent inutiles. C'est alors qu'il faut se diriger vers des méthodes en général plus récentes et non exactes, des heuristiques.

Le principal avantage des méthodes de résolution heuristiques vient de leur capacité à traiter un problème en ne possédant qu'un minimum d'informations sur celui-ci. En contrepartie, la qualité de la solution proposée n'est jamais prouvée. On ne peut donc jamais savoir si le minimum global a été trouvé. Cependant, du point de vue de l'aide à la décision, la certitude d'obtenir le minimum global n'est pas toujours primordiale. La description d'une méthode heuristique peut être assez simple et se transpose alors facilement à de nouveaux types de problèmes. On peut parler alors de méta-heuristique.

Tous ces principes généraux ont évidemment l'inconvénient de leur généralité : la plupart des heuristiques qui forcent le respect s'appuyent sur des informations spécifiques du problème qu'elles traitent. Dans ce cas, la caractéristique de « portabilité » d'une méta-heuristique s'effrite. Le seul enseignement que l'on peut tirer est que pour bien résoudre un problème, il faut parfaitement le connaître et exploiter toutes les connaissances du domaine disponibles.

5.4.2 Les méthodes aléatoires

Le rôle de l'aléatoire est central dans les heuristiques récentes. La figure 5.1 schématise les dépendances entre les méthodes aléatoires (appelées aussi stochastiques) (Byrne, 1997). Les méthodes de recherche aléatoires génèrent aléatoirement des points de $\mathcal{S}$ (d'où leur nom). Au niveau le plus simple, une méthode de recherche aléatoire pure renvoie le meilleur point trouvé en évaluant un ensemble de points générés aléatoirement. L'efficacité de cette recherche est dépendante de la densité de points générés dans l'espace de recherche. Les recherches aléatoires augmentées sont utilisées pour initialiser une recherche locale qui peut mettre à profit des propriétés de convexité locale de f. Si la recherche aléatoire est utilisée pour générer un seul point de départ pour la recherche locale, on parle de démarrage unique. Par contre si la recherche locale est appliquée sur plusieurs points générés aléatoirement, on parle de redémarrages multiples. Les recherches aléatoires adaptatives permettent d'utiliser l'information que constitue l'échantillonnage aléatoire de $\mathcal{S}$ pour le rendre non uniforme (c'est pour cela qu'on les dit adaptatives). Le partitionnement (*Clustering*) fait partie de cette classe de recherche aléatoire. Il s'agit de construire des groupes de points à partir d'un échantillonnage de $\mathcal{S}$ suivant une distance, par exemple euclidienne, et un seuil donné. Les groupes formés

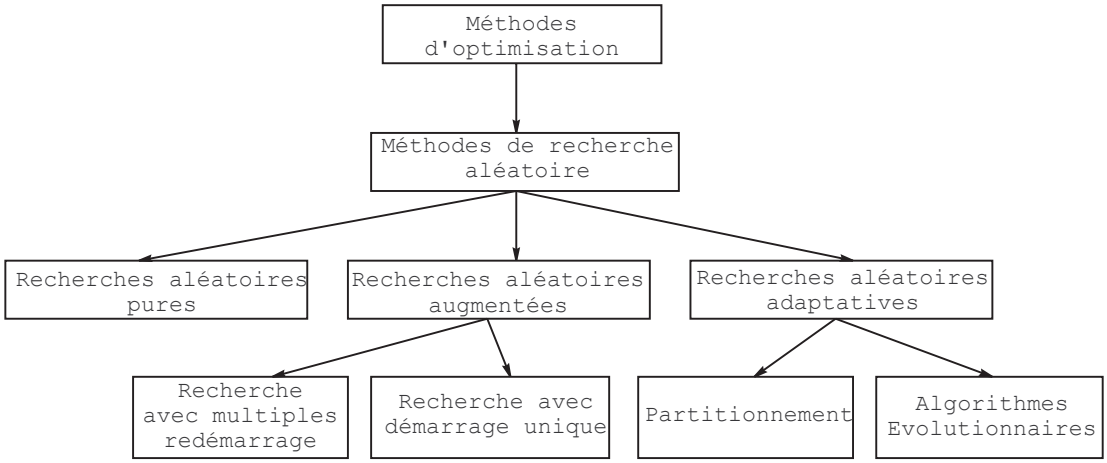


FIG. 5.1 – Les méthodes de recherche aléatoires (d'aprés (Byrne, 1997)).

peuvent servir à initialiser une recherche locale démarrant sur le meilleur point de chaque groupe. L'avantage est qu'il n'y a pas de recouvrement (grâce au seuil) comme cela peut être le cas pour le redémarrage multiple. Enfin, les algorithmes d'évolution rentrent dans cette classe des méthodes de recherche aléatoire adaptatives. Nous y reviendrons plus en détail par la suite.

5.4.3 Les méthodes itératives

La classification des méthodes stochastiques précédemment exposée ne met pas en valeur l'aspect itératif de certaines heuristiques. On parle de méthode itérative quand la solution au problème est donnée à la suite de plusieurs cycles de l'heuristique (contrairement à la recherche aléatoire pure, par exemple). L'illustration la plus simple des méthodes itératives est ce que l'on appelle l'ascension locale[1] (*Hill-Climbing* : HC). L'algorithme 5.1 en donne la structure générale.

[1]On parle d'ascension locale lorsque l'on traite un problème de maximisation et de descente locale sinon.

Algorithme 5.1: Algorithme d'ascension locale
HillClimbing()

```
(1) Générer et évaluer une solution initiale s
(2) tantque La condition d'arrêt n'est pas vérifiée faire
(3)    Modifier s pour obtenir s' et évaluer s'
(4)    si s' est meilleure que s alors
(5)       Remplacer s par s'
(6)    finsi
(7) fintantque
(8) retourner s
```

On peut remarquer que dans la formulation qui est donnée, il n'est pas mentionné que l'aléatoire est mis en jeu (pour la génération de la solution initiale ou sa modification). L'algorithme 5.2 décrit la version stochastique (*Random Hill-Climbing* : RHC). Il faut alors définir un pas de recherche qui détermine la taille du voisinage pouvant être exploré.

Algorithme 5.2: Algorithme d'ascension locale stochastique
RandomHillClimbing()

```
(1) Générer aléatoirement une solution initiale s et l'évaluer
(2) tantque La condition d'arrêt n'est pas vérifiée faire
(3)    Modifier aléatoirement s pour obtenir s' et évaluer s'
(4)    si s' est meilleure que s alors
(5)       Remplacer s par s'
(6)    finsi
(7) fintantque
(8) retourner s
```

Quand l'ascention locale stochastique est redémarrée plusieurs fois de suite on la note MSRHC (*Multiple Start Random Hill-Climbing*). La décision de redémarrer la recherche peut être fonction de la progression ou périodique.

Des heuristiques plus évoluées telles que le recuit simulé (*Simulated Annealing* : SA) (Kirkpatrick et al., 1983) ou la recherche tabou (*Tabu Search* : TS) (Glover, 1989; Glover, 1990) font partie de ces heuristiques itératives. Pour la première, le choix de remplacer s par s' peut aussi se faire si s est meilleure que s' suivant une probabilité dépendant d'un paramètre variant au cours des itérations. Ce paramètre est appelé température par analogie au domaine d'inspiration de cette heuristique (refroidissement lent de métaux à l'état liquide pour obtenir un état solide plus stable). En ce qui concerne la recherche tabou, c'est l'étape de modification de s qui diffère de RHC : s' est créée en s'assurant qu'elle n'a pas été générée dans un passé proche en vérifiant son absence dans une liste (la liste tabou, justement).

Nous continuons notre passage en revue des heuristiques stochastiques itératives par des méthodes s'inspirant de la nature et manipulant à chaque itération un ensemble de solutions.

5.4.4 Les méthodes à base de populations

Les heuristiques d'optimisation qui peuvent alimenter la comparaison avec les algorithmes de fourmis artificielles sont principalement celles qui manipulent un ensemble de solutions. Les algorithmes génétiques (AG) (et plus généralement les algorithmes d'évolution) rentrent dans ce cadre et vont principalement nous intéresser dans la suite de ce chapitre.

Les algorithmes d'évolution

Les algorithmes d'évolution (AE) sont apparus dans les années soixante et les premières publications marquantes datent du milieu des années soixante dix (Fogel et al., 1966; Holland, 1975; De Jong, 1975). Depuis le milieu des années quatrevingts c'est un domaine en expansion permanente et il serait illusoire de vouloir en donner un panorama exhaustif (voir par exemple (Goldberg, 1989; Michalewicz, 1996)).

Sous la dénomination « algorithme d'évolution » se cache en fait plusieurs techniques qui ont convergé par la suite. Toutes ces techniques ont en commun de s'inspirer des théories de l'évolution naturelle et reproduisent plus ou moins fidèlement les mécanismes issus de la génétique. En toute généralité un algorithme d'évolution (*Evolution Program*) peut être condensé tel que présenté par l'algorithme 5.3.

Algorithme 5.3: Structure générale d'un algorithme d'évolution.
EvolutionProgram()

(1) Générer aléatoirement une population de solutions candidates P et les évaluer
(2) **tantque** La condition d'arrêt n'est pas vérifiée **faire**
(3) Produire une population de nouvelles solutions candidates P' en effectuant des modifications aléatoires sur des éléments sélectionnés dans P
(4) Evaluer les solutions de P'
(5) Remplacer certains éléments de P par des éléments de P'
(6) **fintantque**
(7) **retourner** la meilleure solution de P

Une solution peut alors s'appeler un individu et l'ensemble des solutions s'appelle la population par analogie avec le monde du vivant.

Le premier avantage que l'on peut donner à cet algorithme est sa simplicité de formulation. Le second avantage est qu'à ce niveau de précision, la méthode reste très générale et peut donc être appliquée à un grand nombre de problèmes[2].

[2]Gardons à l'esprit que pour être compétitif, un algorithme d'évolution ne gardera pas longtemps cette façade très généraliste.

Nous donnons quelques caractéristiques des principaux algorithmes d'évolution :

Les algorithmes génétiques (AG). Bien que les AG (*Genetic Algorithm* : GA) n'aient pas été conçus initialement comme des algorithmes d'optimisation mais plutôt comme des procédures d'apprentissage pour des systèmes adaptatifs, ils peuvent traiter des problèmes d'optimisation variés. Dans sa formulation initiale (ou canonique (Goldberg, 1989)), l'AG manipule une population de chaînes binaires. L'étape 3 de l'algorithme 5.3 se décompose en trois phases (nous ne donnons que les grandes lignes) :

- **sélection :** $|P|$ individus sont sélectionnés dans la population $P = \{s_1, \ldots, s_{|P|}\}$ où chaque individu s_k a une probabilité $P_s(s_k) = \frac{f(s_k)}{\sum_{i=1}^{|P|} f(s_i)}$ d'être sélectionné, la population P^1 est ainsi constituée ;
- **croisement :** la liste des individus est parcourue et chaque individu a une probabilité P_c d'être choisi pour un croisement, quand on dispose de deux individus on tire aléatoirement un point de coupure et les chaînes binaires sont échangées par rapport à ce point, finalement on obtient la population P^2 ;
- **mutation :** les bits des individus de P^2 sont inversés avec une probabilité P_m, on obtient alors la population P'.

L'AG canonique a beaucoup évolué du point de vue de la représentation des solutions (codage réel, entier, ou mixte), ou de la sélection (par exemple par tournoi : on sélectionne le meilleur individu d'une sous-population extraite aléatoirement de P).

Les stratégies d'évolution (SE). Les premières apparitions des stratégies d'évolution (*Evolution Strategies* : ES) étaient destinées à optimiser la forme d'objets en soufflerie. La population était alors réduite à un seul individu, ses caractéristiques étaient réelles et seule une mutation basée sur la loi normale le faisait évoluer. Depuis, de nombreuses variantes sont apparues (voir par exemple (Bäck et al., 1991; Schwefel, 1995)). L'apport principal des SE est de considérer un individu comme un couple de vecteurs $(\mathbf{s}, \sigma)$ où $\mathbf{s}$ est une solution du problème et σ les écarts type utilisés par la mutation normale. Ces écarts type sont modifiés à chaque itération en fonction de la qualité de l'individu qu'ils ont contribué à générer. Des évolutions plus récentes des SE manipulent une population : λ descendants sont créés à partir de μ parents ($\lambda > \mu$). Deux stratégies de sélection sont souvent utilisées : la population suivante est choisie parmi les λ descendants (notée (μ, λ)-ES) ou alors parmi les $\lambda + \mu$ descendants et parents (notée $(\mu + \lambda)$-ES).

La programmation évolutive (PE). À son origine, la PE (*Evolutionary Programming* : EP) manipule des individus représentant des machines à états finis (Fogel et al., 1966). D'une génération à l'autre, chaque descendant est créé à partir d'un unique parent par mutation. Les meilleurs individus sont alors retenus pour former la génération suivante. De nombreuses évolutions de la PE sont apparues pour traiter notamment des problèmes d'optimisation numérique.

La programmation génétique(PG). Les individus manipulés par la PG (*Genetic Programming* : GP) ne sont plus des solutions du problème mais des programmes per-

mettant de le résoudre (Koza, 1994). Ces programmes sont représentés par des arbres dont les opérateurs génétiques modifient les branches et les feuilles représentant des instructions indivisibles. L'évaluation d'un individu dépend de sa capacité à résoudre le problème.

Il existe de nombreuses méthodes d'optimisation basées sur une population et plus ou moins proche des algorithmes d'évolution. On peut par exemple citer l'optimisation par essaim particulaires (*Particle Swarm Optimization* : PSO) (Kennedy and Eberhart, 1999), ou encore l'évolution différentielle (Price, 1999).

5.5 Objectifs

Il n'y a pas réellement de conclusion à apporter à ce chapitre, il avait pour principal objectif de fixer un certain nombre de notions qui seront utilisées par la suite. Dans le chapitre suivant, nous nous intéressons à une classe particulière d'algorithmes d'évolution : ces algorithmes ne manipulent pas vraiment une population d'individus (au sens ou nous l'avons présenté pour l'AG) mais un échantillonnage de la population. Ils manipulent des fréquences d'apparition des gènes (i.e. des valeurs de bit). Ces méthodes nous intéressent car on peut leur trouver un certain nombre de points commums avec une adaptation que nous proposons de l'heuristique ACO. Dans le chapitre 7, les algorithmes d'évolution nous permettront de donner quelques similitudes ou différences avec la modélisation des fourmis *Pachycondyla apicalis* que nous avons developpée pour résoudre des problèmes d'optimisation.

Chapitre 6

L'optimisation à base de populations

Ce chapitre présente une comparaison de plusieurs modèles probabilistes destinés à l'optimisation numérique. L'objectif est de montrer que de nombreux travaux issus des algorithmes génétiques ou des algorithmes à base de colonies de fourmis utilisent des principes très proches. Nous allons nous intéresser aux algorithmes BSC, PBIL et ACO car ces trois méthodes utilisent un vecteur de probabilités comme échantillon de l'espace de recherche. Les chaînes binaires qui représentent les solutions du problème sont générées selon ce vecteur que chaque méthode tente d'apprendre. Aprés avoir présenté les points communs et les différences entre ces trois algorithmes, nous présentons une étude expérimentale sur des fonctions binaires classiques.

6.1 Introduction

Les Algorithmes Génétiques (AG) et plus généralement les Algorithmes Évolutionnaires (AE) sont de plus en plus utilisés pour résoudre des problèmes d'optimisation difficiles. Dans ce type d'heuristiques, une population de solutions est générée aléatoirement et différents opérateurs, tels que la sélection, le croisement et la mutation, sont appliqués sur cette population pour en créer une nouvelle qui sera plus adaptée au problème. Les solutions peuvent être codées sous forme binaire, comme dans les AG, ou sous forme réelle comme dans les statégies d'évolution (SE).

Dans le but de rapprocher ces algorithmes des algorithmes à base de population de fourmis, nous nous sommes intéressés à un type particulier d'algorithmes évolutionaires : les algorithmes basés sur la fréquence des gènes ; au lieu de simuler une population entière de n individus qui seront sélectionnés, croisés puis mutés, une *distribution de probabilité* d'apparition des gènes est utilisée pour générer la population. Cette distribution est ensuite modifiée selon l'adaptation des individus de la population. Cette approche n'est pas encore très répandue comparée aux techniques d'opti-

misation génétiques standards mais nous sommes confiants dans le développement de cette famille d'algorithmes évolutionnaires[1]. De plus, les domaines d'application ne se limitent pas aux seuls domaines binaires. On peut en trouver un exemple dans (Ramat et al., 1997) où ce type de méthode est mis en œuvre sur des problèmes de planification avec contraintes de ressources multiples. Dans cette application, un algorithme génétique basé sur la fréquence des gènes est utilisé pour choisir les règles d'affectation et d'ordonnancement utilisées pour planifier un ensemble d'activités. Dans (Sebag and Ducoulombier, 1998) on peut trouver une application à des domaines continus. Nous avons aussi proposé d'utiliser une approche évolutionnaire se basant sur les fréquence de gène pour la génération interactive de feuilles de style pour site web (Monmarché et al., 1999). Nous présentons ici deux de ces approches : BSC (*Bit-Simulated Crossover*) (Syswerda, 1993) et PBIL (*Population Based Incremental Learning*) (Baluja, 1994; Baluja and Caruana, 1995; Baluja, 1995).

En ce qui concerne les fourmis artificielles, nous nous intéressons aux techniques d'optimisation de type ACO (*Ant Colonies Optimization*) (Colorni et al., 1991; Dorigo et al., 1996; Dorigo and Gambardella, 1997b; Dorigo and Di Caro, 1999b) présentées en détail dans le chapitre 2. ACO manipule aussi une population d'individus mais de façon assez différente des AGs. La technique générale d'ACO est d'utiliser des traces de phéromones pour guider la recherche d'optimum par les fourmis qui mettent à jour ces traces selon les solutions générées. Ces traces prennent la forme d'une distribution de probabilité sur un espace de recherche discret. Cette idée, qui a été développée indépendamment de BSC et PBIL, est cependant basée sur les mêmes principes : l'utilisation d'une distribution de probabilités pour résoudre un problème d'optimisation. Nous présentons dans ce chapitre deux adaptations des principes d'ACO au problème de l'optimisation de fonctions binaires.

Voici les objectifs de ce chapitre : dans un premier temps nous allons décrire les algorithmes BSC, PBIL et ACO afin d'en exhiber les similitudes et différences. Puis nous comparerons les résultats que l'on peut obtenir sur un ensemble de fonctions de complexités diverses. Enfin, nous présenterons les enseignements que l'on peut tirer de cette étude et les évolutions envisageables.

6.2 Notations et définitions

Afin de faciliter la description de ces trois algorithmes nous allons utiliser une structure commune pour s'assurer d'une comparaison équitable. Nous considérons le problème d'optimisation binaire standard qui consiste à trouver dans un espace de recherche $\mathcal{S} = \{0,1\}^l$ le minimum d'une fonction d'évaluation f, $f : \mathcal{S} \rightarrow \mathbb{R}^+$. Nous notons par s^* un minimum global de f (s^* n'est pas obligatoirement unique). La valeur du bit i de la chaîne s sera noté $s(i)$.

Les notations communes aux trois algorithmes utilisés pour résoudre ce problème standard sont les suivantes :

[1] Voir le workshop « Optimization by Building and Using Probabilistic Models » (OBUPM'2000) de la conférence GECCO2000 qui est l'un des premiers dans ce domaine (voir (Pelikan et al., 1999) pour un état de l'art).

- $V = (p_1, \ldots, p_l)$, avec $p_i \in [0, 1]$, qui représente le vecteur de probabilité qui sera utilisé pour générer des points de $\mathcal{S}$. Nous décidons que p_i représente la probabilité de générer un « 1 » ;
- $P = (s_1, \ldots, s_n)$, avec $s_i \in \mathcal{S}$, qui représente les n chaînes binaires qui seront générées à chaque cycle. P peut être considérée comme la population dans les algorithmes évolutionnaires.

L'algorithme général (noté BPA à partir des initiales de BSC, PBIL et ACO) est donné par l'algorithme 6.1

Algorithme 6.1: Algorithme commun aux algorithmes BSC, PBIL et ACO
BPA()

(1) Initialisation de $V = (p_1, ..., p_l)$ (en général à $(0.5, ..., 0.5)$)
(2) **tantque** La condition de terminaison n'est pas vérifiée **faire**
(3) Générer $P = (s_1, ..., s_n)$ en utilisant V
(4) Évaluer $f(s_1), ..., f(s_n)$
(5) Mettre à jour V selon $(s_1, ..., s_n)$ et $f(s_1), ..., f(s_n)$
(6) **fintantque**
(7) **retourner** s^+, la meilleure solution trouvée

La condition de terminaison peut être de plusieurs types :
- le temps imparti T_1 est atteint ;
- le minimum atteint n'a pas été amélioré depuis T_2 itérations ;
- le nombre d'évaluations de la fonction f a atteint la valeur T_3.

T_1, T_2 ou T_3 sont alors des paramètres d'entrée de BPA.

Les deux premiers critères s'appliquent quand on rencontre des contraintes réelles : « on dispose d'un temps T_1 pour avoir une réponse » ou « on veut le meilleur résultat que la méthode puisse donner ». Par la suite, nous ne considérerons que le paramètre T_3. Ce critère d'arrêt est bien adapté à la comparaison de méthodes d'optimisation puisque le nombre d'itérations de la boucle dépend de la taille de la population. Ce critère permet aussi de prendre en compte des problèmes dont l'évaluation de la fonction objectif est coûteuse en temps de calcul : il s'agit de trouver un optimum global avec un nombre limité d'évaluations de f. Ce critère permet aussi de fixer le nombre de solutions explorées relativement à la taille de l'espace de recherche.

Dans les sections suivantes nous montrons comment ces algorithmes innovants diffèrent les uns des autres, principalement sur l'étape 5, la règle de mise à jour du vecteur de probabilité V. Un exemple complet est donné en annexe B.

6.3 L'algorithme BSC (*Bit-Simulated Crossover*)

À notre connaissance, Syswerda (Syswerda, 1993) a été le premier à suggérer de remplacer l'opérateur de croisement des Algorithmes Génétiques par un opérateur plus général utilisant le vecteur V défini précédemment. Classiquement, l'opérateur de croisement opère de la façon suivante :

– sélectionner deux chaînes binaires s_1 et s_2 dans $\mathcal{S}$;
– échanger des sous-chaînes de s_1 et s_2 pour former deux chaînes « filles » s'_1 et s'_2.

Syswerda a initialement été motivé par l'observation que le croisement, qu'il soit effectué en un point, deux points ou de façon uniforme, conserve la proportion de « 0 » et de « 1 » pour une position i ($i \in \{1, \ldots, l\}$) des chaînes sélectionnées. Le pas suivant franchi par Syswerda a été de calculer explicitement ces proportions et d'utiliser ces probabilités pour générer les individus à la place de l'opérateur classique de croisement. Il a appelé ce nouvel opérateur le BSC (*Bit-Simulated Crossover*). De plus, il suggère de prendre en compte les valeurs d'adaptation des individus pour orienter le vecteur V vers des régions intéressantes de l'espace de recherche en donnant un poids plus important aux individus les mieux adaptés. Si, par exemple, les individus possédant un « 1 » à la position i ont plus de chances d'être sélectionnés, la proportion de « 1 » à cette position va augmenter dans la génération suivante.

La mutation proposée par Syswerda est équivalente à une petite perturbation de V ce qui revient à rapprocher chaque composante de V de l'équiprobabilité de générer un « 0 » ou un « 1 ».

La première étape de BSC consiste à initialiser toutes les valeurs de V à 0.5. La règle de mise à jour de V peut être établie[2] selon les deux points suivants :

1. **Simuler le crossover avec sélection**. Pour chaque position i, $i \in \{1, \ldots, l\}$, il faut calculer les valeurs suivantes :

$$p_i \leftarrow \frac{\sum_{s_j \in P} s_j(i) \cdot \omega(s_j)}{\sum_{s_j \in P} \omega(s_j)} \tag{6.1}$$

où $\omega(s_j)$ représente le poids associé à s_j. Si $\omega(s_j) = 1, \forall s_j \in P$, alors p_i est égal à la proportion de « 1 » à la position i dans P. Pour rapprocher p_i des valeurs intéressantes des bits, on peut par exemple fixer $\omega(s_j)$ à $n - Rank(s_j)$, où $Rank(s_j)$ est le rang de s_j dans l'ensemble P selon les valeurs d'adaptation $f(s_1) \ldots f(s_n)$, ce qui signifie que le rang de la meilleure chaîne de P est 1 et que le rang de la plus mauvaise est n.

2. **Simuler la mutation**. La probabilité qu'un bit soit modifié par la mutation est donnée par p_m ($p_m < 0.5$) qui est un paramètre de BSC. Pour chaque position i, $i \in \{1, \ldots, l\}$, la probabilité p_i de générer un « 1 » avec la mutation p_m est modifiée de la façon suivante :

$$p_i \leftarrow p_i(1 - p_m) + (1 - p_i)p_m \tag{6.2}$$

$p_i(1 - p_m)$ représente la probabilité de générer un « 1 » sans faire de mutation et $(1 - p_i)p_m$ représente la probabilité de générer un « 0 » puis de le muter. La formule 6.2 implique que $p_i \in [p_m, 1 - p_m]$. La probabilité de mutation permet donc de conserver une certaine probabilité d'exploration pour chaque bit.

[2]Il s'agit bien de l'établir puisque l'article de Syswerda ne donne que les idées générales.

BSC ignore les séquences de bit (schémas) présents dans la population, il ne se base que sur les statistiques que l'on peut déduire sur une position. Il a cependant été montré que la génération des individus suivant le vecteur de probabilité V reproduit la transmission des schémas. Par exemple, pour deux séquences de deux bits présentes dans la population, BSC a une probabilité de construire un individu contenant ces deux séquences supérieure à un opérateur de croisement explicite si la probabilité de sélection des deux schémas est supérieure à 0.3.

Syswerda a présenté des résultats obtenus par BSC sur des problèmes comportant de 40 à 300 bits. BSC s'est montré compétitif relativement à des opérateurs de croisement à un, deux points ou uniforme ainsi que face à une mutation seule. Concernant le principal paramètre de cette méthode, la taille de la population (n), Syswerda montre que la taille optimale de la population dépend de la méthode, du problème et du nombre d'itérations (T_3).

6.4 L'algorithme PBIL (*Population-Based Incremental Learning*)

À la suite des travaux de Syswerda, ainsi qu'avec la contribution d'Ari Juels (dont nous ne disposons pas de référence bibliographique), Baluja a développé PBIL, un autre modèle d'algorithme basé sur l'exploration d'un espace de recherche en utilisant un vecteur de probabilité (Baluja, 1994; Baluja and Caruana, 1995; Baluja, 1995).

Le parallélisme implicite d'un AG, c'est-à-dire la capacité à représenter un sous ensemble potentiellement important de l'espace de recherche, est difficile à maintenir tout au long des générations et c'est un des inconvénients majeurs des AG travaillant sur une population de taille finie. Baluja propose de contourner le problème en utilisant un prototype de la population représenté par un vecteur de probabilité.

Les principes de PBIL sont quelque peu différents de ceux utilisés par BSC et s'éloignent un peu plus des algorithmes génétiques « traditionnels » en introduisant une règle de mise à jour inspirée de l'apprentissage compétitif (*competitive learning*) utilisé pour l'apprentissage des cartes de Kohonen (*Kohonen's features map*)(Kohonen, 1988).

Le point central de PBIL concerne la règle utilisée pour l'étape 5 de l'algorithme 6.1. PBIL opère de la façon suivante :

1. **Soit** s^+ la meilleure chaîne générée dans la population $P = (s_1, \ldots, s_n)$;
2. **Mettre à jour** V : pour chaque position de bit i ($i \in \{1, \ldots, l\}$),

$$p_i \leftarrow p_i(1 - LR) + s^+(i)LR \tag{6.3}$$

   où LR correspond au taux d'apprentissage (*Learning Rate*) qui contrôle l'amplitude des modifications de V ;
3. **Effectuer la mutation**. Pour chaque position de bit i, avec une probabilité de mutation p_m, modifier p_i de la façon suivante :

$$p_i \leftarrow p_i(1 - \delta_m) + r\delta_m \tag{6.4}$$

où δ_m est un paramètre fixant l'amplitude de la mutation effectuée et r est une variable aléatoire dans $\{0, 1\}$ représentant la direction de la mutation.

Un certain nombre d'extensions de PBIL ont été proposées :

- au lieu d'utiliser la meilleure chaîne générée à chaque itération on peut utiliser les m meilleures. La mise à jour du vecteur V demande alors quelques ajustements : il y principalement deux solutions :
  1. le vecteur V est déplacé de la même manière vers les m meilleures chaînes,
  2. le vecteur V est déplacé vers les valeurs des bits communs aux m meilleures chaînes.
- le vecteur V peut aussi être « éloigné » de la plus mauvaise chaîne (notée s^-) générée dans P :

$$p_i \leftarrow p_i(1 - LR_{neg}) + s^+(i)LR_{neg} \quad \text{si} \quad s^+(i) \neq s^-(i) \tag{6.5}$$

  Dans ce cas, LR_{neg} est appelé taux d'apprentissage négatif (*Negative Learning Rate*). Cette formule signifie que p_i est rapproché de $s^+(i)$ si les bits à la position i de s^+ et s^- sont différents.

L'algorithme PBIL a été étudié empiriquement sur plusieurs problèmes (Jobshop, voyageur de commerce, bin-packing, optimisation de fonctions numériques) avec les valeurs de paramètres suivants (Baluja, 1994) :

- $n = 100$;
- $LR = 0.1$;
- $p_m = 0.02$;
- $\delta_m = 0.05$;
- $LR_{neg} \in \{0.0, 0.025, 0.075, 0.1\}$.

Une comparaison est proposée avec un algorithme génétique standard et avec un algorithme de descente stochastique (*Multiple Restart, Next-Step Hill Climbing*). Les résultats obtenus montrent que l'apprentissage négatif joue un rôle important mais que sa valeur dépend du problème. De plus PBIL obtient des performances supérieures aux autres méthodes que ce soit en terme de qualité que de rapidité (Baluja, 1994; Baluja, 1995).

Plusieurs extensions ont été proposées et abandonnent l'espace de recherche binaire pour des espaces à variables continues (Rudlof and Koeppen, 1996; Sebag and Ducoulombier, 1998). Dans ce cas, V est utilisé pour générer des valeurs réelles suivant des distributions gaussiennes autour de chaque valeur p_i.

6.5 L'algorithme ACO(*Ant Colony Optimization*)

La capacité des fourmis à trouver le plus court chemin entre une source de nourriture et leur nid a été utilisée pour résoudre des problèmes d'optimisation combinatoire. Les traces de phéromones représentent une attirance pour un arc du graphe modélisant le problème. Chaque fourmi construit une solution du problème et l'évaluation de chaque solution est utilisée pour mettre à jour les traces de phéromones. Ces principes ont été appliqués en premier au problème du voyageur de commerce (Colorni et al., 1991) puis

à d'autres problèmes combinatoires comme le problème de l'affectation quadratique (Maniezzo and Colorni, 1999). Nous renvoyons le lecteur au chapitre 2 pour le détail de ces heuristiques inspirées des fourmis qui sont rassemblées sous l'acronyme ACO (Ant Colony Optimization).

Nous avons reformulé le problème d'optimisation considéré dans ce chapitre pour proposer une heuristique basée sur ACO. Plus précisément, nous nous inspirons des développements d'ACO qui ont été appliqués au problème du voyageur de commerce, à savoir les heuristiques AS (*Ant System*) (Dorigo et al., 1996) et ACS (*Ant Colony System*) (Dorigo and Gambardella, 1997b). Nous appelerons respectivement AS_b et ACS_b les deux adaptations que nous proposons de AS et ACS pour le problème d'optimisation binaire.

Nous pouvons montrer que si nous adaptons les principes d'ACO au problème d'optimization binaire, cela diffère de BSC et PBIL principalement sur l'étape de mise à jour de l'algorithme 6.1. Le problème d'optimisation a été reformulé de la façon suivante : nous construisons un graphe où chaque sommet correspond à la position d'un bit et où les arcs correspondent au choix de la valeur du bit. La figure 6.1.a représente le graphe contenant les différents sommets qu'une fourmi doit parcourir pour construire une solution. La fourmi part du premier sommet sur la gauche et choisit un arc, soit « 1 » ou « 0 », pour atteindre le sommet suivant. La décision de choisir l'arc « 1 » ou l'arc « 0 » suit une distribution de probabilité que l'on appelle trace de phéromones dans ACO mais qui peut être ramenée à une seule valeur correspondant à la probabilité de suivre l'arc « 1 ». Notons par 0_i et 1_i les deux arcs correspondant à la position i de la chaîne de bits. Les quantités de phéromones de chaque arc sont τ_{0_i} et τ_{1_i}. Ces deux valeurs réelles peuvent être utilisées pour définir une unique valeur p_i, la probabilité de générer un « 1 » :

$$p_i = \frac{\tau_{1_i}}{\tau_{1_i} + \tau_{0_i}} \tag{6.6}$$

Les traces de phéromones sont donc équivalentes au vecteur V utilisé par les deux précédentes méthodes. La figure 6.1.b illustre la solution $s = 010\ldots00$ générée par une fourmi.

Initialement, dans ACO, chaque arc 0_i et 1_i ($i \in \{1, \ldots, l\}$) a une quantité de phéromone τ_{0_i} et τ_{1_i} fixée à une valeur positive τ^0.

Il est assez évident que cette modélisation de la recherche d'une chaîne binaire sous la forme d'un graphe aussi peu élaboré représente une simplification assez forte de ce que les algorithmes de type ACO étaient habitués à traiter : pour un problème de voyageur de commerce à l villes, à chaque sommet, la fourmi doit choisir parmi $l-k$ arcs (k représentant le nombre de villes déjà explorées) alors que pour notre modélisation, elle ne possède à chaque sommet qu'une alternative entre deux chemins. Cependant, et à notre connaissance, cela n'avait pas été proposé alors que ce problème compte parmi les problèmes classiques en informatique.

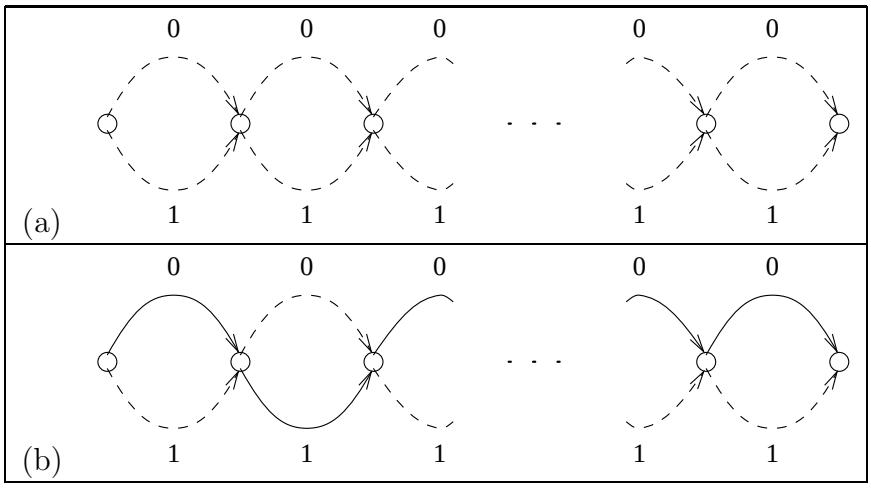


FIG. 6.1 – Adaptation de ACO au problème d'optimization binaire. Le choix des l bits est modélisé par le choix d'un arc « 0 » ou « 1 » entre les $l+1$ sommets.

6.5.1 L'algorithme AS$_b$ (*Ant System*)

Selon la règle de mise à jour de AS, τ_{k_i} ($k \in \{0,1\}, i \in \{1 \dots l\}$), est modifié de la façon suivante :

$$\tau_{k_i} \leftarrow (1-\rho)\tau_{k_i} + \sum_{j=1}^{n} \Delta_{k_i}^{j} \tag{6.7}$$

où $\rho \in [0,1]$ est un paramètre représentant l'évaporation des phéromones et $\Delta_{k_i}^{j}$ correspond à la quantité de phéromone déposée par la fourmi j sur l'arc k_i :

$$\Delta_{k_i}^{j} = \begin{cases} \frac{1}{1+f(s_j)} & \text{si } s_j(i) = k \\ 0 & \text{sinon} \end{cases} \tag{6.8}$$

Comme f est définie sur $\mathbb{R}^+$, au dénominateur, nous avons ajouté 1 à $f(s_j)$ pour éviter les problèmes survenant lorsque $f(x) = 0$.

Il existe une extension de l'algorithme AS : AS-Rank où toutes les solutions sont utilisées de façon proportionnelle à leur performance (Bullnheimer et al., 1997b). Ceci est comparable à l'utilisation des m meilleures chaînes dans PBIL.

6.5.2 L'algorithme ACS$_b$ (*Ant Colony System*)

Une variante d'AS, ACS, a apporté des changements importants à la règle de mise à jour. ACS utilise uniquement la meilleure chaîne générée depuis le début de l'algorithme (notée s^{++}) :

$$\tau_{k_i} \leftarrow (1-\rho)\tau_{k_i} + \rho\Delta_{k_i} \tag{6.9}$$

où :

$$\Delta_{k_i} = \begin{cases} \frac{1}{1+f(s^{++})} & \text{si } s^{++}(i) = k \\ 0 & \text{sinon} \end{cases} \tag{6.10}$$

On peut remarquer que cette règle est assez proche de celle utilisée par PBIL (formule 6.3) à la différence qu'elle s'applique ici sur les phéromones et non pas directement sur les probabilités p_i du vecteur V. Comme pour AS_b, ρ sert de coefficient d'évaporation des phéromones.

ACS introduit un moyen de contrôler le compromis entre l'exploration et l'exploitation [3]. Pour chaque bit généré pour la solution i à l'étape 3 de l'algorithme 6.1 :

- diversifier avec la probabilité $1 - q_0$: le bit j prend la valeur « 1 » avec la probabilité p_j ;
- intensifier avec la probabilité q_0 : le bit j prend la valeur suivante

$$s_i(j) = \begin{cases} 1 & \text{si } p_i > 0.5 \\ 0 & \text{sinon} \end{cases} \tag{6.11}$$

De plus, ACS utilise une règle de mise à jour locale mise en œuvre à l'étape 3 de génération de la population dans l'algorithme 6.1 :

$$\tau_{k_i} \leftarrow (1-\alpha)\tau_{k_i} + \alpha\tau^0 \quad \text{si l'arc } k_i \text{ a été choisi} \tag{6.12}$$

où $\alpha \in [0, 1]$ est un paramètre. Cette mise à jour locale a pour but de modifier très légèrement la quantité de phéromones sur l'arc choisi par une fourmi afin de pousser les autres à explorer les autres arcs, cela afin d'éviter que toutes les fourmis se suivent. Ainsi, si la quantité de phéromones sur l'arc k_i est supérieure à τ^0, aprés le passage d'une fourmi, la quantité de phéromone aura diminué. Par contre, si la quantité τ_{k_i} est inférieure à τ^0, la formule 6.12 augmente la quantité de phéromone. Cette mise à jour locale agit statistiquement de la même façon que la mutation de BSC sur les composantes de V. En effet, si $\tau_{i_0} < \tau_{i_1}$, après le passage des n fourmis, τ_{i_0} sera augmenté et τ_{i_1} sera diminué ce qui implique que p_i sera rapprochée de 0.5.

6.6 À propos de complexité

Un certain nombre de remarques peuvent être formulées sur la complexité des ces quatre méthodes. Voici pour une itération les opérations effectuées par chaque méthode :

- BSC :
  - $n \times l$ tirages aléatoires pour générer la population,
  - n évaluations de f,
  - trier les résultats de l'évaluation pour déterminer le rang de chaque solution ($\mathcal{O}(n \log n)$),
  - parcourir l fois toutes les solutions pour mettre à jour V (formule 6.1), ce qui représente la consultation de $l \times n$ bits,
  - parcourir le vecteur V pour effectuer la mutation (formule 6.2).
- PBIL :
  - $n \times l$ tirages aléatoires pour générer la population,
  - n évaluations de f,

[3] On peut aussi parler d'intensification et de diversification par analogie avec le recuit simulé.

  - parcourir le vecteur V pour le mettre à jour (formules 6.3 et 6.5), ce qui représente l'exploration de $2 \times l$ bits,
  - parcourir le vecteur V pour effectuer la mutation (formule 6.4), ce qui représente l tirages aléatoires.
- AS_b :
  - $n \times l$ tirages aléatoires pour générer la population,
  - n évaluations de f,
  - parcourir l fois toutes les solutions pour mettre à jour V (formule 6.7), ce qui représente la consultation de $2 \times l \times n$ bits,
- ACS_b :
  - $2 \times n \times l$ tirages aléatoires pour générer la population,
  - n évaluations de f,
  - parcourir le vecteur V pour le mettre à jour (formule 6.9), ce qui représente l'exploration de $2 \times l$ bits,

La différence la plus importante se situe pour la mise à jour de V : alors que PBIL et ACS_b ne « consomment » que l'exploration de $2 \times l$ bits, AS_b et BSC explorent $l \times n$ bits (avec pour BSC un tri suplémentaire).

6.7 Comparaison expérimentale

L'objet de ce chapitre 6 est de comparer quatre méthodes d'optimisation. Les sections précédentes ont décrit en détail les méthodes d'un point de vue algorithmique. Il faut maintenant passer à la pratique. Dans un premier temps, nous allons étudier chaque méthode par rapport à elle même : quels sont les bonnes valeurs pour leurs paramètres ? Ensuite, une fois chaque méthode affûtée, nous pourrons les comparer entre elles.

6.7.1 Jeux de tests

Nous allons nous intéresser à un type de problème d'optimisation particulier : l'optimisation de fonctions numériques. Ces fonctions peuvent être définies à partir de valeurs réelles (qui peuvent être traduites sous forme binaire) ou avoir déjà une forme binaire. Nous mettons de côté les problèmes dynamiques, les problèmes avec contraintes ainsi que les problèmes comportant du bruit.

Le choix du jeu de tests est un problème difficile. Il y a principalement deux directions :

- on dispose de problèmes réels avec un certain nombre de contraintes, par exemple de temps, et d'un expert du domaine à l'origine du problème qui pourra donner son état de satisfaction sur chacune des méthodes ;
- nous devons nous même nous poser les problèmes et évaluer l'intérêt des différentes méthodes sur ces jeux de tests.

La deuxième solution s'impose à nous. La question est alors : comment construire un jeu de fonctions de tests correspondant à un certain nombre de préoccupations de généralité et de difficulté ? En ce qui concerne la généralité, il ne suffit pas de

disposer d'un nombre pléthorique de fonctions pour s'assurer de couvrir tous les cas possibles. Nous devrons donc prendre un certain nombre de précautions et rappeler que les conclusions obtenues seront évidemment uniquement valides pour le jeu de tests choisi. Pour ce qui est de la difficulté, un certain nombre de travaux dans le domaine des algorithmes évolutionnaires peuvent nous aider. Ces travaux sont apparus à la suite de nombreux travaux mettant en valeur une méthode plutôt qu'une autre, ce qui est notre cas ici avec la nuance que nous ne cherchons pas à montrer que les algorithmes basés sur un vecteur de probabilité sont plus ou moins efficaces que d'autres méthodes, nous cherchons « simplement » à comparer ces algorithmes entre eux (et encore nous limitons l'étude aux quatre méthodes présentées précédenmment). WHITLEY et ses collègues ont proposé un certain nombre de reflexions sur le sujet (Whitley et al., 1995; Whitley et al., 1996), que voici :

- la plupart des jeux de tests utilisés pour évaluer les algorithmes d'évolution sont souvent les mêmes et ne vérifient pas les propriétés qui suivent ;
- les fonctions de tests doivent être non séparables, ce qui signifie que les intéractions entre les variables doivent être non linéaires ;
- les changements d'échelle ne doivent pas altérer les propriétés d'une fonction ;
- les fonctions doivent être invariantes par rapport au codage binaire utilisé.

Un des prérequis à l'étude d'algorithmes stochastiques est que les fonctions de tests soient suffisamment difficiles pour les algorithmes les plus simples de cette classe : les algorithme du type « hill-climbing ». En effet, pourquoi développer des méthodes d'une complexité supérieure alors que le problème est résolu avec un effort calculatoire moindre ? Cependant cette dernière remarque ne nous concerne que modérément : nous ne nous intéressons pas à la résolution efficace de problèmes en général mais à la comparaison de quatre méthodes entre elles, ce qui signifie que nous ne cherchons pas ici à démontrer leurs peformances « brutes » mais simplement à déterminer s'il existe des différences entre quatres approches qui ne diffèrent qu'en quelques points précis.

La non-séparabilité des fonctions est un point qui mérite une certaine attention. Si on prend une fonction séparable de n variables, chacune codée sur k bits on peut s'attendre à travailler sur un espace de recherche de 2^{nk} solutions. Cependant, du fait de la séparabilité, il suffit de parcourir $n2^k$ solutions pour connaître l'optimum global. La séparabilité signifie que l'on peut chercher l'optimum d'un problème de façon indépendante pour chacune de ses dimensions. L'exemple de problème séparable le plus immédiat est la fonction suivante :

$$F_1(x) = \sum_{i=1}^{n} x_i^2 \tag{6.13}$$

Si $n = 100$ et $k = 10$, la dimension de l'espace de recherche est 2^{1000} (environ 10^{300}). Comme chaque variable peut être étudiée séparément, le nombre de solutions à tester est alors 100×2^{10} (environ 10^5).

Les problèmes de changement d'échelle surviennent quand l'augmentation de la dimensionalité d'un problème altère ses caractéristiques, principalement sa complexité.

Pour un même problème, deux codages différents des solutions peuvent modifier la topologie de l'espace de recherche. Par exemple, un codage binaire décimal ne conserve

pas les propriétés d'un problème à variables réelles comme pourrait le faire un codage de Gray. Autrement dit, si le problème admet une représentation pour laquelle sa résolution est facilitée, il perd de son intérêt.

Etant données toutes ces précautions, nous considérons que si nous comparons des algorithmes très proches dans leurs principes, nous ne faussons pas leur comparaison en les appliquant à des problèmes ne vérifiant pas toutes ces propriétés. Cela signifie que même en sachant que la fonction F_1 ne représente pas un problème complexe[4], cela ne nous empêche pas de comparer BSC, PBIL, AS_b et ACS_b, qui utilisent des principes très proches, pour traiter F_1. De plus, il est tout de même instructif de constater les performances d'un algorithme stochastique sur un problème facile, d'abord du fait de sa stochasticité, puis parce que les problèmes du monde réel qui se présentent ne sont pas toujours identifiables comme faciles ou difficiles.

Le tableau 6.1 donne les descriptions analytiques des problèmes de tests que nous utiliserons dans la suite.

La fonction F_0 est conçue pour mettre en difficulté les algorithmes génétiques (Whitley, 1991). Elle est construite de façon à posséder un bassin d'attraction autour d'un minimum local au sens de la distance de Hamming. Elle est définie de la façon suivante : pour chaque groupe de quatre bit on a :

$$\begin{array}{cccc} f(1111)=0 & f(0100)=8 & f(0110)=16 & f(1110)=24 \\ f(0000)=2 & f(1000)=10 & f(1001)=18 & f(1101)=26 \\ f(0001)=4 & f(0011)=12 & f(1010)=20 & f(1011)=28 \\ f(0010)=6 & f(0101)=14 & f(1100)=22 & f(0111)=30 \end{array}$$

F_0 est alors la somme des valeurs données par f sur les groupes de quatre bits. Le minimum local est rencontré lorsque $s = 00\ldots00$ alors que le minimum global est atteint lorsque $s = 11\ldots11$.

F_1 et F_2 font partie des fonctions de test définies par DeJong (De Jong, 1975), F_3 est la fonction dite de Rastrigin, F_4 et F_5 sont deux versions de dimensions différentes de la fonction de Griewank. La fonction F_6 est la fonction de Schaffer. Toutes les fonctions sont définies sur un produit d'intervalles de $\mathbb{R}$ et admettent un minimum de 0. Ce minimum est atteint en $(0,\ldots,0)$ pour les fonctions F_1 à F_9. Nous avons restreint cet échantillon de fonctions afin de pouvoir tester un nombre important de paramètres pour chaque méthode.

6.7.2 Paramètres des expériences

Un problème couramment rencontré avec les heuristiques d'optimisation telles que BSC, PBIL ou ACO consiste à fixer les paramètres de ces méthodes et de trouver les valeurs sensibles de ces paramètres. Nous fournissons une réponse empirique à cette question.

Le tableau 6.2 donne les paramètres que nous avons testés pour chaque méthode. Pour AS_b et ACS_b nous avons fixé τ^0 à 0.5.

[4]De toutes les façons, F_1 n'était pas un problème difficile avant de constater la séparabilité des variables puisque les méthodes d'optimisation numériques classiques donnent des résultats exacts pour ce problème.

Fonction		x_i
F_0	10×4 bits deceptive function DF2 (Whitley, 1991)	
F_1	$\sum_{i=1}^{3} x_i^2$	$[-5.12, 5.11]$
F_2	$100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$[-2.048, 2.047]$
F_3	$50 + \sum_{i=1}^{5} \left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12, 5.11]$
F_4	$1 + \sum_{i=1}^{2} \frac{x_i^2}{4000} - \prod_{i=1}^{2} \cos\left(\frac{x_i}{\sqrt{i}}\right)$	$[-512, 511]$
F_5	$1 + \sum_{i=1}^{5} \frac{x_i^2}{4000} - \prod_{i=1}^{5} \cos\left(\frac{x_i}{\sqrt{i}}\right)$	$[-512, 511]$
F_6	$0.5 + \frac{\sin^2(\sqrt{x_1^2 + x_2^2}) - 0.5}{\left(1 + 0.001(x_1^2 + x_2^2)\right)^2}$	$[-100, 100]$
F_7	$(x_1^2 + x_2^2)^{0.25} \times \left(1 + \sin^2 50(x_1^2 + x_2^2)^{0.1}\right)$	$[-100, 100]$
F_8	$\sum_{i=1}^{100} \lvert y_i \rvert$ avec $y_1 = x_1$, $y_i = x_i + y_{i-1}$ si $i \geq 2$	[-2.56,2.56]
F_9	$\sum_{i=1}^{100} \lvert y_i \rvert$ avec $y_1 = x_1$, $y_i = x_i + \sin(y_{i-1})$ si $i \geq 2$	[-2.56,2.56]
F_{10}	$\sum_{i=1}^{100} \lvert 0.024(i+1) - x_i \rvert$	[-2.56,2.56]

TAB. 6.1 – Les fonctions binaires qui sont utilisées pour les tests. DF2 se réfère à une « deceptive function » décrite dans (Whitley, 1991).Les sept fonctions suivantes, sont décrites dans (Whitley et al., 1995). Les trois dernières fonctions sont tirées de (Sebag and Ducoulombier, 1998).

Le tableau 6.3 donne les valeurs testées pour chaque paramètre. Trois paramètres sont communs à toutes les méthodes et ne figurent pas dans ce tableau : la taille de la population, le nombre d'évaluations de la fonction et le nombre de bits utilisés pour coder une valeur. Concernant les deux premiers, les valeurs suivantes sont testées :

- $n \in \{10, 50, 100, 200\}$;
- $T_3 \in \{1\,000, 10\,000, 100\,000\}$.

Pour toutes les expériences les valeurs réelles (pour toutes les fonctions sauf F_0) seront codées sur 10 bits. Par la suite, chaque résultat correspond à une moyenne sur trente essais. Le plus simple est de relever le minimum obtenu pour chaque fonction. Pour PBIL, cela représente $4 \times 3 \times 3 \times 3 \times 4 \times 3 = 1\,296$ jeux de paramètres différents. Le nombre de jeux de tests par méthode est indiqué dans le tableau 6.4.

Algorithmes	Paramètres
BSC	p_m
PBIL	$p_m, LR, \delta_m, LR_{neg}$
AS_b	ρ
ACS_b	ρ, q_0, α

TAB. 6.2 – Paramètres utilisés pour les algorithmes BSC, PBIL, AS_b et ACS_b.

Paramètres	Valeurs
LR, ρ	$\{0.001, 0.01, 0.05, 0.1\}$
LR_{neg}	$\{0.001, 0.01, 0.1\}$
p_m	$\{0.001, 0.01, 0.1\}$
δ_m	$\{0.001, 0.01, 0.1\}$
q_0	$\{0, 0.2, 0.5, 0.7, 0.9\}$
α	$\{0.001, 0.01, 0.05, 0.1\}$

TAB. 6.3 – Valeurs des paramètres utilisées pour les algorithmes BSC, PBIL, AS_b et ACS_b.

6.7.3 Etude de la convergence

Nous avons utilisé deux mesures permettant d'estimer la vitesse à laquelle chacune des méthodes apprend le vecteur V. Il faut garder à l'esprit que si le vecteur de probabilité converge trop rapidement (c'est-à-dire que ses composantes sont proches de 1 ou 0), les individus générés seront très similaires, ce qui représente un inconvénient majeur quand V représente un minimum local. Quand on manipule une population trop homogène, on perd le parallélisme implicite des algorithmes à base de population. Il convient donc de mettre en parallèle les mesures de convergence de V avec le minimum atteint par la méthode.

La mesure de convergence C

La convergence $C(t)$ est définie de la façon suivante :

$$C(t) = \frac{1}{l} \sum_{i=1}^{l} |p_i(t) - 0.5| \tag{6.14}$$

C peut être interprétée comme l'éloignement moyen d'une composante de V de l'équiprobabilité. $C \in [0, 0.5]$.

La mesure d'entropie E

La mesure $E(t)$ est définie par la formule :

$$E(t) = \frac{\log(\prod_{i=1}^{l} \max(p_i(t), 1 - p_i(t)))}{\log 0.5^l} \tag{6.15}$$

Algorithmes	Nombre de jeux de test
BSC	36
PBIL	1 296
AS	48
ACS	960

TAB. 6.4 – Nombre de jeux de test par algorithme.

Cette mesure donne une évaluation de dispersion de V. La dispersion étant minimale en 0 et maximale en 1. Cette mesure est assez similaire à la mesure de convergence précédemment introduite avec comme avantage d'être bornée par 0 et 1 et de représenter le désordre de V.

Le coefficient de Fisher F

Afin d'avoir une idée de la stabilité des quatre algorithmes, nous calculons pour les trente essais le coefficient d'aplatissement correspondant (second coefficient de Fisher). Ce coefficient permet de juger l'étalement des solutions trouvées par rapport à une loi normale. Cela permet d'avoir un peu plus de renseignements qu'en se basant simplement sur la moyenne et l'écart type des minima obtenus. Le coefficient de Fisher, noté F, est calculé de la façon suivante :

$$F = \frac{\mu_4}{\sigma^4} \tag{6.16}$$

où μ_4 est le moment centré d'ordre 4 :

$$\mu_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^k \tag{6.17}$$

et σ est l'écart type :

$$\sigma = \sqrt{\mu_2} \tag{6.18}$$

On peut alors montrer (Lebœuf et al., 1983) :

- si $F < 3$, la série statistique considérée est plus aplatie qu'une série statistique normale[5] ;
- si $F = 3$, la série statistique considérée a le même aplatissement qu'une série statistique normale[6] ;
- Si $F > 3$, la série statistique considérée est plus pointue qu'une série statistique normale[7].

[5]La série est dite platykurtique.
[6]La série est dite mésokurtique.
[7]La série est dite leptokurtique.

6.7.4 Recherche des meilleurs paramètres

Les tableaux 6.5, 6.6, 6.7 et 6.8 présentent les résutats obtenus respectivement par les algorithmes BSC, PBIL, AS et ACS. Chaque méthode a disposé de 1 000 évaluations de la fonction considérée et pour chaque fonction nous fournissons le meilleur et le pire minimum moyen obtenu sur 30 essais ainsi que les paramètres correspondants. C'est PBIL qui a obtenu le plus souvent le meilleur résultat pour une fonction donnée (signalé par ▶).

F_i	$\text{Param}_{\min}(n,p_m)$ $\text{Param}_{\max}(n,p_m)$		Min Max	$[\sigma_{\min}]$ $[\sigma_{\max}]$	$F_{\min}$ $F_{\max}$
F_0	(10,0.01)	▶	$1.77 \times 10^{+1}$	$[2.26 \times 10^{+0}]$	3.34
	(200,0.1)		$5.24 \times 10^{+1}$	$[8.60 \times 10^{+0}]$	2.50
F_1	(10,0.01)		4.48×10^{-4}	$[2.99 \times 10^{-4}]$	6.88
	(10,0.001)		7.75×10^{-1}	$[8.02 \times 10^{-1}]$	4.03
F_2	(100,0.001)		1.61×10^{-2}	$[1.35 \times 10^{-2}]$	3.48
	(10,0.001)		3.43×10^{-1}	$[3.85 \times 10^{-1}]$	5.16
F_3	(10,0.01)		$9.11 \times 10^{+0}$	$[4.04 \times 10^{+0}]$	4.75
	(200,0.01)		$2.64 \times 10^{+1}$	$[4.49 \times 10^{+0}]$	2.09
F_4	(50,0.001)		1.16×10^{-1}	$[1.12 \times 10^{-1}]$	8.56
	(10,0.001)		$1.13 \times 10^{+0}$	$[1.31 \times 10^{+0}]$	9.95
F_5	(10,0.01)		6.37×10^{-1}	$[2.27 \times 10^{-1}]$	2.29
	(200,0.001)		$8.47 \times 10^{+0}$	$[3.14 \times 10^{+0}]$	1.71
F_6	(100,0.1)		5.71×10^{-2}	$[3.06 \times 10^{-2}]$	2.40
	(10,0.001)		2.49×10^{-1}	$[1.26 \times 10^{-1}]$	1.96
F_7	(50,0.01)		$1.24 \times 10^{+0}$	$[6.72 \times 10^{-1}]$	3.43
	(10,0.001)		$3.20 \times 10^{+0}$	$[1.54 \times 10^{+0}]$	1.86
F_8	(10,0.01)	▶	$1.45 \times 10^{+2}$	$[1.14 \times 10^{+1}]$	2.50
	(100,0.1)		$1.76 \times 10^{+2}$	$[1.50 \times 10^{+1}]$	2.49
F_9	(10,0.001)	▶	$6.62 \times 10^{+1}$	$[4.37 \times 10^{+0}]$	2.87
	(200,0.01)		$1.10 \times 10^{+2}$	$[2.82 \times 10^{+0}]$	4.26
F_{10}	(10,0.01)		$6.10 \times 10^{+1}$	$[4.63 \times 10^{+0}]$	2.73
	(200,0.1)		$1.22 \times 10^{+2}$	$[4.51 \times 10^{+0}]$	2.00

TAB. 6.5 – Performances obtenue par BSC disposant de 1 000 évaluations avec les paramètres correspondants. ▶ signale le meilleur résultat obtenu pour une fonction parmi les quatre algorithmes, σ correspond à l'écart type et F au second coefficient de Fisher.

Les meilleurs résultats obtenus pour chaque méthode disposant de 10 000 évaluations sont donnés dans le tableau 6.9. Les meilleurs résultats sont obtenus dans tous les cas soit par PBIL, soit par ACS_b. Cela peut être dû au nombre de combinaisons de paramètres testées qui a été beaucoup plus important pour ces deux méthodes. Il ressort tout de même que ACS_b profite d'un nombre d'évaluations supérieur puisque pour 1 000 évaluations il obtenait le meilleur minimum pour 2 fonctions sur les 11 alors qu'avec 10 000 évaluations, le meilleur minimum a pratiquement été trouvé pour la moitié des fonctions par ACS_b.

F_i	$\text{Param}_{\min}(n,LR,p_m,LR_{neg},\delta_m)$ $\text{Param}_{\max}(n,LR,p_m,LR_{neg},\delta_m)$		Min Max	$[\sigma_{\min}]$ $[\sigma_{\max}]$	$F_{\min}$ $F_{\max}$
F_0	(10,0.1,0.1,0.01,0.01)		$1.81 \times 10^{+1}$	$[1.78 \times 10^{+0}]$	1.99
	(100,0.01,0.001,0.001,0.01)		$6.20 \times 10^{+1}$	$[6.55 \times 10^{+0}]$	2.94
F_1	(10,0.05,0.1,0.1,0.01)	▶	3.68×10^{-4}	$[2.29 \times 10^{-4}]$	4.46
	(10,0.01,0.001,0.01,0.001)		4.77×10^{-1}	$[2.81 \times 10^{-1}]$	3.76
F_2	(10,0.05,0.1,0.1,0.1)		1.01×10^{-2}	$[1.53 \times 10^{-2}]$	8.44
	(10,0.1,0.001,0.1,0.01)		1.57×10^{-1}	$[2.50 \times 10^{-1}]$	9.17
F_3	(10,0.1,0.1,0.1,0.01)	▶	$5.86 \times 10^{+0}$	$[3.29 \times 10^{+0}]$	2.75
	(100,0.05,0.01,0.001,0.1)		$2.75 \times 10^{+1}$	$[3.81 \times 10^{+0}]$	2.49
F_4	(10,0.05,0.001,0.1,0.1)	▶	5.88×10^{-2}	$[3.20 \times 10^{-2}]$	2.55
	(100,0.01,0.001,0.01,0.01)		6.81×10^{-1}	$[2.54 \times 10^{-1}]$	1.72
F_5	(10,0.1,0.1,0.1,0.01)	▶	4.22×10^{-1}	$[1.98 \times 10^{-1}]$	3.38
	(50,0.01,0.001,0.01,0.001)		$9.72 \times 10^{+0}$	$[3.29 \times 10^{+0}]$	2.65
F_6	(10,0.1,0.1,0.1,0.1)		1.36×10^{-2}	$[2.15 \times 10^{-2}]$	20.26
	(10,0.01,0.1,0.001,0.1)		1.00×10^{-1}	$[4.81 \times 10^{-2}]$	2.39
F_7	(10,0.05,0.1,0.1,0.01)	▶	5.39×10^{-1}	$[4.10 \times 10^{-1}]$	1.31
	(50,0.01,0.001,0.001,0.1)		$2.49 \times 10^{+0}$	$[5.44 \times 10^{-1}]$	2.48
F_8	(10,0.1,0.01,0.001,0.01)		$1.47 \times 10^{+2}$	$[1.62 \times 10^{+1}]$	2.80
	(200,0.01,0.1,0.01,0.001)		$1.81 \times 10^{+2}$	$[1.11 \times 10^{+1}]$	2.19
F_9	(10,0.1,0.1,0.1,0.001)		$7.59 \times 10^{+1}$	$[4.79 \times 10^{+0}]$	2.78
	(200,0.05,0.001,0.01,0.1)		$1.11 \times 10^{+2}$	$[2.52 \times 10^{+0}]$	4.49
F_{10}	(10,0.1,0.01,0.1,0.001)	▶	$5.12 \times 10^{+1}$	$[3.40 \times 10^{+0}]$	3.60
	(200,0.01,0.001,0.001,0.1)		$1.31 \times 10^{+2}$	$[2.56 \times 10^{+0}]$	2.49

TAB. 6.6 – Performances obtenue par PBIL disposant de 1 000 évaluations avec les paramètres correspondants. ▶ signale le meilleur résultat obtenu pour une fonction parmi les quatre algorithmes, σ correspond à l'écart type et F au second coefficient de Fisher.

Comme chaque résultat a pu être obtenu avec des paramètres différents, nous allons tenter d'estimer la dépendance de chaque méthode quant à son paramètrage.

Principes

L'interprétation de ces résultats est un problème à part entière. Rappelons que l'objectif est de déterminer un jeu de paramètre satisfaisant pour le jeu de test que nous nous sommes donnés. Pour chaque méthode nous avons pris le jeu de paramètre p qui minimise la quantité :

$$\sum_{i=0}^{10} \sum_{T_3 \in \{1\,000,10\,000,100\,000\}} \frac{F_i^+(p,T_3)}{\max_j\{F_i^+(p_j,T_3)\} - \min_j\{F_i^+(p_j,T_3)\}} \tag{6.19}$$

où $F_i^+(p,T_3)$ représente le minimum obtenu par l'algorithme considéré sur la fonction F_i avec le jeu de paramètre p et disposant de T_3 évaluations.

Le tableau 6.10 donne les jeux de paramètres retenus par cette méthode.

F_i	$\text{Param}_{\min}(n,\rho)$ $\text{Param}_{\max}(n,\rho)$	Min Max	$[\sigma_{\min}]$ $[\sigma_{\max}]$	$F_{\min}$ $F_{\max}$
F_0	(10,0.1)	$5.13 \times 10^{+1}$	$[8.19 \times 10^{+0}]$	2.81
	(200,0.01)	$5.88 \times 10^{+1}$	$[7.58 \times 10^{+0}]$	2.21
F_1	(10,0.1)	9.45×10^{-3}	$[2.05 \times 10^{-2}]$	22.74
	(200,0.01)	3.90×10^{-1}	$[2.63 \times 10^{-1}]$	2.57
F_2	(100,0.01)	1.94×10^{-2}	$[2.57 \times 10^{-2}]$	15.16
	(10,0.1)	2.38×10^{-1}	$[2.76 \times 10^{-1}]$	2.59
F_3	(200,0.01)	$2.38 \times 10^{+1}$	$[4.90 \times 10^{+0}]$	3.21
	(10,0.01)	$2.84 \times 10^{+1}$	$[4.80 \times 10^{+0}]$	2.18
F_4	(10,0.1)	9.60×10^{-2}	$[6.37 \times 10^{-2}]$	9.97
	(200,0.1)	6.15×10^{-1}	$[3.16 \times 10^{-1}]$	3.36
F_5	(10,0.1)	$3.14 \times 10^{+0}$	$[2.91 \times 10^{+0}]$	5.45
	(50,0.05)	$9.17 \times 10^{+0}$	$[3.19 \times 10^{+0}]$	2.40
F_6	(50,0.001)	6.31×10^{-2}	$[3.89 \times 10^{-2}]$	4.54
	(200,0.05)	9.03×10^{-2}	$[4.70 \times 10^{-2}]$	3.37
F_7	(10,0.1)	$1.84 \times 10^{+0}$	$[6.58 \times 10^{-1}]$	3.63
	(200,0.05)	$2.45 \times 10^{+0}$	$[3.79 \times 10^{-1}]$	2.29
F_8	(10,0.05)	$1.68 \times 10^{+2}$	$[1.26 \times 10^{+1}]$	3.14
	(100,0.1)	$1.79 \times 10^{+2}$	$[1.28 \times 10^{+1}]$	3.18
F_9	(100,0.1)	$1.09 \times 10^{+2}$	$[3.79 \times 10^{+0}]$	6.18
	(200,0.05)	$1.11 \times 10^{+2}$	$[2.53 \times 10^{+0}]$	3.07
F_{10}	(10,0.1)	$1.27 \times 10^{+2}$	$[4.98 \times 10^{+0}]$	2.00
	(200,0.01)	$1.31 \times 10^{+2}$	$[3.45 \times 10^{+0}]$	2.71

TAB. 6.7 – Performances obtenue par AS_b disposant de 1 000 évaluations avec les paramètres correspondants. ▶ signale le meilleur résultat obtenu pour une fonction parmi les quatre algorithmes, σ correspond à l'écart type et F au second coefficient de Fisher.

Résultats

Chaque algorithme a été lancé avec 10 000 évaluations de la fonction et les paramètres du tableau 6.10. Le tableau 6.11, page 115, donne les résultats obtenus en moyenne sur 30 essais.

Le choix du paramétrage qui a été fait désavantage fortement ACS_b qui obtient le meilleur minimum pour une seule fonction. Cela a principalement profité à BSC qui obtient le meilleur résultat pour trois fonctions alors que dans le tableau 6.9, BSC ne s'était montré compétitif pour aucune fonction. Cela peut signifier que la méthode de choix des jeux de paramètres n'est pas adaptée à ACS_b ou alors que ACS_b est plus sensible à ses paramètres que les autres méthodes.

Les figures 6.2 et 6.3 présentent pour chaque méthode et chaque fonction, la courbe d'entropie ($E(t)$) obtenue lors des tests du tableau 6.11. En abscisse, le nombre d'itérations est variable suivant la méthode. Ceci provient de la taille de la population utilisée : pour BSC et PBIL, 50 chaînes binaires sont générées à chaque itération alors que pour AS_b et ACS_b seulement 10 sont utilisées.

Deux points sont à noter concernant ces courbes :

F_i	$\text{Param}_{\min}(n,\rho,\alpha,q_0)$ $\text{Param}_{\max}(n,\rho,\alpha,q_0)$		Min Max	$[\sigma_{\min}]$ $[\sigma_{\max}]$	$F_{\min}$ $F_{\max}$
F_0	(200,0.01,0.1,0.7)		$3.08 \times 10^{+1}$	$[7.46 \times 10^{+0}]$	2.37
	(10,0.1,0.05,0.9)		$7.08 \times 10^{+1}$	$[1.23 \times 10^{+1}]$	3.15
F_1	(10,0.1,0.05,0.5)		4.48×10^{-4}	$[2.56 \times 10^{-4}]$	9.00
	(10,0.1,0.1,0.9)		$1.16 \times 10^{+0}$	$[1.64 \times 10^{+0}]$	2.71
F_2	(100,0.1,0.1,0.5)	►	7.42×10^{-3}	$[1.22 \times 10^{-2}]$	11.47
	(10,0.1,0.05,0.9)		$1.32 \times 10^{+0}$	$[2.83 \times 10^{+0}]$	12.60
F_3	(200,0.001,0.1,0.7)		$1.10 \times 10^{+1}$	$[3.36 \times 10^{+0}]$	2.19
	(10,0.1,0.01,0.9)		$3.23 \times 10^{+1}$	$[8.25 \times 10^{+0}]$	3.05
F_4	(50,0.001,0.05,0.7)		9.49×10^{-2}	$[5.08 \times 10^{-2}]$	3.35
	(10,0.1,0.1,0.9)		$2.13 \times 10^{+0}$	$[2.20 \times 10^{+0}]$	2.77
F_5	(100,0.001,0.1,0.7)		$1.50 \times 10^{+0}$	$[7.18 \times 10^{-1}]$	6.46
	(10,0.1,0.05,0.9)		$1.98 \times 10^{+1}$	$[1.19 \times 10^{+1}]$	2.55
F_6	(10,0.01,0.001,0.7)	►	7.75×10^{-3}	$[5.66 \times 10^{-3}]$	2.34
	(10,0.1,0.01,0.9)		2.31×10^{-1}	$[1.13 \times 10^{-1}]$	2.56
F_7	(100,0.001,0.05,0.7)		6.24×10^{-1}	$[4.23 \times 10^{-1}]$	1.88
	(10,0.1,0.05,0.9)		$4.66 \times 10^{+0}$	$[1.68 \times 10^{+0}]$	2.61
F_8	(200,0.01,0.1,0.9)		$1.47 \times 10^{+2}$	$[1.20 \times 10^{+1}]$	5.52
	(10,0.1,0.01,0.9)		$2.15 \times 10^{+2}$	$[4.49 \times 10^{+1}]$	3.48
F_9	(200,0.1,0.05,0.9)		$1.01 \times 10^{+2}$	$[4.25 \times 10^{+0}]$	2.61
	(10,0.1,0.01,0.9)		$1.14 \times 10^{+2}$	$[5.17 \times 10^{+0}]$	3.03
F_{10}	(200,0.1,0.1,0.5)		$1.16 \times 10^{+2}$	$[4.19 \times 10^{+0}]$	3.23
	(10,0.1,0.05,0.9)		$1.38 \times 10^{+2}$	$[5.50 \times 10^{+0}]$	2.62

TAB. 6.8 – Performances obtenue par ACS_b disposant de 1 000 évaluations avec les paramètres correspondants. ► signale le meilleur résultat obtenu pour une fonction parmi les quatre algorithmes, σ correspond à l'écart type et F au second coefficient de Fisher.

- pour les méthodes BSC, PBIL et ACS_b, on distingue deux groupes de fonctions. Le premier groupe concerne les fonctions produisant une diminution rapide de l'entropie (F_0 à F_7) par opposition au groupe de fonctions (F_8, F_9 et F_{10}) produisant une diminution lente de l'entropie. Il est intéressant de noter que les fonctions comportant un nombre important de dimensions provoquent une diminution plus lente du désordre du vecteur de probabilité V que les autres fonctions ;
- les courbes obtenues pour AS_b sont différentes des autres méthodes et ceci de façon marquée. La diminution d'entropie, est beaucoup moins rapide pour AS_b que pour ACS_b et on ne retrouve pas de façon aussi nette les deux groupes de fonctions.

En dehors des problèmes de paramétrage on peut s'intéresser au temps de calcul consommé par chaque méthode. Le tableau 6.12 donne les temps de calcul pour chaque méthode et chaque fonction pour les exécutions qui ont donné les résulats du tableau 6.11. BSC et PBIL sont plus rapides que AS_b et ACS_b mais l'augmentation du nombre de bits est plus pénalisante pour BSC que les autres méthodes. Ces conclusions

F_i	BSC		PBIL		AS_b		ACS_b
F_0	$1.76 \times 10^{+1}$		$1.58 \times 10^{+1}$		$1.97 \times 10^{+1}$	▶	$1.38 \times 10^{+1}$
F_1	3.34×10^{-4}		2.94×10^{-4}		4.88×10^{-4}	▶	2.15×10^{-4}
F_2	1.45×10^{-3}		2.41×10^{-4}		2.61×10^{-3}	▶	2.08×10^{-4}
F_3	$4.51 \times 10^{+0}$	▶	$1.41 \times 10^{+0}$		$5.53 \times 10^{+0}$		$2.19 \times 10^{+0}$
F_4	3.59×10^{-2}	▶	3.04×10^{-2}	▶	3.04×10^{-2}		3.85×10^{-2}
F_5	2.08×10^{-1}	▶	1.47×10^{-1}		2.67×10^{-1}		1.96×10^{-1}
F_6	1.10×10^{-2}		6.52×10^{-3}		1.76×10^{-2}	▶	5.87×10^{-3}
F_7	5.80×10^{-1}		4.48×10^{-1}		5.99×10^{-1}	▶	3.37×10^{-1}
F_8	$1.04 \times 10^{+2}$	▶	$8.71 \times 10^{+1}$		$1.27 \times 10^{+2}$		$9.54 \times 10^{+1}$
F_9	$3.54 \times 10^{+1}$	▶	$2.90 \times 10^{+1}$		$9.52 \times 10^{+1}$		$5.76 \times 10^{+1}$
F_{10}	$1.45 \times 10^{+1}$	▶	$9.94 \times 10^{+0}$		$1.02 \times 10^{+2}$		$4.66 \times 10^{+1}$

TAB. 6.9 – Meilleurs résultats obtenus pour chaque algorithme avec 10 000 évaluations. ▶ signale pour une fonction donnée le meilleur résultats trouvé parmi les quatre algorithmes.

Algorithme (paramètres)	valeurs des paramètres
BSC (n,p_m)	(50,0.001)
PBIL (n,LR,p_m,LR_{neg},δ_m)	(50,0.1,0.01,0.1,0.1)
AS_b (n,ρ)	(10,0.05)
ACS_b (n,ρ,α,q_0)	(10,0.1,0.001,0.0)

TAB. 6.10 – Valeurs des paramètres retenues pour chaque algorithme.

sont à modérer par le fait que les populations utilisées ne sont pas de même taille et cela a une influence sur le nombre d'opérations effectuées par chaque méthode (seul le nombre d'évaluations de la fonction est constant d'une méthode à l'autre).

Pour les quatre algorithmes considérés, on constate globalement que les deux méthodes les plus « perfectionnées » (PBIL et ACS_b) obtiennent de meilleurs résultats que les deux autres quand leur paramètres sont bien choisis. Mais du fait de leur perfectionnement, le nombre de paramètres est plus important et cela peut poser des difficultés quand on désire leur donner des valeurs « universelles », du moins pour le jeu de tests que nous considérons.

F_i		BSC	PBIL	AS_b	ACS_b
F_0	Min	$1.96 \times 10^{+1}$	▶ $1.75 \times 10^{+1}$	$2.00 \times 10^{+1}$	▶ $1.75 \times 10^{+1}$
	$[\sigma]$	$[9.52 \times 10^{-1}]$	$[1.84 \times 10^{+0}]$	$[5.16 \times 10^{-1}]$	$[2.95 \times 10^{+0}]$
	F	7.90	3.96	15.00	2.31
F_1	Min	▶ 3.48×10^{-4}	4.34×10^{-4}	1.60×10^{-3}	8.33×10^{-3}
	$[\sigma]$	$[1.82 \times 10^{-4}]$	$[1.66 \times 10^{-4}]$	$[4.99 \times 10^{-3}]$	$[2.09 \times 10^{-2}]$
	F	2.28	2.05	27.10	20.45
F_2	Min	4.16×10^{-2}	▶ 3.21×10^{-3}	1.60×10^{-1}	1.41×10^{-1}
	$[\sigma]$	$[6.28 \times 10^{-2}]$	$[9.11 \times 10^{-3}]$	$[1.97 \times 10^{-1}]$	$[1.95 \times 10^{-1}]$
	F	6.09	24.57	3.30	6.14
F_3	Min	$8.94 \times 10^{+0}$	▶ $3.26 \times 10^{+0}$	$5.60 \times 10^{+0}$	$7.59 \times 10^{+0}$
	$[\sigma]$	$[3.12 \times 10^{+0}]$	$[1.79 \times 10^{+0}]$	$[2.33 \times 10^{+0}]$	$[3.21 \times 10^{+0}]$
	F	2.75	2.80	6.30	2.56
F_4	Min	1.45×10^{-1}	7.61×10^{-2}	▶ 4.76×10^{-2}	1.84×10^{-1}
	$[\sigma]$	$[1.66 \times 10^{-1}]$	$[4.66 \times 10^{-2}]$	$[2.47 \times 10^{-2}]$	$[1.00 \times 10^{-1}]$
	F	10.62	6.66	4.44	2.52
F_5	Min	4.96×10^{-1}	▶ 2.24×10^{-1}	2.89×10^{-1}	8.51×10^{-1}
	$[\sigma]$	$[2.53 \times 10^{-1}]$	$[7.81 \times 10^{-2}]$	$[9.17 \times 10^{-2}]$	$[3.24 \times 10^{-1}]$
	F	4.95	2.78	2.35	3.03
F_6	Min	5.48×10^{-2}	▶ 1.28×10^{-2}	2.20×10^{-2}	4.44×10^{-2}
	$[\sigma]$	$[5.23 \times 10^{-2}]$	$[1.46 \times 10^{-2}]$	$[2.36 \times 10^{-2}]$	$[5.10 \times 10^{-2}]$
	F	5.04	14.38	13.37	3.95
F_7	Min	9.10×10^{-1}	▶ 6.18×10^{-1}	7.48×10^{-1}	$1.11 \times 10^{+0}$
	$[\sigma]$	$[3.87 \times 10^{-1}]$	$[3.73 \times 10^{-1}]$	$[3.28 \times 10^{-1}]$	$[3.03 \times 10^{-1}]$
	F	4.20	2.11	5.46	4.25
F_8	Min	$1.11 \times 10^{+2}$	$1.07 \times 10^{+2}$	$1.49 \times 10^{+2}$	▶ $9.72 \times 10^{+1}$
	$[\sigma]$	$[1.02 \times 10^{+1}]$	$[7.09 \times 10^{+0}]$	$[7.90 \times 10^{+0}]$	$[8.99 \times 10^{+0}]$
	F	1.98	2.66	4.07	2.35
F_9	Min	▶ $3.59 \times 10^{+1}$	$3.68 \times 10^{+1}$	$1.02 \times 10^{+2}$	$5.73 \times 10^{+1}$
	$[\sigma]$	$[2.52 \times 10^{+0}]$	$[2.38 \times 10^{+0}]$	$[3.44 \times 10^{+0}]$	$[4.14 \times 10^{+0}]$
	F	2.69	2.63	3.36	2.83
F_{10}	Min	▶ $1.42 \times 10^{+1}$	$1.79 \times 10^{+1}$	$1.05 \times 10^{+2}$	$4.52 \times 10^{+1}$
	$[\sigma]$	$[1.38 \times 10^{+0}]$	$[1.94 \times 10^{+0}]$	$[4.82 \times 10^{+0}]$	$[5.10 \times 10^{+0}]$
	F	2.32	3.64	2.38	2.76

TAB. 6.11 – Résultats obtenus pour chaque fonction et chaque algorithme en utilisant les paramètres du tableau 6.10. La valeur indiquée correspond au minimum moyen obtenu sur 30 essais quand la méthode dispose de 10 000 évaluations de la fonction considérée, σ est l'écart type et F le coefficient de Fischer. ▶ signale le meilleur résultat pour chaque fonction.

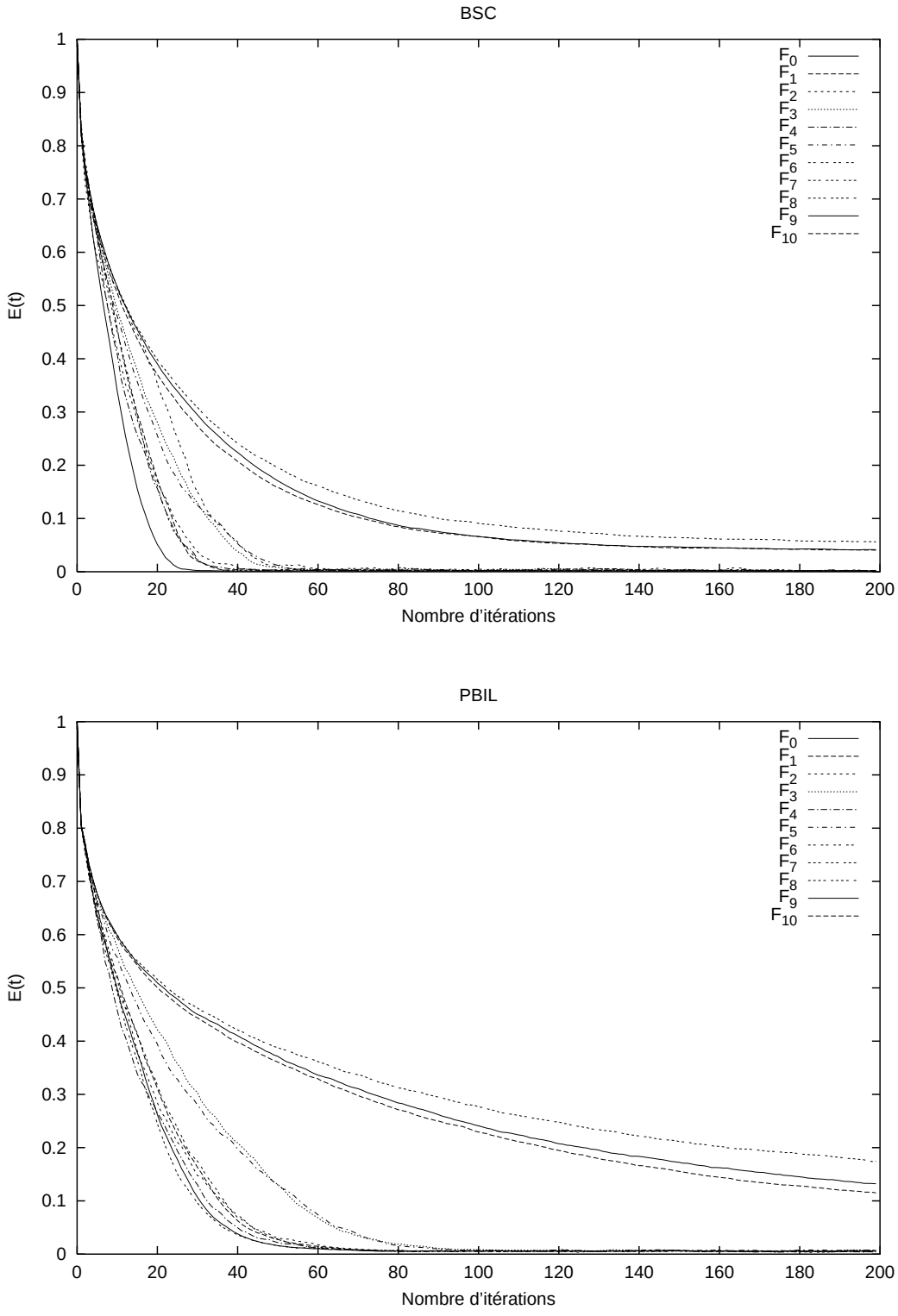


FIG. 6.2 – Courbes de d'entropie (E(t)) pour BSC et PBIL.

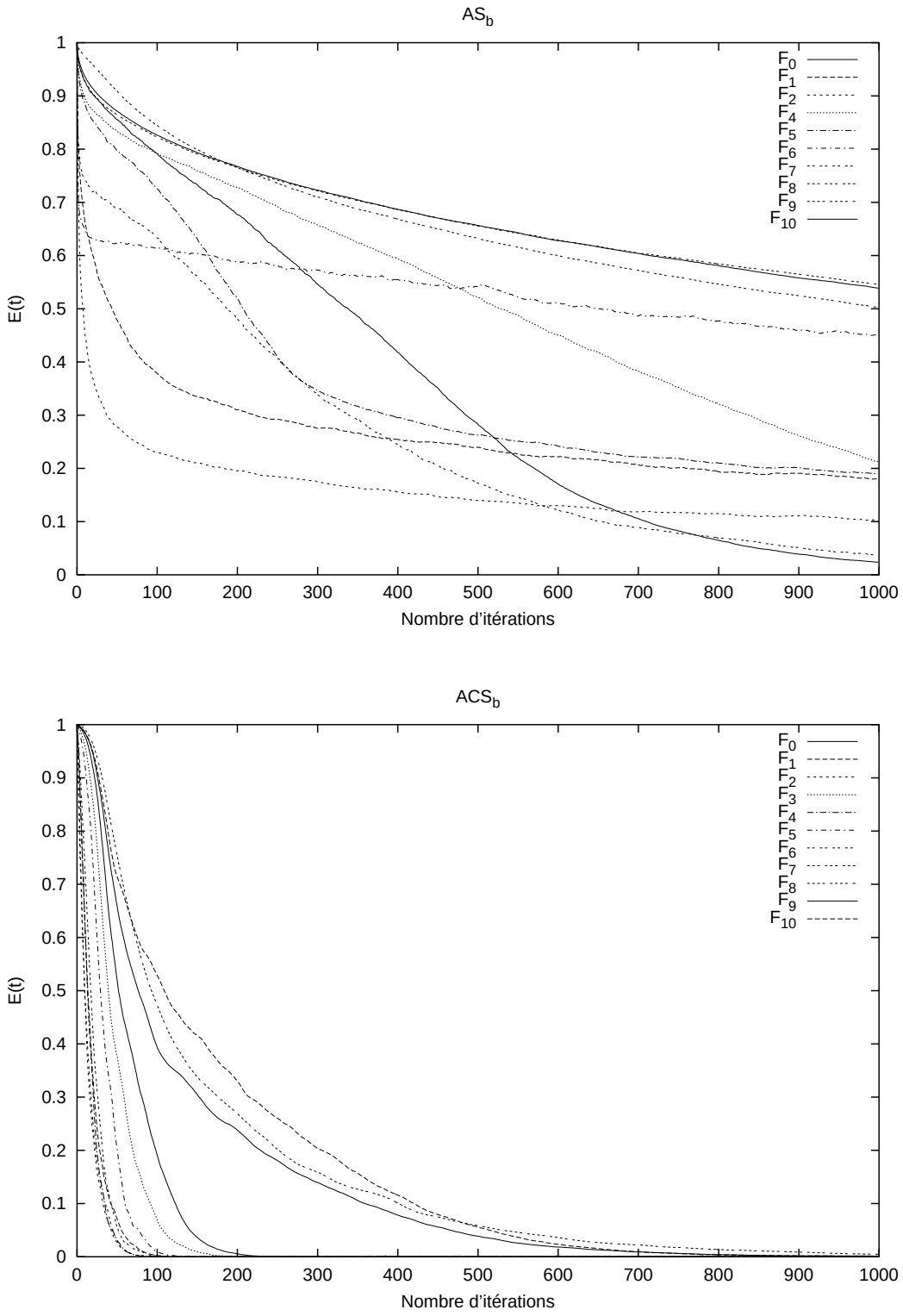


FIG. 6.3 – Courbes de d'entropie (E(t)) pour AS_b et ACS_b.

F_i	BSC	PBIL	AS_b	ACS_b
F_0	0.09	0.12	0.17	0.17
F_1	0.10	0.14	0.17	0.18
F_2	0.08	0.11	0.13	0.14
F_3	0.18	0.21	0.28	0.27
F_4	0.09	0.12	0.15	0.15
F_5	0.19	0.21	0.28	0.28
F_6	0.12	0.15	0.18	0.17
F_7	0.12	0.15	0.18	0.18
F_8	3.06	2.95	4.09	4.38
F_9	3.41	3.31	4.45	4.74
F_{10}	3.11	3.02	4.15	4.43

TAB. 6.12 – Temps de calcul moyen, en seconde, pour chaque méthode. À titre indicatif, ces résultats sont obtenus sur une machine équipée d'un processeur Pentium III à 450 MHz.

6.8 Extensions

La manière la plus intuitive d'étendre ces méthodes est d'augmenter la quantité d'informations. Deux possibilités s'offrent à nous :

- augmenter la quantité d'informations stockées par V. On peut par exemple modifier l'adaptation de ACO en adoptant la représentation donnée par la figure 6.4.a. La figure 6.4.b représente la solution $011\ldots10$ obtenue en parcourant le graphe du nœud fictif de départ D au nœud fictif d'arrivée A. Dans ce cas, on ne peut plus représenter l'information stockée par les phéromones déposées sur les arcs par un vecteur V de probabilité : Le choix d'un sommet « 1 » ou « 0 » dépend du choix effectué pour le sommet précédent.
  On peut envisager des représentations plus complexes, et par là même abandonner la notion de phéromone, en utilisant les chaînes de Markov cachées pour générer les séquences binaires. On dispose alors de méthodes plus sophistiquées pour réaliser l'apprentissage de cette information. L'algorithme de Baum-Welch en est un exemple ;
- diviser l'information. Au lieu de manipuler un seul vecteur V, on peut parfaitement envisager de faire travailler les algorithmes présentés sur K sous-populations, chacune « centrée » autour d'un vecteur V_i. On reproduirait ainsi un modèle très prisé par les algorithmes génétiques : l'« island model » où plusieurs populations évoluent parallèlement tout en faisant quelques échanges de génomes (Mühlenbein et al., 1991). Ce type d'extension a déjà aussi été étudié dans le domaine des fourmis artificielles avec l'utilisation de différents types de phéromones (Kawamura et al., 1998).

Les travaux concernants des modèles plus complexes, notamment pour traiter des problèmes où les variables ne sont pas indépendantes, sont résumés dans (Pelikan et al., 1999).

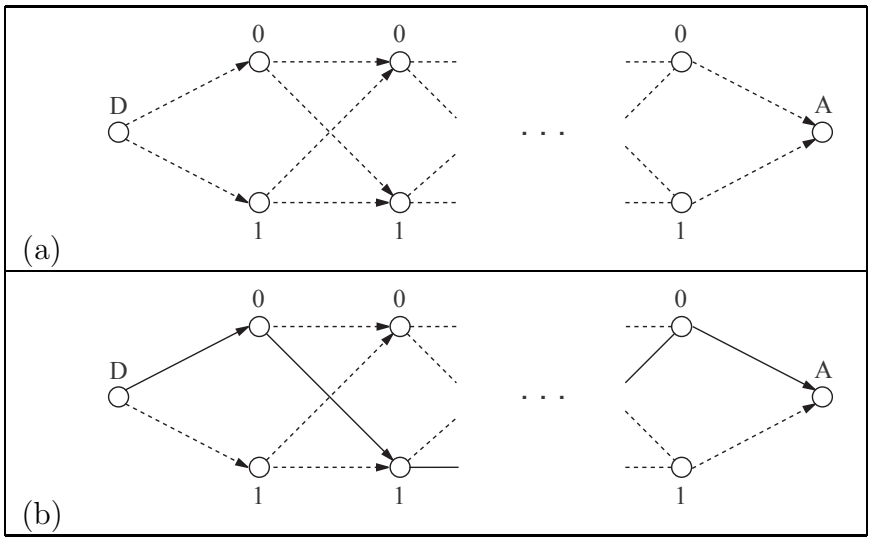


FIG. 6.4 – Extension de ACO appliqué au problème d'optimisation binaire. Le choix des l bits est modélisé par le choix d'un sommet « 0 » ou « 1 » entre les noeuds D (Départ) et A (Arrivée).

Le deuxième type d'extension concerne certaines améliorations que l'on peut apporter simplement aux algorithmes exposés :

- le choix qui est fait par ACS pour la génération des solutions peut être raffiné : la probabilité d'effectuer une exploration ou une exploitation dépend de la probabilité q_0 donnée en paramètre. On peut envisager de faire varier cette probabilité au cours du temps : par exemple en fixant $q_0 = \log(T)/\log(T_3)$ où T représente le numéro de l'itération courante. Nous avons déjà expérimenté cette méthode qui a donné des résultats concluants pour le problème du flowshop bicritère (T'Kindt et al., 2000) ;
- l'initialisation de V peut être faite par un ensemble composé de bonnes solutions construites à l'aide d'heuristiques et de solutions aléatoires garantissant une distribution des solutions initiales dans tout l'espace de recherche ;
- d'une manière générale, la difficulté de fixer les paramètres de chaque méthode peut être contre-balancée de plusieurs manières :
  - en faisant varier les paramètres au cours du déroulement de l'algorithme de façon prédéterminée (déterministe),
  - en utilisant les résultats du processus de recherche (*feedback*) (adaptation),
  - en intégrant au problème d'optimisation le choix des paramètres (auto-adaptation).

  Certains auteurs proposent de classifier les stratégies d'adaptation en plusieurs groupes (Eiben et al., 1998b) :
  - selon le type d'adaptation :
    - statique,
    - dynamique (déterministe, adaptatif ou auto-adaptatif).
  - selon le niveau d'adaptation :
    - sur l'environnment,

– sur la population,
– sur un individu,
– sur un composant.

Enfin, il est possible de mixer les méthodes entre elles. C'est une voie qui est étudiée notamment par Cordón, Herrera et Moreno de l'université de Grenade en adaptant les principes de PBIL à AS pour le problème du voyageur de commerce et de l'assignement quadratique (Cordón et al., 1999).

6.9 Conclusion

L'objectif de ce chapitre était de montrer que différentes méthodes d'optimisation se comportaient de façon très similaire : elles utilisent toutes un vecteur de probabilité échantillonnant un espace de recherche binaire à la recherche d'un optimum global. D'un côté les algorithmes issus des méthodes évolutionnaires, tels que BSC et PBIL, manipulent de façon explicite les statistiques que les algorithmes génétiques manipulent de façon implicite. D'un autre côté les algorithmes inspirés des colonies de fourmis qui mettent en avant l'apprentissage collectif d'une information distribuée : les phéromones.

Les résultats obtenus permettent de confirmer les similitudes observées dans les définitions des méthodes. La présentation de ces méthodes ainsi que les résultats obtenus nous font penser qu'un certain nombre d'études plus théoriques permettraient de proposer un modèle rassemblant les avantages et écartant les inconvénients des quatres algorithmes.

Les possibilités d'extension sont grandes, par exemple en abandonnant le codage binaire, considéré à raison comme inefficace pour la plupart des problèmes d'optimisation numérique[8]. Dans le chapitre suivant, nous présentons une nouvelle heuristique inspirée des colonies de fourmis s'adaptant à différents types de représentation. Nous n'utiliserons alors plus de codage binaire.

Les travaux présentés dans ce chapitre ont été menés en collaboration avec Guillaume Dromel, Ameur Soukhal, Laëtitia Desbarats (Laboratoire d'Informatique), et Eric Ramat (Université du Litoral) (Dromel, 1999; Monmarché et al., 1999a; Monmarché et al., 2000a)

[8]Par exemple : un problème à 100 variables, toutes définies sur l'intervalle $[-500, 500]$, avec une précision de six chiffres après la virgule, nécessite d'utiliser des chaînes binaires de 3 000 bits.

Chapitre 7

L'algorithme API

Nous présentons dans ce chapitre une modélisation du comportement de fourragement d'une population de fourmis primitives (*Pachycondyla apicalis*) et son application au problème général d'optimisation. Ces fourmis sont caractérisées par une stratégie de recherche de proie relativement simple où les individus chassent en solitaire et tentent de couvrir uniformément un espace donné autour de leur nid par des recherches locales sur des sites de chasse. Le nid peut être déménagé périodiquement. Cela correspond en optimisation à un algorithme effectuant plusieurs recherches aléatoires en parallèle et localisées uniformément dans un sous-espace centré en un point. Le déplacement du nid correspond à un opérateur de réinitialisation dans les recherches parallèles où le point central est déplacé. Nous testons ce modèle, appelé API, sur un ensemble de problèmes d'optimisation combinatoire et numérique.

7.1 Introduction

Ce chapitre présente les travaux menés autour de la modélisation des fourmis de l'espèce *Pachycondyla apicalis*. Cette étude trouve son origine dans les travaux de Dominique Fresneau sur la stratégie de fourragement originale de cette espèce de fourmis ponérines (Fresneau, 1985; Fresneau, 1994). Nous commencerons donc par présenter les caractéristiques biologiques qui ont pu être exploitées du point de vue d'une modélisation algorithmique avec comme objectif de l'appliquer à plusieurs problèmes d'optimisation.

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 rechercher leurs proies. A partir de leur nid, elles couvrent globalement une surface donnée en la partitionnant en sites de chasse individuels. Pour une fourmi donnée, on observe une stratégie d'exploration aléatoire des sites sensible au succès rencontré. Ces principes peuvent être repris pour résoudre un problème analogue qui est la recherche d'un minimum global, par exemple d'une fonction f définie de $\mathbb{R}^l$ dans $\mathbb{R}$.

La suite de ce chapitre est organisée de la façon suivante : la section 7.2 décrit de manière succincte les principes utilisés dans la recherche de proies de *Pachycondyla apicalis*. La section 7.3 donne une modélisation possible de ces principes, à la fois du point de vue mathématique de l'optimisation que du point de vue algorithmique de l'informatique. Un certain nombre d'extensions tirées du comportement naturel des fourmis sont proposées dans la section 7.4 afin d'enrichir le modèle. La section 7.5 propose une étude expérimentale des principaux paramètres du modèle sur le problème de l'optimisation de fonctions numériques. Les propriétés fondamentales de ce nouvel algorithme, appelé API, sont également passées en revue, et notamment ses liens avec d'autres méthodes d'optimisation (section 7.6). La section 7.7 décrit les tests expérimentaux qui ont été réalisés sur des problèmes d'optimisation numérique (optimisation de fonctions numériques, apprentissage de Chaînes de Markov Cachées, apprentissage d'un Réseau de Neurones Artificiels) ainsi que sur un problème d'optimisation combinatoire classique (le problème du Voyageur de Commerce (PVC)). Puis nous analyserons les faiblesses et les atouts de API et présenterons les perspectives possibles (sections 7.8 et 7.9).

7.2 Biologie de *Pachycondyla apicalis*

Cette section donne un aperçu de la biologie de *Pachycondyla apicalis* et notamment de leur stratégie de recherche de nourriture (le fourragement). Une étude très complète leur est consacrée dans (Fresneau, 1994). *Pachycondyla apicalis* est une ponérine néotropicale que l'on rencontre en Amérique du Sud, en particulier au Mexique. 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).

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. Le comportement de recherche de nourriture n'utilise pas de comportement collectif direct. 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. Elles ne mettent donc pas en œuvre des comportements de recrutement de masse.

On peut supposer que la nature des proies recherchées n'encourage pas spécialement ce genre de communication inter-individus : 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 (Hölldobler and Wilson, 1990). 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 de fourrageuses qui décroît évidemment quand la distance au 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 7.1 présente les aires de fourragement d'une colonie *Pachycondyla apicalis*.

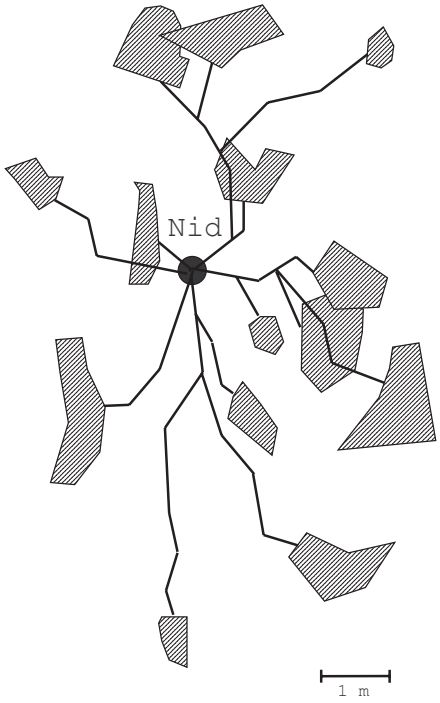


FIG. 7.1 – Exemple (fictif) de carte des trajets et aires de récolte des fourrageuses (inspiré de (Fresneau, 1994)).

Le comportement de fourragement de *Pachycondyla apicalis* peut être résumé en trois règles (Fresneau, 1994) :

1. 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 ;

2. la découverte d'une proie pèse sur la décision de sortie des fourrageuses en réduisant l'intervalle de temps qu'elles passent au nid ;
3. les fourrageuses semblent progressivement apprendre une association entre une direction de recherche opposée au nid et l'augmentation de la probabilité de succès.

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 sinueux ;
- 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 (en moyenne quatre fois) ;
- 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 interfèreraient 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 proie ou que le nid a changé 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 : 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.

Dans la réalité, l'efficacité de la stratégie n'est pas parfaite si on considère le nombre de proies trouvées relativement au nombre de proies présentes. Cependant, la taille d'une colonie de *Pachycondyla apicalis* étant relativement réduite, les besoins en nourriture sont relativement modestes. Il semble que l'adaptation de ces fourmis ne réside pas seulement dans leur comportement de fourragement mais aussi dans le maintien de colonies peu populeuses. Ceci permet aux *Pachycondyla apicalis* de survivre dans des secteurs comportant de nombreux prédateurs concurrents et de ne pas nécessiter une grande quantité d'insectes.

7.3 Modélisation algorithmique

Nous présentons l'adaptation de la stratégie de fourragement des *Pachycondyla apicalis* pour la résolution de problèmes d'optimisation. Cette section démontre la généricité de la méthode relativement à l'espace de recherche. Nous montrerons l'influence des différents paramètres puis un certain nombre d'applications dans les sections suivantes.

La modélisation que nous proposons est assez différente de celle proposée par Deneubourg et ses collègues (Deneubourg et al., 1987) puisqu'elle a été développée indépendamment et qu'elle s'applique au problème d'optimisation. Nous en discuterons dans la section7.4.3.

7.3.1 Espace de recherche et fonction d'évaluation

Nous allons considérer une population de n fourmis fourrageuses $a_1, \ldots, a_n$ de l'espèce *Pachycondyla apicalis*. Ces agents sont positionnés dans l'espace de recherche, noté $\mathcal{S}$, et vont tenter de minimiser[1] une fonction d'évaluation f définie de $\mathcal{S}$ dans $\mathbb{R}$. Chaque point $s \in \mathcal{S}$ est une solution valide du problème, ce qui signifie que f est définie en tout point de $\mathcal{S}$. Cet espace de recherche peut être un espace continu, binaire ou un espace de permutations. Notre algorithme, nommé API (pour APIcalis), est générique en ce qui concerne l'espace de recherche $\mathcal{S}$, c'est là un de ses principaux atouts. La définition des deux opérateurs suivants est suffisante pour déterminer le déplacement des fourmis :

1. l'opérateur $\mathcal{O}_{\text{rand}}$ qui génère un point de $\mathcal{S}$ de manière uniformément aléatoire ;
2. l'opérateur $\mathcal{O}_{\text{explo}}$ qui génère un point s' dans le voisinage d'un point s.

Concernant le deuxième opérateur, la taille du voisinage de s est paramétrée par une amplitude, notée A telle que $A \in [0, 1]$. Cette amplitude fixe la portée de l'exploration autour de s relativement à la taille de l'espace. $\mathcal{O}_{\text{explo}}$ peut être une exploration aléatoire tout comme une heuristique inspirée par le domaine de recherche.

[1] D'une façon générale, et comme dans le chapitre précédent, nous ne parlons que de minimisation, considérant le problème de maximisation comme tout à fait réciproque.

7.3.2 Comportement local des fourmis

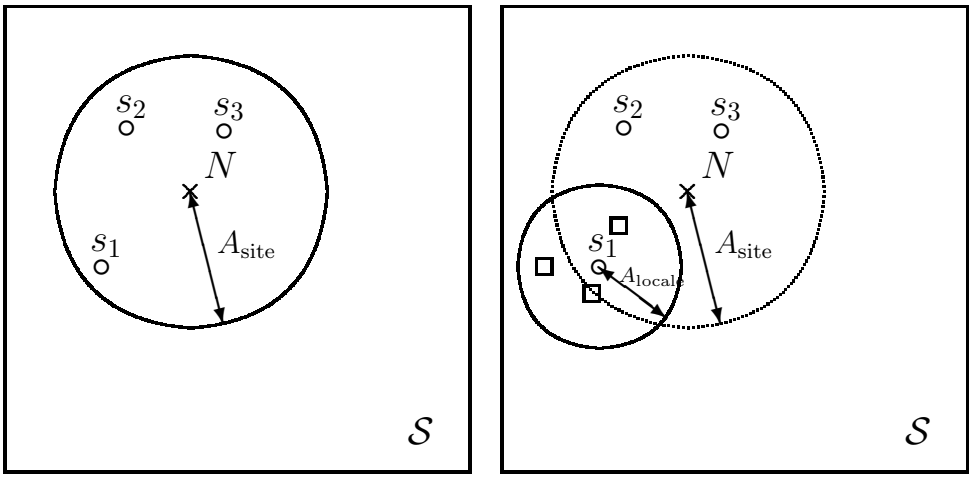


FIG. 7.2 – (a) : Recherche de sites de chasse. Ici, la fourmi se constitue une liste de 3 sites de chasse s_1, s_2 et s_3 dans le voisinage du nid N à une distance maximale A_{site} de celui-ci.
(b) : Exploration locale de la fourmi autour du site s_1. Les carrés représentent les explorations menées dans la zone de rayon A_{locale} autour de s_1.

Au départ, et à chaque déplacement de nid, chaque fourmi a_i quitte le nid pour se constituer une liste de p sites de chasse qu'elle mémorise (figure 7.2.a). Un site de chasse est un point de $\mathcal{S}$ construit par l'opérateur $\mathcal{O}_{\text{explo}}$ avec une amplitude $A_{\text{site}}(a_i)$ dans le voisinage de N. La fourmi a_i va ensuite procéder à une exploration locale autour d'un de ses sites de chasse (figure 7.2.b). 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' de $\mathcal{S}$ dans le voisinage de s grâce à l'opérateur $\mathcal{O}_{\text{explo}}$ avec une amplitude $A_{\text{locale}}(a_i)$. 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 $f(s') < f(s)$. Une amélioration de f modélise donc la capture d'une proie. À 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.

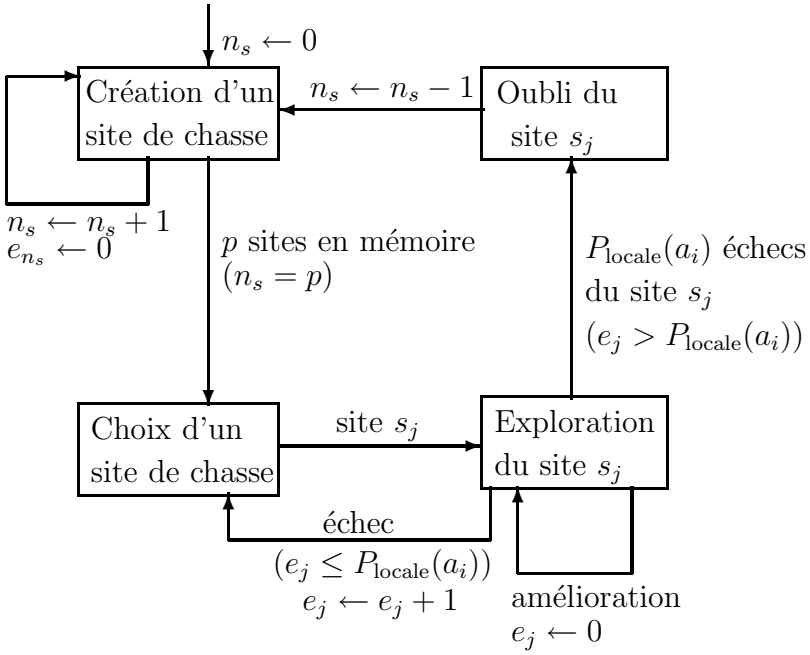


FIG. 7.3 – Automate représentant le comportement individuel de fourragement d'une fourmi. n_s représente le nombre de sites de chasse dans la mémoire de la fourmi et e_j le nombre d'échecs successifs rencontrés sur le site s_j.

L'automate de la figure 7.3 résume le comportement individuel d'une fourrageuse. n_s représente le nombre de sites que la fourmi mémorise à un instant donné. e_j donne le nombre d'échecs successifs du site s_j mémorisé.

7.3.3 Exploration globale

D'un point de vue global, API place le nid à une position N de $\mathcal{S}$ et procède à l'exploration de $\mathcal{S}$ autour de N. L'exploration est déterminée par le comportement local des fourmis. À chaque pas de l'algorithme les n fourmis sont simulées en parallèle. À l'initialisation, le nid est placé dans $\mathcal{S}$ de manière uniformément aléatoire par l'opérateur $\mathcal{O}_{\text{rand}}$. Puis le nid est déplacé toutes les P_N déplacements des n fourmis. Il est alors placé sur le meilleur point s^+ trouvé depuis son dernier déplacement. À chaque déplacement du nid les fourmis reprennent leur exploration à partir de la nouvelle position du nid.

La figure 7.4 montre la statégie globale d'exploration.

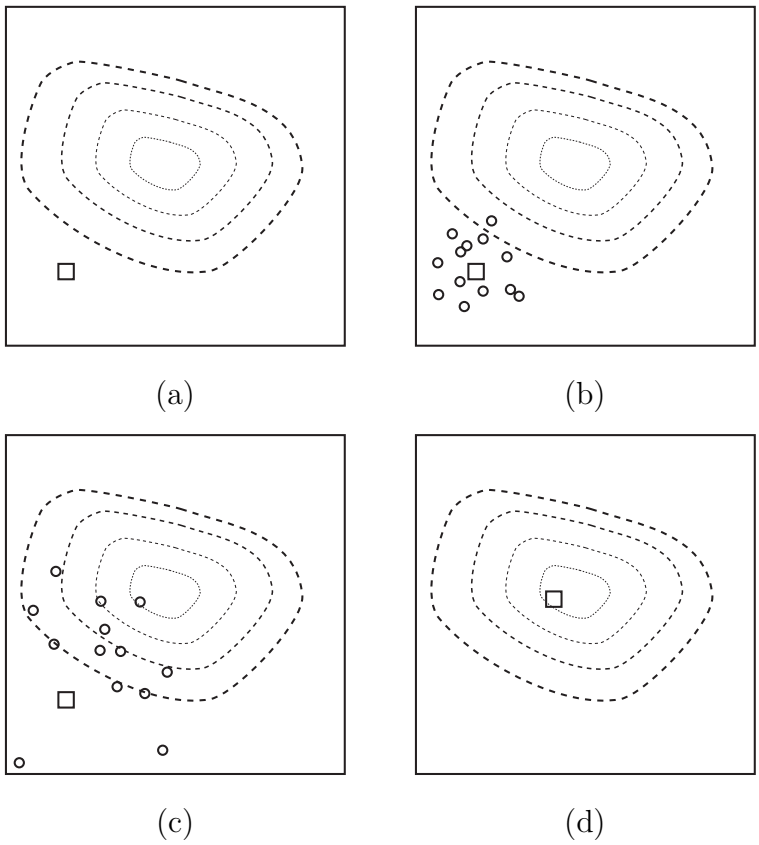


FIG. 7.4 – Exploration globale : déplacement du nid. En (a), le nid (carré) est placé aléatoirement dans l'espace de recherche. En (b) sont représentés les sites de chasse (cercles) créés autour du nid. En (c), à cause de l'exploration locale, les sites de chasse se déplacent vers des zones plus intéressantes de l'espace de recherche (ici les pointillés les plus au centre). En (d) le nid est déplacé sur la position du meilleur site de chasse, les sites seront ensuite générés à partir de cette nouvelle position, comme en (b), et ainsi de suite.

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 de la mémoire d'une fourmi (p) :

$$P_N = 2 \times (P_{\text{locale}} + 1) \times p \tag{7.1}$$

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, disons P_N, ou encore en utilisant certaines informations sur la population comme la dispersion des fourmis dans l'espace de recherche par exemple. Les exemples pouvant être tirés de la littérature sont nombreux concernant cette technique de redémarrage (par exemple le *Delta Coding* (Whitley et al., 1991)).

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, cela permet d'éviter des minima locaux dans lesquels les fourmis resteraient enfermées. Cela permet aussi de rassembler les fourmis autour du meilleur point trouvé et ainsi de concentrer les 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 $s^+ : N \leftarrow (1-\gamma)N + \gamma s^+$ où $\gamma \in [0, 1]$ (à condition que l'espace de recherche $\mathcal{S}$ le permette).

7.3.4 Algorithmes

Les principales étapes de la simulation de la colonie de fourmis sont données par l'algorithme 7.1.

Algorithme 7.1: Simulation d'une colonie de fourmis *Pachycondyla apicalis*
API()

(1) Choisir aléatoirement l'emplacement initial du nid : $N \leftarrow \mathcal{O}_{\text{rand}}$
(2) $T \leftarrow 0$ `/* indice du nombre d'itérations */`
(3) **tantque** La condition d'arrêt n'est vérifiée **faire**
(4) **pour tout** $a_i \in A$ **faire**
(5) API-FOURRAGEMENT(a_i)
(6) **finpour**
(7) **si** le nid doit être déplacé **alors**
(8) $N \leftarrow s^+$ `/* meilleure solution atteinte */`
(9) Vider la mémoire de toutes les fourmis
(10) **finsi**
(11) $T \leftarrow T + 1$
(12) **fintantque**
(13) **retourner** s^+ et $f(s^+)$

La condition d'arrêt peut être l'une des suivantes :
- s^+ n'a pas été améliorée depuis un certain nombre d'itérations (T_1) ;
- T a atteint une valeur limite (T_2) ;
- un certain nombre d'évaluations de solutions de f a été atteint (T_3).

Dans la majorité des tests présentés par la suite, nous appliquons la dernière condition. D'une part cela suppose que l'évaluation des solutions est une opération coûteuse en temps de calcul, et d'autre part cela permet de comparer plus facilement API avec d'autres méthodes.

Le comportement local de fourragement d'une fourmi est donné par l'algorithme 7.2.

Algorithme 7.2: Comportement local d'une fourmi a_i de l'espèce *Pachycondyla apicalis*.

API-FOURRAGEMENT(a_i)

(1) **si** $n_s(a_i) < p$ **alors**
(2) `/* La mémoire de la fourmi n'est pas pleine */`
(3) $n_s(a_i) \leftarrow n_s(a_i) + 1$
(4) Construction d'un site de chasse autour du nid : $s_{n_s(a_i)} \leftarrow \mathcal{O}_{\text{explo}}(N, A_{\text{site}})$
(5) Initialisation du compteur d'echecs du site construit : $e_{n_s(a_i)} \leftarrow 0$
(6) **sinon**
(7) Soit s_j le site que la fourmi a exploré à sa dernière sortie
(8) **si** $e_j > 0$ **alors**
(9) `/* la dernière exploration de` s_j `a été infructueuse */`
(10) Choisir aléatoirement un site s_j de chasse ($j \in \{1, \ldots, p\}$)
(11) **finsi**
(12) Exploration locale autour du site s_j : $s' \leftarrow \mathcal{O}_{\text{explo}}(s_j, A_{\text{locale}})$
(13) **si** $f(s') < f(s_j)$ **alors**
(14) $s_j \leftarrow s'$
(15) $e_j \leftarrow 0$
(16) **sinon**
(17) $e_j \leftarrow e_j + 1$
(18) **si** $e_j > P_{\text{locale}}$ **alors**
(19) Effacer le site s_j de la mémoire de la foumi
(20) $n_s(a_i) \leftarrow n_s(a_i) - 1$
(21) **finsi**
(22) **finsi**
(23) **finsi**

7.4 Extensions de l'algorithme API

Cette section présente quelques une des inspirations biologiques supplémentaires qui ont été exploitées pour enrichir l'algorithme API.

7.4.1 Recrutement

Nous avons vu que *Pachycondyla apicalis* ne faisait pas de recrutement pendant le fourragement, nous avons cependant introduit le recrutement en *tandem running* (marche en tandem) dans API. 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. Par la suite nous noterons l'algorithme API$_r$ lorsque le recrutement est utilisé.

Variantes du mode de recrutement

Le recrutement peut être mis en œuvre de plusieurs façons. Du point de vue de l'optimisation, une fourmi recrute une fourmi moins performante. Nous notons par R_{xy} les différents types de recrutements, x détermine la règle de décision utilisée par une fourmi pour tenter un recrutement et y détermine le choix de la fourmi à recruter. Par exemple, une fourmi peut recruter dans les cas suivants :

1. si elle améliore son optimum ($R_{1\cdot}$) ;
2. si elle améliore un site ($R_{2\cdot}$).

Le premier cas suppose que la fourmi possède une mémoire supplémentaire qui lui permette de conserver la position et la valeur du meilleur point trouvé depuis le début de la recherche.

La fourmi recrutée peut être choisie de différentes manières :

1. au hasard ($R_{\cdot 1}$) ;
2. la fourmi ayant le plus mauvais optimum ($R_{\cdot 2}$) ;
3. la fourmi ayant le plus mauvais site de chasse ($R_{\cdot 3}$).

L'avantage de $R_{\cdot 1}$ est qu'il n'est pas nécessaire de consulter la mémoire de toutes les fourmis pour choisir la candidate au recrutement. Cela représente donc un coût calculatoire moindre.

Recrutement progressif

On peut remarquer qu'il n'est peut être pas très pertinent de recruter dès les premières itérations de la recherche. En effet, il est prématuré de déplacer des fourmis car on dispose de peu d'informations quant aux performances individuelles des fourmis. Nous avons donc introduit un recrutement progressif. Le recrutement devient de plus en plus important avec l'âge du nid. Quand le nid vient d'être déplacé, il y a peu de raisons de pratiquer du recrutement. Ainsi, la fourmi recrute une de ses congénères suivant la probabilité P_r donnée par :

$$P_r = \frac{T}{T_N} \times P_{\text{recrutmax}} \tag{7.2}$$

où T est le nombre d'itérations depuis le dernier déplacement du nid, T_N le nombre d'itérations entre chaque déplacement du nid, et $P_{\text{recrutmax}}$ la probabilité de recruter au moment du déplacement du nid, qui est maximale. Le recrutement s'intensifie donc quand T « s'approche » de T_N. Par la suite nous notons par RP_{xy} le recrutement progressif.

7.4.2 Population hétérogène

La difficulté de choisir les paramètres de l'algorithme et des fourmis en particulier nous a amenés à envisager plusieurs propositions :

1. **supervision des paramètres des fourmis par un algorithme évolutionnaire**. Les paramètres des fourmis sont sélectionnés en fonction de leur adaptation à résoudre le problème traité. Il faut alors considérer plusieurs générations de fourmis où chaque génération prend en compte l'adaptation des générations précédentes. Cette approche a été utilisée pour ACO (Botee and Bonabeau, 1999) ;

2. **Variation au cours du temps des paramètres**. Cette approche est par exemple utilisée pour les fourmis artificielles dans (Bilchev and Parmee, 1995; Bilchev and Parmee, 1996a) où le rayon d'exploration locale décroît avec le temps. On peut noter que dans la nature, les fourmis n'ont pas la même stratégie de chasse en fonction de leur âge : plus elles sont âgées, plus elles sortent fréquemment du nid et plus elle vont chasser loin de celui-ci (Fresneau, 1994). Une autre façon de voir les choses serait d'utiliser l'influence de l'environnement, par exemple à travers les variations de température. En effet, l'activité des ouvrières est liée à la température extérieure. Par exemple, les *Pachycondyla apicalis* chassent le plus aux heures chaudes de la journée. Dans les deux cas (variation de l'âge ou de la température), du point de vue de la modélisation, cela se traduit par une variation des paramètres de chaque fourmi au cours du temps ;

3. **Hétérogénéité des paramètres**. Dans la nature, les fourmis ont des caractéristiques différentes les unes des autres, elles sont parfois même regroupées en castes du fait de leur différences morphologiques. Des modèles de division du travail ont été élaborés pour rendre compte de la capacité dynamique des fourmis à effectuer certaines tâches (Bonabeau et al., 1999). Cette plasticité nous suggère de constituer une population de fourmis ayant des caractéristiques variées.

Nous retenons la dernière proposition. Nous utiliserons cependant un paramétrage statique car la variation d'un paramètre au cours du temps est délicate surtout si elle est adaptative. La supervision par un algorithme évolutionnaire a déjà été expérimentée pour fixer les amplitudes A_{site} et A_{locale} de chaque fourmi et les résultats se sont montrés encourageants (Venturini, 1997).

La population de fourmis sera donc constituée d'individus ayant des paramètres différents. Par la suite, les populations hétérogènes utilisées dans API, que nous noterons API_h, se composeront de fourmis dont seules les amplitudes A_{site} et A_{locale} varieront d'une fourmi à l'autre et seront fixes dans le temps. Nous fixons ces valeurs de façon automatique afin de couvrir le plus de combinaisons possibles :

$$A_{\text{site}}(a_1) = 0.01\varepsilon^0, \ldots, A_{\text{site}}(a_i) = 0.01\varepsilon^{i-1}, \ldots, A_{\text{site}}(a_n) = 0.01\varepsilon^{n-1} = 1 \qquad (7.3)$$

où $\varepsilon = \left(\frac{1}{0.01}\right)^{\frac{1}{n-1}}$. Le paramètre $A_{\text{locale}}(a_i)$ est fixé pour toutes les fourmis à la valeur $0.1A_{\text{site}}(a_i)$. La figure 7.5 donne en exemple les valeurs des paramètres A_{site} et A_{locale} pour 10 fourmis.

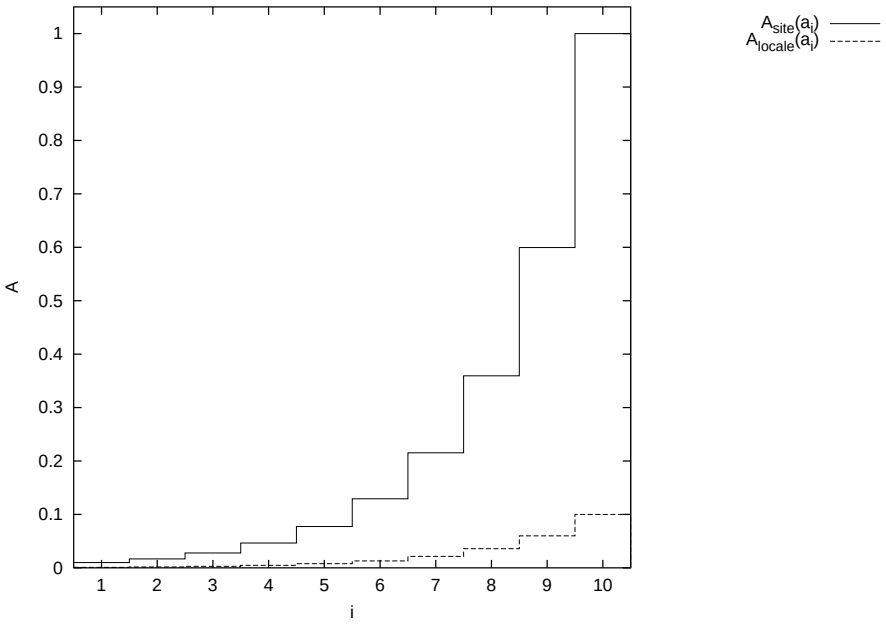


FIG. 7.5 – Valeurs des paramètres $A_{\text{site}}(a_i)$ et $A_{\text{locale}}(a_i)$ pour 10 fourmis.

7.4.3 Prise en compte de la décision de sortir du nid

Un certain nombre de choix ont été arbitrairement faits pour la modélisation des *Pachycondyla apicalis*. Nous n'avons en effet pas pris en compte certaines caractéristiques de cette espèce qui peuvent cependant se révéler intéressantes du point de vue de notre problème d'optimisation. Le comportement prédateur d'une fourmi évolue par exemple en fonction de son âge. Les jeunes fourrageuses (naïves) ont tendance à rester près du nid sans présenter de spécialisation spatiale alors que les fourrageuses expérimentées s'éloignent beaucoup plus du nid tout en montrant une fidélité à une zone de chasse importante. En ce qui concerne la recherche d'un optimum, cela consiste à augmenter l'effort de recherche dans le voisinage du point central. Bien que nous n'avons pas utilisé explicitement de fourmis naïves, la diversité statique introduite dans les paramètres, dans le cas d'une population hétérogène, reproduit en quelque sorte cette diversification basée sur l'âge. Un deuxième aspect n'a pas été pris en compte : l'intervalle de temps qui sépare deux sorties est plus court si la fourmi a rencontré un succés à sa première sortie. S'il est probable que ce comportement permette une certaine économie d'énergie au niveau de la colonie en n'allouant des essais qu'aux fourmis les plus chanceuses/efficaces, il n'est pas évident que dans la transposition au problème de l'optimisation cela améliore les performances de l'ensemble. Comme nous considérons que l'évaluation de la fonction objectif représente une contrainte de ressource nous proposons l'amélioration suivante : la décision de sortir pour une fourmi a_i suit la probabilité $P_s(a_i)$ qui, à chaque itération, est modifiée de la façon suivante :

$$P_s(a_i) \leftarrow (1-\alpha)P_s(a_i) + \alpha\delta \tag{7.4}$$

où α est un paramètre réel compris dans l'intervalle $[0, 1]$ et commun à toutes les fourmis. δ est une valeur binaire qui vaut 1 si la dernière sortie a été fructueuse et 0 sinon. Au départ, et à chaque déplacement de nid, nous fixons P_s à 1 et δ à 0.

L'algorithme API que nous proposons peut être rapproché d'une modélisation des *Pachycondyla apicalis* proposée en 1987 (Deneubourg et al., 1987). L'objectif de cette modélisation, que nous noterons API87 par la suite, était de montrer que l'amplification de la probabilité de sortie ($P_s(a_i)$) à chaque retour fructueux au nid se rapportait à une capacité d'apprentissage. Dans API87, la probabilité de sortir du nid était modifiée de la façon suivante :

$$P_s(a_i) \leftarrow P_s(a_i) + \delta \min\{p^+, 1 - P_s(a_i)\} + (\delta - 1) \min\{p^-, P_s(a_i) - P_s\} \tag{7.5}$$

où p^+ et p^- sont des constantes représentant des taux d'apprentissage et P_s est la borne inférieure de la probabilité de quitter le nid. Ceci signifie qu'en cas de succès, la probabilité de sortir du nid à la prochaine itération est augmentée (tout en étant bornée par 1). En cas d'échec, par contre, la probabilité de sortir du nid est diminuée (tout en étant bornée par P_s).

Dans API, lorsqu'une fourmi a connu un succès, elle retourne systématiquement sur le site fructueux. API87 ne suit pas exactement cette règle et introduit une probabilité de choisir un site. Cette probabilité, $P_j(a_i)$, de choisir le site s_j est augmentée quand une proie vient d'être trouvée :

$$P_j(a_i) \leftarrow P_j(a_i) + \min\{q^+, 1 - P_j(a_i)\} \tag{7.6}$$

où q^+ représente le taux d'apprentissage spatial positif. Par contre, si aucune proie n'est trouvée lors d'une sortie, la probabilité de choisir ce site par rapport aux autres est diminuée. Ce qui s'exprime, dans le cas de deux sites :

$$\forall j \in \{1, 2\} : P_j(a_i) \leftarrow \begin{cases} P_j(a_i) - \min\{q^-, P_j(a_i) - 0.5\} & \text{si } P_j(a_i) > 0.5 \\ P_j(a_i) + \min\{q^-, 0.5 - P_j(a_i)\} & \text{sinon} \end{cases} \tag{7.7}$$

où q^- représente le taux d'apprentissage spatial positif. Dans ce cas les probabilités de choisir un site sont rapprochées de l'équiprobabilité.

7.5 Etude expérimentale

Dans cette section nous présentons une étude epérimentale sur l'influence des paramètres d'API sur les résultats obtenus. Puis nous évaluerons les extensions que nous avons proposées. Comme il faut appuyer cette étude sur un problème particulier, nous nous intéressons au problème d'optimisation numérique défini au chapitre 5. Les fonctions étudiées sont celles données dans le tableau 6.1 (page 107). La section 7.7 donne un certain nombre d'applications de API à d'autres types de problèmes d'optimisation.

7.5.1 Les paramètres d'API

Commençons par rappeler les différents paramètres à fixer dans API. Le tableau 7.1 en donne un résumé. Les paramètres A_{site} et A_{locale} ne sont pas utilisés dans la version

Paramètre	Description
n	nombre de fourmis
$\mathcal{O}_{\text{rand}}$	opérateur générant aléatoirement un point de $\mathcal{S}$
$\mathcal{O}_{\text{explo}}(s)$	opérateur générant un point de $\mathcal{S}$ dans le voisinage de s
A_{site}	amplitude de création d'un site de chasse
A_{locale}	amplitude d'exploration locale
P_{locale}	patience locale d'une fourmi
p	nombre de site mémorisés par une fourmi
P_N	patience du nid
T_3	nombre d'évaluations de f
R_{xy}, RP_{xy}	types de recrutement
P_{recrut}, $P_{\text{recrutmax}}$	probabilités de recrutement

TAB. 7.1 – Paramètres de API.

hétérogène (API_h). Les paramètres R_{xy}, RP_{xy}, P_{recrut} et $P_{\text{recrutmax}}$ ne sont utilisés que si on autorise le recrutement (API_r). Enfin le paramètre P_N est déterminé automatiquement par la formule 7.1.

7.5.2 Opérateurs spécifiques à l'optimisation numérique

Nous avons défini d'une manière générale les opérateurs $\mathcal{O}_{\text{rand}}$ et $\mathcal{O}_{\text{explo}}$ dans la section 7.3 pour pouvoir les adapter à différents problèmes. Il convient maintenant de les préciser un peu plus.

Les fonctions étudiées sont définies sur un produit d'intervalles de $\mathbb{R}$, chacun délimité par deux bornes b_i et B_i. L'opérateur $\mathcal{O}_{\text{rand}}$ doit donc générer de façon aléatoire un point de l'espace $\mathcal{S} = [b_1, B_1] \times \ldots \times [b_l, B_l]$ où l représente la dimension de la fonction à minimiser. Nous choisissons de générer un point $s = (s_i)_{i=1\ldots l} \in \mathcal{S}$ de façon uniforme :

$$s_i = b_i + \mathcal{U}[0,1](B_i - b_i) \tag{7.8}$$

où $\mathcal{U}[0,1]$ est un nombre aléatoire uniformément tiré dans $[0,1]$.

L'opérateur d'exploration $\mathcal{O}_{\text{explo}}$ tient compte d'une amplitude A :

$$s'_i = s_i + A\mathcal{U}[-0.5, 0.5](B_i - b_i) \quad \forall i \in \{1, \ldots, l\} \tag{7.9}$$

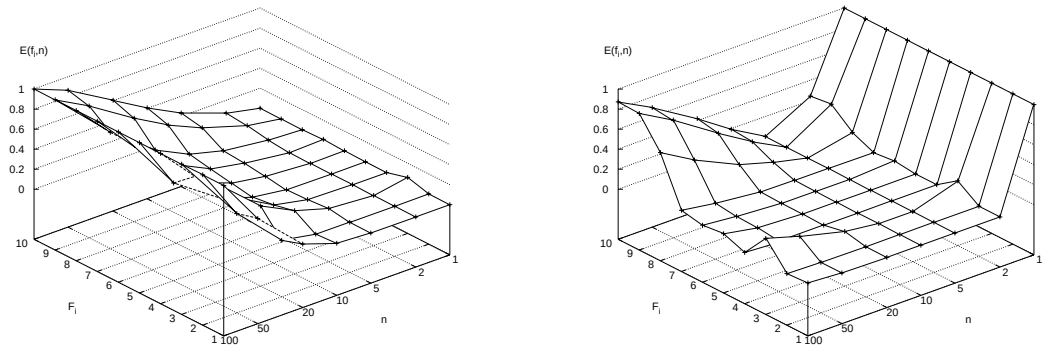
où $\mathcal{U}[-0.5, 0.5]$ est un nombre aléatoire uniformément tiré dans $[-0.5, 0.5]$. Evidemment, il faut s'assurer que $s'_i \in [b_i, B_i]$. Cet opérateur a lui aussi un comportement uniforme dans l'intervalle $[s_i - 0.5 \times A(B_i - b_i), s_i + 0.5 \times A(B_i - b_i)]$. Nous verrons par la suite qu'une distribution normale est une alternative intéressante à cette uniformité (section 7.7.1).

Sur ces bases, les paragraphes suivants explorent les performances des paramétrages possibles de API. Les valeurs des paramètres testées sont données dans le tableau 7.2.

Paramètre	valeurs
n	$\{1, 2, 5, 10, 20, 50, 100\}$
A_{site}	$\{0.01, 0.1, 0.5\}$
A_{locale}	$\{0.01, 0.05, 0.1\}$
P_{locale}	$\{10, 20, 30, 40\}$
p	$\{1, 2, 3, 5\}$
T_N	automatique (voir équation 7.1)
T_3	$\{10000\}$
R_{xy}, RP_{xy}	$(x, y) \in \{1, 2\} \times \{1, 2, 3\}$

TAB. 7.2 – Valeurs des paramètres testées. La version homogène de la population est la seule concernée pour A_{site} et A_{locale}.

7.5.3 Influence du nombre de fourmis



(a) : API (b) : API_h

FIG. 7.6 – Influence du nombre de fourmis sur API et API_h. En abscisse figure le numéro de chaque fonction, en ordonnée figure la taille de la population(n). La mesure $E(F_i, n)$ correspond à la moyenne des meilleures évaluations trouvées pour tous les jeux de paramètres.

Les tests concernant la taille de la population ont été effectués sur une population homogène (API) et sur une population hétérogène (API_h). Le recrutement n'a pas été utilisé ce qui signifie que selon le tableau 7.2, API a été testé sur $7 \times 3 \times 3 \times 4 \times 4 = 1\,008$ jeux de paramètres par fonction et API_h sur $7 \times 4 \times 4 = 112$. Pour les 10 fonctions, chaque jeux de paramètre a été testé 30 fois[2]. Sauf mention contraire, la moyenne sur 30 essais est toujours utilisée par la suite.

Les courbes présentées sur la figure 7.6 donnent la moyenne des meilleures évaluations moyennes obtenues pour chaque fonction et pour un nombre de fourmis fixé par rapport aux autres paramètres. Chaque valeur est replacée dans l'intervalle $[0, 1]$ pour

[2]Ce qui fait un total de $(1\,008 + 112) \times 10 \times 30 = 336\,000$ exécutions de API.

faire disparaitre les différences entre les fonctions. Par exemple, si on utilise la notation suivante pour représenter le résultat renvoyé par API et des paramètres associés : $\mathrm{API}(F_i, n, A_{\text{site}}, A_{\text{locale}}, P_{\text{locale}}, p)$, ($\mathrm{API}(F_i, n, A_{\text{site}}, A_{\text{locale}}, P_{\text{locale}}, \cdot)$ représente alors la moyenne obtenue en faisant varier le paramètre p), $E(F_i, 10)$ est calculée de la façon suivante :

$$E(F_i, 10) = \frac{\mathrm{API}(F_i, 10, \cdot, \cdot, \cdot, \cdot) - \min_n \mathrm{API}(F_i, n, \cdot, \cdot, \cdot, \cdot)}{\max_n \mathrm{API}(F_i, n, \cdot, \cdot, \cdot, \cdot) - \min_n \mathrm{API}(F_i, n, \cdot, \cdot, \cdot, \cdot)} \tag{7.10}$$

Ce qui signifie que $E(F_i, n)$ vaut 0 pour le nombre de fourmis donnant le meilleur résultat et 1 pour le plus mauvais.

Les différences entre API et API_h sont importantes :

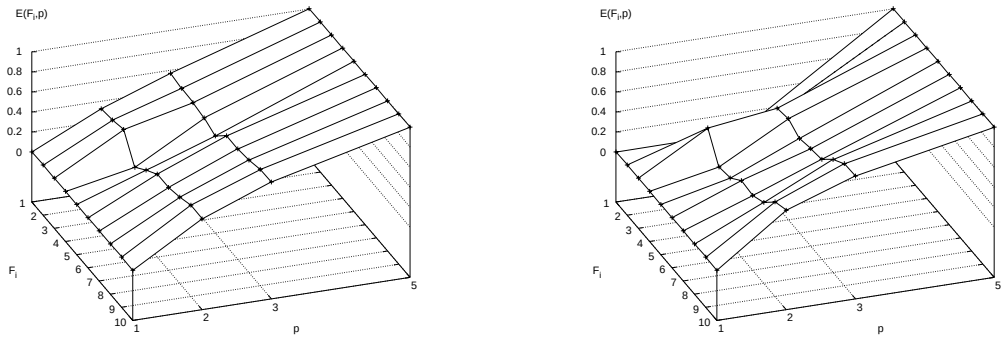
- la version homogène (API) voit ses performances diminuer fortement quand le nombre de fourmis augmente alors que la version hétérogène (API_h) est plus performante quand $n > 2$;
- pour toutes les fonctions API obtient ses plus mauvais résultats avec $n = 100$ et ses meilleurs avec $n = 1$ alors que API_h obtient ses plus mauvais résultats avec $n = 1$ et ses meilleurs avec $n = 5$. Il faut cependant noter que lorsque $n = 1$, API_h utilise les paramètres suivants : $A_{\text{locale}} = 0.001$ et $A_{\text{site}} = 0.01$ (formule 7.3).

Les conclusions sur le nombre de fourmis à utiliser que nous pouvons déduire doivent être prudentes étant donné que les autres paramètres ne sont pas fixés. En effet, l'inconvénient d'utiliser la moyenne des résultats sur tous les jeux de paramètres peut cacher un certain nombre d'associations entre les paramètres qui contredirait la tendance constatée. On peut cependant conclure que :

- API_h est moins sensible que API au nombre de fourmis, ce qui est un avantage étant donné que les fonctions étudiées demeurent relativement simples. Dans le cas de problèmes plus difficiles il peut en effet être intéressant d'augmenter le nombre de fourmis pour élargir la recherche et dans ce cas la version homogène se montre plus rapidement instable. En effet, en utilisant une population homogène, on risque plus d'avoir un jeu de paramètres mal adapté au problème, ce qui signifie qu'en augmentant le nombre de fourmis, la capacité de recherche n'est pas plus étendue et le nombre d'explorations augmente au détriment du nombre d'exploitations puisque nous limitons le nombre d'évaluations de la fonction objectif ;
- dans le cas hétérogène, une population de 5 fourmis est le plus adapté quand on considère la moyenne des résultats obtenus pour tous les jeux de paramètres. Si le nombre de fourmis est trop restreint, les variations des paramètres dans la population sont trop rapides et ne permettent pas de couvrir suffisamment de cas. Par contre, similairement à la version homogène, s'il y a beaucoup de fourmis, le nombre d'explorations est trop important par rapport au nombre d'exploitations.

7.5.4 Influence du nombre de sites de chasse

Les courbes de la figure 7.7 sont obtenues en fixant le nombre de sites de chasse et en faisant varier les autres paramètres. Dans les deux cas (API et API_h) il semble évident qu'il est inutile d'utiliser trop de sites de chasse. Cela peut s'expliquer par le

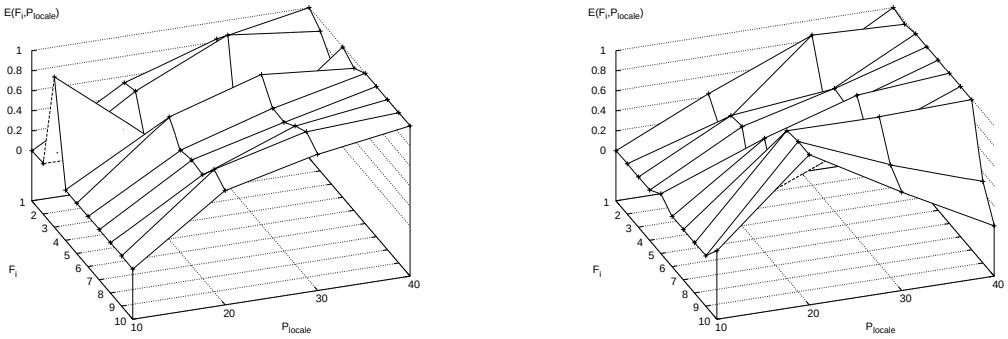


(a) : API (b) : API_h

FIG. 7.7 – Influence du nombre de sites de chasse p sur API et API_h.

fait que les fourmis passent trop de temps en exploration (création de sites de chasse) relativement à l'effort d'exploitation (recherche de proie sur un site) quand le nombre de sites de chasse par fourmi est trop grand.

7.5.5 Influence de la patience locale



(a) : API (b) : API_h

FIG. 7.8 – Influence de la patience locale P_{locale} sur API et API_h.

Les courbes de la figure 7.8 sont obtenues en fixant la patience locale et en faisant varier les autres paramètres. Dans les deux cas (API et API_h) il semble qu'une patience locale trop élevée ne soit pas bénéfique (sauf pour API sur la fonction F_3 et API_h sur les fonctions F_9 et F_{10}). On peut aussi noter qu'une patience de 10 donne de mauvais résultats pour la fonction F_3 si on utilise une population homogène.

7.5.6 Population homogène contre population hétérogène

Le tableau 7.3 donne les meilleurs résultats obtenus par API (population homogène) et API_h (population hétérogène) sur les 10 fonctions ainsi que le jeu de paramètre correspondant.

F_i	API		API_h	
	min.	$(A_{\text{locale}}, A_{\text{site}}, P_{\text{locale}}, p, n)$	min.	$(P_{\text{locale}}, p, n)$
F_1	6.65×10^{-6}	(0.01, 0.01, 20, 1, 2)	▶ 1.79×10^{-7}	(40, 1, 1)
F_2	1.01×10^{-6}	(0.01, 0.01, 30, 1, 2)	▶ 1.75×10^{-7}	(40, 1, 2)
F_3	▶ $2.32 \times 10^{+0}$	(0.01, 0.5, 40, 3, 1)	$2.54 \times 10^{+0}$	(20, 1, 10)
F_4	2.22×10^{-3}	(0.01, 0.01, 20, 1, 1)	▶ 1.24×10^{-4}	(30, 1, 2)
F_5	1.74×10^{-1}	(0.01, 0.01, 40, 2, 1)	▶ 3.38×10^{-2}	(40, 2, 2)
F_6	2.23×10^{-3}	(0.05, 0.01, 20, 1, 5)	▶ 3.53×10^{-5}	(30, 3, 5)
F_7	1.98×10^{-1}	(0.1, 0.01, 10, 1, 20)	▶ 8.50×10^{-2}	(40, 1, 2)
F_8	▶ $4.71 \times 10^{+1}$	(0.01, 0.01, 30, 1, 1)	$7.45 \times 10^{+1}$	(40, 1, 5)
F_9	▶ $2.45 \times 10^{+1}$	(0.01, 0.01, 40, 1, 1)	$4.72 \times 10^{+1}$	(40, 1, 5)
F_{10}	▶ $8.04 \times 10^{+0}$	(0.01, 0.01, 40, 1, 1)	$4.32 \times 10^{+1}$	(40, 1, 5)
Moy.		(0.02, 0.06, 29, 1.3, 3.5)		(36, 1.3, 3.9)

TAB. 7.3 – Comparaison de API et API_h avec le jeux de paramètres correspondant. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction. La dernière ligne donne pour chaque méthode les paramètres moyens.

Un certain nombre de remarques précédemment faites trouvent ici une confirmation :

- le nombre de sites de chasse (p) doit être réduit (1.3 en moyenne pour API et API_h) ;
- il ne sert à rien d'utiliser trop de fourmis (3.5 en moyenne pour API et 3.9 pour API_h).

Par contre, les conclusions que nous avions tirées sur la patience locale (P_{locale}) sont invalidées : elle est plus élevée que ce qui avait été préconisé dans la section précédente puisqu'en moyenne il faut 29 explorations successivement infructueuses pour API et 36 pour API_h pour que chaque méthode obtienne ses meilleurs résultats.

Le tableau 7.3 ne nous permet pas de dire si l'hétérogénéïté est plus valable que l'homogénéïté de la population pour la simple raison que API a bénéficié d'un nombre de jeux de paramètres beaucoup plus important que API_h.

Nous avons donc lancé API et API_h avec pour chacun le jeu de paramètres moyen calculé dans le tableau 7.3. Le tableau 7.4 donne les résultats obtenus. On constate que par rapport au tableau 7.3 c'est API qui a le plus souffert de ce nouveau paramétrage puisque ses résultats ont en moyenne subi une déviation de 3.74 alors que cette déviation moyenne vaut 1.65 pour API_h. Cela nous permet de conclure, et ce n'est pas vraiment une surprise, que API_h est plus robuste que API relativement au jeu de paramètres utilisé.

F_i	API			API_h		
	min.	σ	F	min.	σ	F
F_1	3.45×10^{-5}	2.74×10^{-5}	3.24	▶ 3.72×10^{-7}	2.65×10^{-7}	2.44
F_2	1.09×10^{-5}	1.05×10^{-5}	3.91	▶ 3.50×10^{-7}	2.46×10^{-7}	1.89
F_3	$3.28 \times 10^{+1}$	$1.66 \times 10^{+1}$	1.95	▶ $2.17 \times 10^{+0}$	$1.53 \times 10^{+0}$	2.55
F_4	1.15×10^{-2}	5.04×10^{-3}	2.84	▶ 4.63×10^{-4}	8.50×10^{-4}	13.79
F_5	2.47×10^{-1}	6.97×10^{-2}	2.79	▶ 9.07×10^{-2}	6.58×10^{-2}	4.45
F_6	8.73×10^{-3}	2.55×10^{-3}	6.48	▶ 3.86×10^{-4}	1.75×10^{-3}	26.92
F_7	2.37×10^{-1}	7.72×10^{-2}	2.56	▶ 1.13×10^{-1}	3.20×10^{-2}	2.71
F_8	▶ $6.03 \times 10^{+1}$	$4.08 \times 10^{+0}$	4.14	$7.27 \times 10^{+1}$	$4.99 \times 10^{+0}$	3.67
F_9	▶ $3.44 \times 10^{+1}$	$2.94 \times 10^{+0}$	1.86	$4.77 \times 10^{+1}$	$4.67 \times 10^{+0}$	2.92
F_{10}	▶ $2.28 \times 10^{+1}$	$3.89 \times 10^{+0}$	2.66	$3.92 \times 10^{+1}$	$5.76 \times 10^{+0}$	2.48

TAB. 7.4 – Comparaison de API et API_h avec le jeu de paramètres moyen calculé dans le tableau 7.3. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction, σ représente l'écart type et F le coefficient de Fisher définit dans la section 6.7.3, page 108.

7.5.7 Intérêt du recrutement

Avec les notations introduites dans la section 7.4.1, les figures 7.9 et 7.10 présentent les résultats obtenus respectivement pour API_r et API_{rh} pour toutes les combinaisons de recrutement possibles avec $P_{\text{recrut}} = 0.8$ et $P_{\text{recrutmax}} = 0.8$. Les paramètres, quand ils s'appliquent, sont les suivants : $A_{\text{locale}} = 0.01$, $A_{\text{site}} = 0.1$, $P_{\text{locale}} = 20$, $p = 2$, $n = 20$ et $T_3 = 10\,000$. Le recrutement le moins efficace est représenté par la valeur 1 et le plus efficace par la valeur 0.

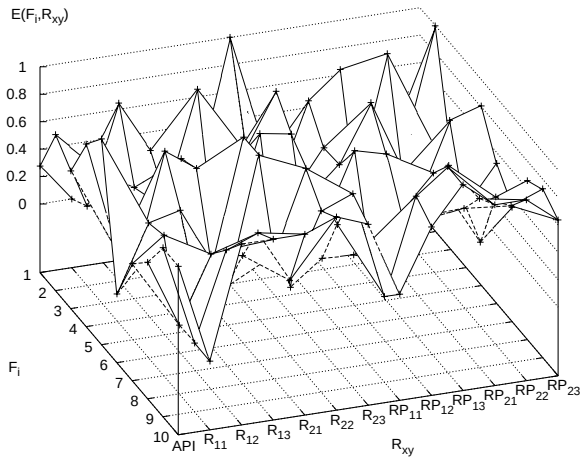


FIG. 7.9 – Comparaison des différentes méthodes de recrutement pour une population homogène. En abscisse figurent les fonctions et en ordonnée les différentes méthodes de recrutement (API correspond à aucun recrutement).

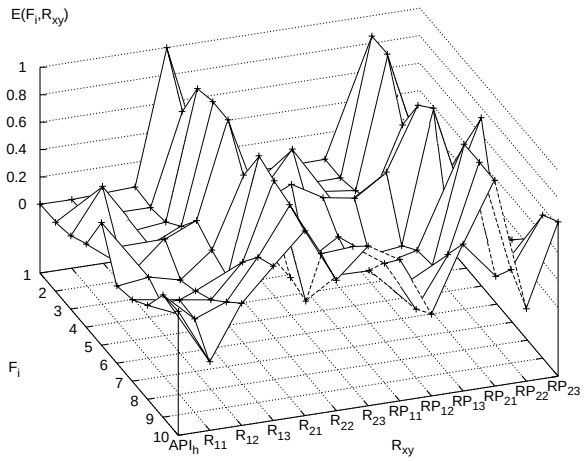


FIG. 7.10 – Comparaison des différentes méthodes de recrutement pour une population hétérogène. En abscisse figurent les fonctions et en ordonnée les différentes méthodes de recrutement (API_h correspond à aucun recrutement).

Concernant la version homogène, aucune tendance franche ne semble se dégager. En ce qui concerne la version hétérogène on constate que deux méthodes de recrutement donnent des résultats médiocres pour la plupart des fonctions. Il s'agit des méthodes R_{21} et RP_{21} qui correspondent au choix, pour une fourmi, de recruter lorsqu'elle améliore un site et qu'elle tente de recruter une autre fourmi au hasard. Les courbes de la

figure 7.11 donnent la moyenne des performances sur l'ensemble des fonctions. On y retrouve les deux contre-performances de R_{21} et RP_{21}.

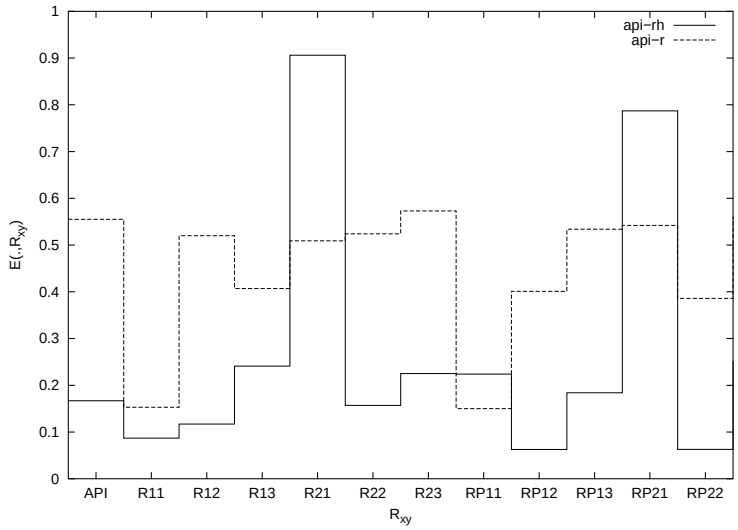


FIG. 7.11 – Comparaison de API_r et API_{rh}. En abscisse figurent les différentes méthodes de recrutement (API correspond à aucun recrutement). Ces valeurs correspondent aux moyennes de chaque type de recrutement sur les dix fonctions étudiées.

7.5.8 Quand faut-il sortir du nid ?

Le tableau 7.5 donne les résultats de la modélisation de la décision de sortir du nid avec α fixé à 0.1. Les autres paramètres ont pour valeurs : $n = 20$, $P_{\text{locale}} = 20$ et $p = 2$. Ces résultats sont une moyenne sur 30 essais et la population est hétérogène dans les deux cas. On note alors cette version API_{sh}. Le critère d'arrêt T_3 est fixé à 100 000 et correspond au nombre de tentatives de sortie du nid et non pas au nombre d'évaluations de la fonction, c'est pour cette raison que nous avons fait figurer le nombre moyen d'évaluations effectuées réellement par API_s (k). Sur les dix fonctions testées, la version API_{sh} a trouvé de meilleurs résultats que API_h dans 3 cas seulement. Ceci semble indiquer que l'incorporation de la décision de sortir du nid n'est pas bénéfique pour le jeux de test étudié. Cependant il faut remarquer qu'API_{sh} bat API_h pour les fonctions définies sur des espaces de grande dimension (100). La figure 7.12 donne les courbes de convergence moyennes de API_{sh} et API_h pour la fonction F_8. Comme API_h effectue 20 évaluations de la fonction F_8 à chaque itération, il ne faut que $\frac{37791}{20} = 1890$ itérations. Par contre, pour API_{sh}, il faut $\frac{100\,000}{20} = 5000$ itérations. La figure 7.13 donne la probabilité $\bar{P}_s$ moyennée sur l'ensemble des fourmis et des essais pour la même fonction.

F_i	API_{sh}	k	API_h
F_1	2.37×10^{-6}	38970	▶ 5.95×10^{-7}
F_2	9.45×10^{-7}	35060	▶ 4.08×10^{-7}
F_3	$2.60 \times 10^{+0}$	34376	▶ 9.32×10^{-1}
F_4	8.16×10^{-4}	25899	▶ 5.99×10^{-4}
F_5	1.35×10^{-1}	29453	▶ 6.07×10^{-2}
F_6	3.56×10^{-5}	25609	▶ 1.94×10^{-5}
F_7	1.37×10^{-1}	20602	▶ 1.18×10^{-1}
F_8	▶ $6.66 \times 10^{+1}$	37791	$7.03 \times 10^{+1}$
F_9	▶ $4.64 \times 10^{+1}$	37158	$4.98 \times 10^{+1}$
F_{10}	▶ $4.17 \times 10^{+1}$	38024	$4.97 \times 10^{+1}$

TAB. 7.5 – Résultats de API_{sh} ($\alpha = 0.1$). k correspond au nombre moyen d'évaluations effectuées par API_{sh}. Ce nombre a servi à initialiser le nombre d'évaluations de API_h pour que les deux méthodes disposent du même nombre d'évaluations de la fonction. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction.

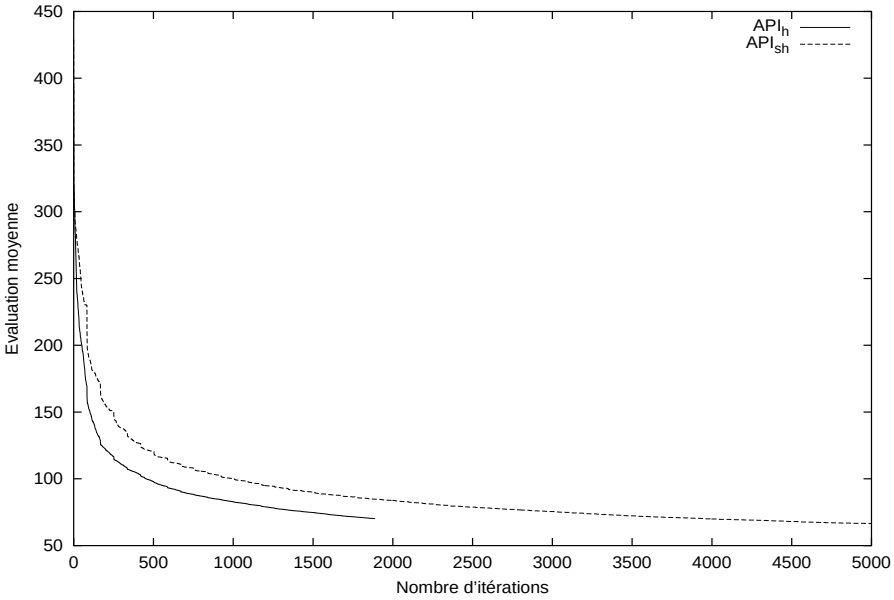


FIG. 7.12 – Convergence de API_{sh} par rapport à API_h pour la fonction F_8. Comme toutes les fourmis ne sortent pas à chaque itérations dans API_{sh}, il faut plus d'itérations à API_{sh} pour effectuer le même nombre d'évaluations de F_8.

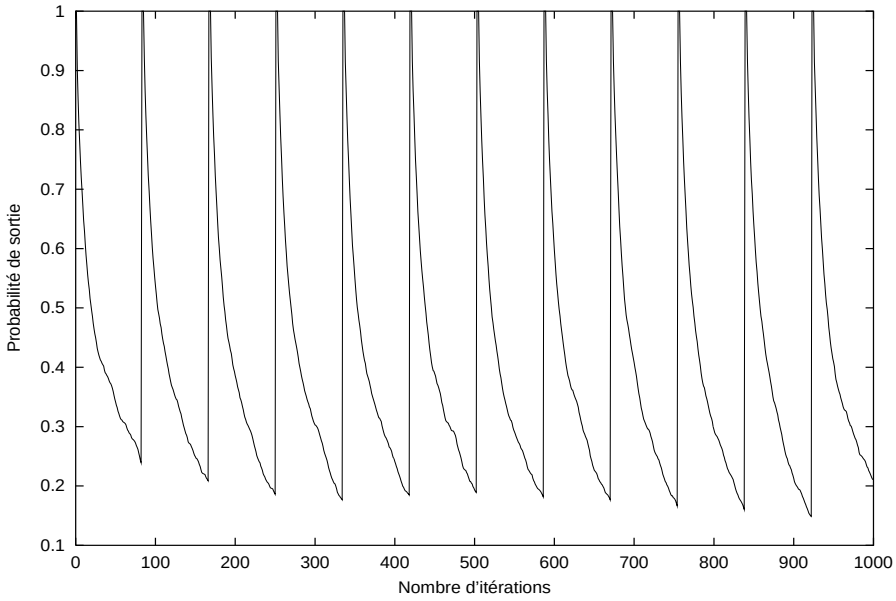


FIG. 7.13 – Evolution de la probabilité moyenne de sortir du nid de API_{sh} pour la fonction F_8 limitée aux 1 000 premières itérations. On remarque la forte influence du déplacement périodique du nid.

F_i	API_s	k	API
F_1	3.80×10^{-7}	81765	▶ 3.20×10^{-7}
F_2	2.70×10^{-7}	79464	▶ 1.72×10^{-7}
F_3	3.67×10^{-1}	79695	▶ 3.52×10^{-1}
F_4	1.37×10^{-4}	75599	▶ 1.36×10^{-4}
F_5	3.58×10^{-2}	77628	▶ 3.03×10^{-2}
F_6	▶ 5.04×10^{-6}	75381	5.45×10^{-6}
F_7	▶ 8.79×10^{-2}	73236	8.85×10^{-2}
F_8	▶ $5.73 \times 10^{+1}$	81295	$5.84 \times 10^{+1}$
F_9	▶ $3.75 \times 10^{+1}$	81518	$3.83 \times 10^{+1}$
F_{10}	▶ $2.91 \times 10^{+1}$	81864	$2.96 \times 10^{+1}$

TAB. 7.6 – Résultats de API_{sh} ($\alpha = 0.01$). k correspond au nombre moyen d'évaluations effectuées par API_{sh}. Ce nombre a servi à initialiser le nombre d'évaluations de API_h pour que les deux méthodes disposent du même nombre d'évaluations de la fonction. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction.

Le tableau 7.6 présente les résultats obtenus en suivant le même protocole de test mais en fixant la valeur de α à 0.01. Comme on pouvait s'y attendre, on constate que le nombre d'évaluations (c'est-à-dire le nombre de sorties du nid) a augmenté par rapport aux tests présentés dans le tableau 7.5. Ces résultats confirment le comportement de API_{sh} par rapport à API_h.

En conclusion, API_{sh} permet d'obtenir des résultats comparables à API_h pour un nombre équivalent d'évaluations de la fonction objectif. L'avantage de cette amélioration n'est donc pas évident en terme de qualité. Il reste que pour un nombre de tentatives de sorties du nid équivalent, API_{sh} peut être plus rapide en évaluant moins de fois la fonction que API_h où chaque tentative de sortie du nid donne lieu à une évaluation. Le tableau 7.7 donne les temps de calcul obtenus en donnant à API_{sh} et à API_h 100 000 tentatives de sortie du nid. Les valeurs $\Delta(t)$:

$$\Delta(t) = 100\frac{t(\text{API}_h) - t(\text{API}_{sh})}{t(\ \text{API}_h)} \tag{7.11}$$

et $\Delta(\text{min})$ (même formulation) donnent une idée de l'apport de API_{sh} par rapport à API_h. On constate que le gain de temps est variable d'une fonction à l'autre : API_{sh}

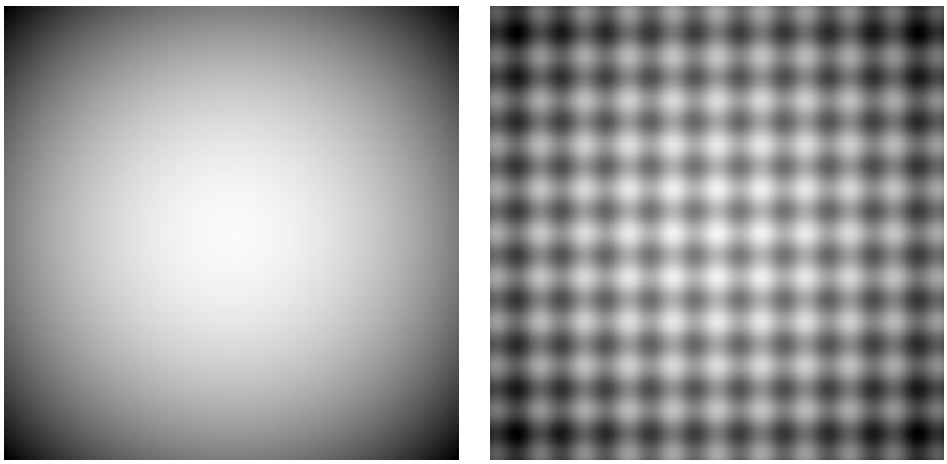
F_i	API_{sh}				API_h			$\Delta(t)$	$\Delta(\text{min})$
	min	F	t		min	F	t	%	%
F_1	3.80×10^{-7}	3.72	0.97	▶	3.03×10^{-7}	2.30	1.02	4.90	-25.41
F_2	2.70×10^{-7}	6.01	0.92	▶	1.20×10^{-7}	6.78	0.96	4.17	-125.00
F_3	3.67×10^{-1}	1.31	1.22	▶	3.33×10^{-1}	3.82	1.34	8.96	-10.21
F_4	1.37×10^{-4}	4.14	1.00	▶	9.19×10^{-5}	2.93	0.88	-13.64	-49.08
F_5	3.58×10^{-2}	4.22	1.30	▶	2.91×10^{-2}	2.59	1.11	-17.12	-23.02
F_6	5.04×10^{-6}	7.49	1.21	▶	3.37×10^{-6}	4.44	1.07	-13.08	-15.33
F_7	8.79×10^{-2}	3.02	1.23	▶	7.76×10^{-2}	2.57	1.30	5.38	-13.27
F_8	$5.73 \times 10^{+1}$	3.34	8.94	▶	$5.43 \times 10^{+1}$	3.12	10.98	18.58	-5.52
F_9	$3.75 \times 10^{+1}$	2.26	11.86	▶	$3.56 \times 10^{+1}$	2.72	14.40	17.64	-5.33
F_{10}	$2.91 \times 10^{+1}$	2.55	9.58	▶	$2.56 \times 10^{+1}$	2.57	11.39	15.89	-13.67

TAB. 7.7 – Résultats de API_{sh} ($\alpha = 0.01$) : évaluation de la durée. Pour les deux méthodes, les résultats sont obtenus avec 100 000 itérations. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction, F au second coefficient de Fisher et t à la durée moyenne en secondes pour un essai. $\Delta(t)$ et $\Delta(\text{min})$ sont décrits dans le texte.

fait gagner jusqu'à 18.58% de temps par rapport à API_h (fonction F_8) alors qu'il peut être plus lent de 17.12% (fonction F_5). Cependant, API_{sh} est globalement plus rapide que API_h pour 100 000 tentatives de sorties du nid (7 fonctions sur 10). Du point de vue des performances, API_{sh} est toujours moins performant que API_h avec là aussi de fortes disparités ($\Delta(\text{min})$). On constate ainsi qu'en utilisant API_{sh}, on peut améliorer le temps de calcul d'un peu moins de 5%, mais on divise alors les performances par plus de deux (fonction F_2). D'un autre coté, en gagnant un peu plus de 18% du temps de calcul pour la fonction F_8, API_{sh} obtient des résultats moins bons que API_h d'environ 5%. La tendance principale que l'on peut retirer de ce jeux de test est que API_{sh} est bien adapté aux fonctions de grande dimension puisque dans ce cas le gain en temps est supérieur à la perte en qualité. API_{sh} est donc tout indiqué pour des problèmes dont l'évaluation d'une solution est coûteuse en temps de calcul.

7.5.9 Cartes d'exploration

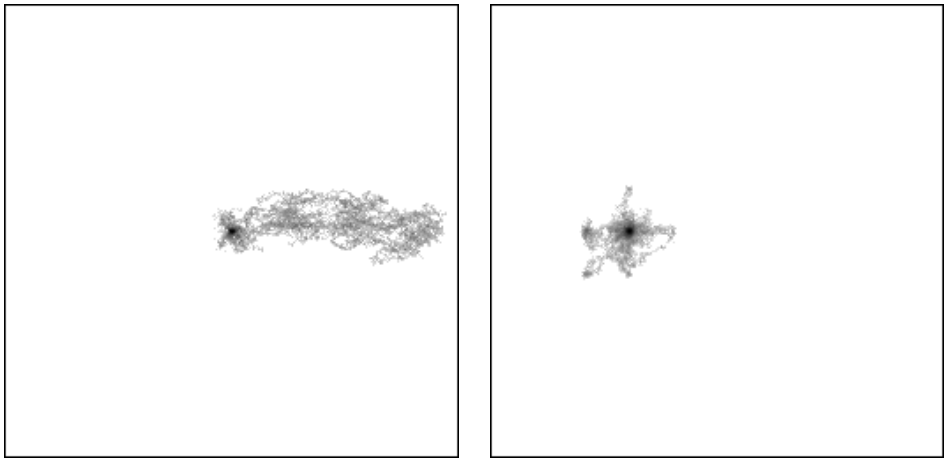
Afin d'illustrer le fonctionnement d'API, nous avons représenté sur une carte les explorations qui étaient effectuées par les fourmis. Cela permet d'avoir une idée sur la répartition de l'effort de recherche. Prenons par exemple la fonction F_3 qui présente la caractéristique de posséder de nombreux minima locaux. La figure 7.14 donne une représentation en deux dimensions des fonctions F_1 et F_3, les niveaux de gris étant utilisés pour représenter la valeur de la fonction en un point. Les points les plus bas (au sens de l'évaluation de la fonction) sont représentés en blanc alors que les plus hauts le sont en noir et le minimum global est au centre de l'image.



F_1 F_3

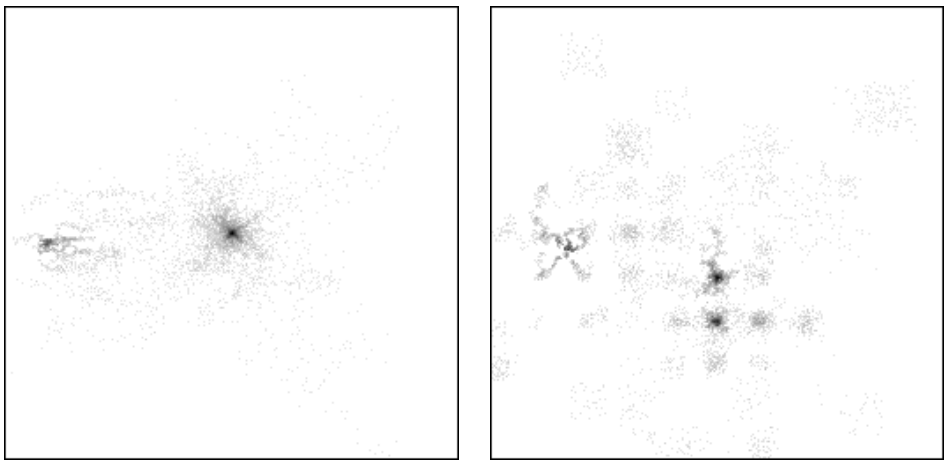
FIG. 7.14 – Représentation de la projection en deux dimensions des fonctions F_1 et F_3.

Chaque point de la carte d'exploration représente l'évaluation de la fonction dans l'intervalle recouvert par le point. Etant donné que la taille de la carte est de 256×256 pixels, chaque point recouvre un intervalle de côté $(B_i - b_i)/256$. L'intensité (en niveau de gris) d'un point correspond au logarithme du nombre d'évalutations effectuées dans l'intervalle correspondant. Quand on lance API sur F_1 et F_3 en version homogène avec les paramètres $n = 20$, $A_{\text{locale}} = 0.01$, $A_{\text{site}} = 0.1$, $P_{\text{locale}} = 30$, $p = 2$ et $T_3 = 10\,000$, on obtient les cartes de la figure 7.15. Le même jeux de paramètres mais en version hétérogène a été utilisé pour obtenir les cartes de la figure 7.16



F_1 F_3

FIG. 7.15 – Carte d'exploration des fonctions F_1 et F_3 dans le cas homogène.



F_1 F_3

FIG. 7.16 – Carte d'exploration des fonctions F_1 et F_3 dans le cas hétérogène.

L'avantage de l'hétérogénéité de la population prend ici une dimension visuelle. Concernant la fonction F_1, on constate qu'une population hétérogène se déplace beaucoup plus rapidement vers l'optimum placé au centre de l'espace de recherche et le nombre d'évaluations économisées permet d'affiner la recherche autour de cette position. Pour la fonction F_3, constituée de multiples optima locaux, la version homogène est tout à fait inadaptée puisqu'on constate que la majorité des explorations restent confinées dans un minimum local (où se trouve le nid). La version hétérogène explore plusieurs optima locaux et obtient en conséquence des performances bien meilleures.

Ces cartes d'exploration illustrent un point que nous avions déjà noté : l'importance de l'initialisation de la position du nid. La réponse la plus simple à cette difficulté peut être de deux types :

– le redémarrage régulier ou adaptatif de la recherche à une nouvelle position choisie aléatoirement ou de façon à couvrir l'espace de recherche. Cette méthode rentre dans le cadre du modèle multi-population ;
– l'utilisation d'une méthode d'initialisation basée sur une exploration méthodique de l'espace de recherche (quadrillage) ou sur une autre heuristique, par exemple un algorithme génétique[3].

7.6 Considérations théoriques

Cette section tente d'étudier de manière plus théorique le comportement de l'algorithme API. Il est notamment intéressant de comparer API aux techniques de recherche qui ont fait leurs preuves comme les algorithmes génétiques.

7.6.1 Interaction entre les sites mémorisés

On a pu constater que la gestion de plusieurs sites par une fourmi n'avait pas de répercussions notable sur les résultats. On peut deviner quelques éléments de réponse : bien que la fourmi puisse choisir le site qu'elle va visiter il semble équivalent de ne lui autoriser qu'un seul site en mémoire et qu'elle n'en construise un autre que lorsque ce site est épuisé. Cependant, le fait qu'elle dispose de plusieurs sites quand elle vient de subir un échec laisse supposer qu'elle perdra moins de temps à s'acharner sur un seul site pour pouvoir en explorer un autre. Quand on compare API à un AG (voir plus loin dans cette section), on constate que la sélection utilisée par API n'est peut-être pas assez forte. On peut envisager deux améliorations :

1. au lieu de choisir un site à visiter de manière uniformément aléatoire on peut diriger le choix de la fourmi en fonction du taux de succès de chaque site. On augmente ainsi la fréquence de visite des sites qui ont eu un passé glorieux ;
2. la sélection des AG dépend de l'adaptation d'une solution relativement à l'adaptation de toutes les solutions manipulées. Or API est distribué et il est inopportun de fournir à une fourmi l'adaptation des sites de toute la population pour pouvoir faire ses choix. On peut donc envisager que la fourmi choisisse le site à visiter non plus suivant son taux de succès mais suivant son adaptation par rapport aux autres sites actuellement dans la mémoire de la fourmi[4].

7.6.2 De l'utilité de mémoriser plusieurs sites

Dans cette section, nous allons montrer l'utilité de partager la mémoire de la fourmi sur plusieurs sites. Supposons que la fourmi dispose de trois sites de chasse en mémoire, chaque site de chasse a une certaine probabilité de fournir une proie quand la fourmi

[3]On peut ici envisager d'utiliser API à la place d'un AG avec un jeu de paramètres « plus large » ou encore de faire varier les paramètres d'API au cours des itérations à la façon du recuit simulé.

[4]Ou de façon plus directe suivant l'adaptation de la meilleure solution trouvée par la fourmi depuis le début de l'algorithme.

s'y déplace. Notons P_i cette probabilité pour le site s_i[5]. On peut modéliser le passage d'un site à un autre par une chaîne de Markov où les états représentent la position de la fourmi à chaque instant. La figure 7.17 donne graphiquement les probabilités de transition d'un état à l'autre. La figure 7.17(a) prend en compte le retour au nid à chaque sortie de la fourmi et la figure 7.17(b) représente les passages entre états tels que la modélisation algorithmique les a décrits (en incluant le retour au nid qui est systématique effectué entre deux sorties).

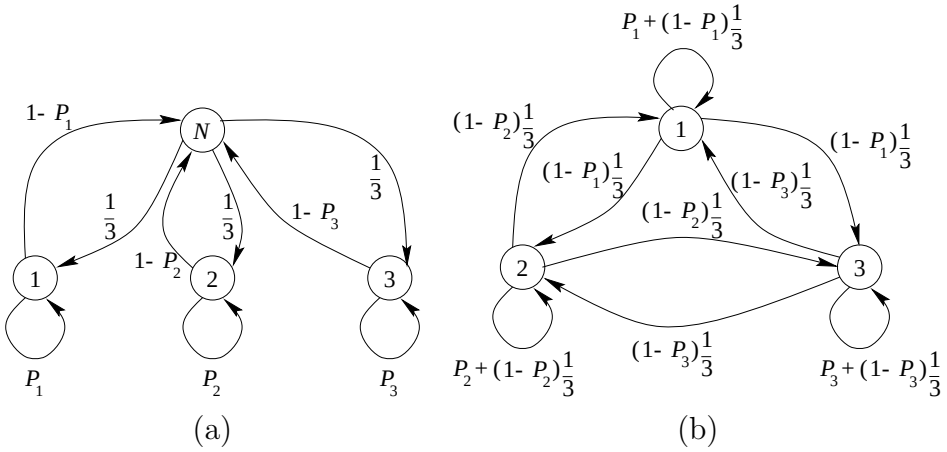


FIG. 7.17 – Modélisation du choix du site de chasse par une fourmi. Dans l'exemple présenté, La fourmi dispose de trois sites de chasse en mémoire et pour chaque site s_i, la probabilité de capturer une proie est notée P_i.
(a) : Chaîne de Markov prenant en compte le retour au nid.
(b) : Chaîne de Markov incluant le retour au nid dans le passage au site suivant.

Si on suppose que la fourmi n'oublie pas un site et ne s'en construit pas de nouveau, cela nous permet d'établir la probabilité que la fourmi soit sur un certain site à l'équilibre. Posons $\mathcal{M}$, la matrice de la chaîne de Markov construite :

$$\mathcal{M} = \begin{pmatrix} P_1 + \frac{1}{3}(1-P_1) & \frac{1}{3}(1-P_1) & \frac{1}{3}(1-P_1) \\ \frac{1}{3}(1-P_2) & P_2 + \frac{1}{3}(1-P_2) & \frac{1}{3}(1-P_2) \\ \frac{1}{3}(1-P_3) & \frac{1}{3}(1-P_3) & P_3 + \frac{1}{3}(1-P_3) \end{pmatrix} \tag{7.12}$$

On utilise le résultat suivant : Si $\mathcal{M}$ est une matrice de transition d'une chaîne de Markov régulière, alors :

$$\lim_{n\to\infty} \mathcal{M}^{(n)} = \begin{pmatrix} e_1 & e_2 & e_3 \\ e_1 & e_2 & e_3 \\ e_1 & e_2 & e_3 \end{pmatrix}$$

et $E = (e_1, e_2, e_3) \quad e_i > 0 \quad \forall i = 1, 2, 3$ est l'unique distribution de probabilité telle que :

$$E\mathcal{M} = E. \tag{7.13}$$

[5]Nous supposerons cette probabilité fixe dans le temps alors qu'à chaque amélioration locale le site est déplacé.

Grâce au système 7.13, on en déduit :

$$e_1 = \frac{(1-P_2)(1-P_3)}{(1-P_1)(1-P_2)+(1-P_1)(1-P_3)+(1-P_2)(1-P_3)} \quad (7.14)$$

$$e_2 = \frac{(1-P_1)(1-P_3)}{(1-P_1)(1-P_2)+(1-P_1)(1-P_3)+(1-P_2)(1-P_3)} \quad (7.15)$$

$$e_3 = \frac{(1-P_1)(1-P_2)}{(1-P_1)(1-P_2)+(1-P_1)(1-P_3)+(1-P_2)(1-P_3)} \quad (7.16)$$

Si on compare les sites 1 et 2, et que l'on sait que $P_1 > P_2$, comme $\frac{e_1}{e_2} = \frac{1-P_2}{1-P_1}$ d'après 7.14 et 7.15, on en déduit qu'à l'équilibre on aura $e_1 > e_2$ ce qui valide donc le fait que la modélisation algorithmique présentée précédemment conserve bien une répartition des sorties de la fourmi sur les sites de chasse proportionnelle à la probabilité de capturer sur chaque site.

7.6.3 Comparaison avec des méthodes voisines

La modélisation proposée du comportement de prédation de *Pachycondyla apicalis* manipule une population de solutions (l'ensemble des sites de chasse) et peut être rapprochée d'un certain nombre d'heuristiques existantes. C'est par les algorithmes d'évolution, qui manipulent aussi une population, que nous commençons ce rapprochement pour ensuite comparer la recherche locale effectuée par API à des heuristiques telles que la recherche tabou ou le recuit simulé.

L'ascension locale stochastique

Par rapport à l'algorithme RHC (voir l'algorithme 5.2, page 91), API utilise non pas un mais deux pas de recherche : le premier pas local d'amplitude A_{locale} est assimilable à celui de RHC, mais le pas global d'amplitude A_{site} n'existe pas dans RHC ainsi que le déménagement du nid. De plus, pour une fourmi donnée, l'exploration des sites n'est pas uniforme. Le succès associé aux sites biaise cette uniformité vers les meilleurs sites alors que RHC explore toujours le même point.

Les algorithmes génétiques

D'une manière générale, l'algorithme API peut être comparé avec les algorithmes manipulant une population de solutions. Les Algorithmes Génétiques (AG) en font partie. Le tableau 7.8 présente une comparaison des AG avec API. Voici points par points les similarités et différences entre API et les techniques issues des AG :

- les n fourmis manipulent p sites de chasse. La colonie de fourmis toute entière manipule donc $n \times p$ solutions ;
- la création d'un site correspond à l'arrivée d'une nouvelle solution dans la population. Pour un AG on parle d'immigration ;
- la recherche locale (opérateur $\mathcal{O}_{\text{explo}}$) revient à faire une mutation de la solution représentée par le site exploré ;

API	AG
n fourmis, p sites	$n \times p$ individus
création d'un site	immigration
recherche locale	opérateur de mutation
patience locale	pression de selection
déplacement du nid	redémarrage (*restart*)
recrutement	opérateur de sélection
multi-populations	niche (*Island Model*)

TAB. 7.8 – Comparaison de API avec les techniques issues des Algorithmes Génétiques (AG).

- la patience locale (P_{locale}) indique le nombre de tentatives d'amélioration d'un site. Cela revient à indiquer le niveau de confiance dans l'amélioration d'un site qu'une fourmi peut avoir. Si P_{locale} a une valeur faible, cela signifie que la fourmi sélectionne les meilleures solutions avec peu de tentatives sur chacune d'elles. A l'opposé, une grande valeur de P_{locale} indique que la fourmi dispose d'un plus grand nombre de tentatives pour améliorer une solution. La patience locale permet donc de régler le comportement de la fourmi en fonction des résultats qu'elle obtient. C'est en cela que l'on peut comparer la patience locale à la pression de sélection des algorithmes génétiques qui permet de favoriser plus ou moins les solutions les plus adaptées. Il faut cependant noter que la patience locale est un paramètre individuel alors que la pression de sélection agit sur l'ensemble de la population ;
- le déplacement du nid permet de relancer la recherche et éviter de cette façon les optima locaux. Les techniques de « *restart* » visent le même objectif ;
- le recrutement a pour effet de dupliquer les meilleures solutions manipulées par les fourmis ce qui est très similaire à la fonction de l'opérateur de sélection des AG ;
- on peut noter l'absence d'un équivalent chez API de l'opérateur de croisement des AGs.

Dans les AG manipulant une population finie de solutions, l'effet cumulé de la sélection et du croisement a l'inconvénient de faire disparaitre la diversité nécessaire à l'exploration. En quelque sorte, on peut considérer que dans un AG les solutions sont au départ uniformément distribuées dans l'espace de recherche et qu'après un certain nombre d'itérations, la population se concentre sur les points les plus sélectionnés. Seule la mutation introduit alors une certaine diversité. Dans API, le processus est inversé : les fourmis explorent à partir d'un point central, le nid, et tendent à s'en éloigner au fur et à mesure des itérations.

Si on élargit la comparaison à d'autres techniques évolutionnaires, on peut découvrir un certain nombre de points communs. Si on considère les stratégies d'évolution (SE), μ solutions parentes génèrent λ solutions filles par un mécanisme de mutation gaussienne. Comme les solutions sont des vecteurs réels, la ressemblance est plus forte avec API que dans le cas des AG manipulant des chaînes binaires. Il y alors principalement deux

solutions pour constituer la nouvelle population :

- les μ meilleures solutions filles parmi les λ construites forment la nouvelle population (notée (μ,λ)-SE) ;
- les μ meilleures solutions parmis les $\mu + \lambda$ sont sélectionnées pour constituer la nouvelle population (notée ($\mu + \lambda$)-SE).

C'est cette dernière qui se rapproche le plus de API où les solutions construites lors de l'exploration des sites de chasse ne sont conservées que si elles sont meilleures que le site qui les a générées. Cela dit, il ne faut pas perdre de vue que pour API la sélection se fait uniquement par rapport au site concerné et non pas par rapport à l'ensemble des sites mémorisés par les fourmis. C'est en cela que API est plus distribué qu'une stratégie d'évolution.

7.7 Applications

Dans cette section, nous présentons un certain nombre d'applications de l'algorithme API à des problèmes dont l'espace de recherche peut prendre différentes formes. L'objectif est de montrer que l'on peut réellement appliquer API à n'importe quel problème d'optimisation à partir du moment où l'on est capable de définir les opérateurs $\mathcal{O}_{\text{rand}}$ et $\mathcal{O}_{\text{explo}}$ sur l'espace de recherche.

7.7.1 Optimisation de fonctions numériques

Un certain nombre de résultats ont déjà été présentés pour ce type de problème dans la section 7.5. Nous présentons ici un certain nombre d'extensions spécifiques au problème de l'optimisation numérique.

Heuristique locale

Dans la nature, les fourmis ont tendance à choisir des directions les éloignant du nid lors de leur phase d'exploration locale. Afin de prendre en compte ce comportement qui tend à les éloigner du nid, quand une fourmi revient sur un site où elle vient de capturer une proie, elle continue sa recherche locale dans la même direction que la direction qui l'a conduite à la précédente capture. Ainsi, quand la fourmi améliore un site s en une position s', elle mémorise le vecteur $d = s' - s$ de $\mathbb{R}^l$ qui l'a amenée à cette découverte. Son exploration locale (équation 7.9) est alors orientée par $d = (d_i)_{i=1\ldots l}$ de la façon suivante :

$$s'_i = s_i + d_i + A\mathcal{U}[-0.5, 0.5](B_i - b_i) \tag{7.17}$$

où A est l'amplitude de l'exploration et $\mathcal{U}[-0.5, 0.5]$ est un nombre aléatoire uniformément tiré dans $[-0.5, 0.5]$.

Notons par API_d cette version de API. L'ajoût de cette heuristique locale s'apparente à l'utilisation d'une information locale de type gradient. Le tableau 7.9 donne les résultats obtenus pour les dix fonctions étudiés en utilisant les valeurs suivantes pour

F_i		API_h			API_{hd}		
		min	σ	F	min	σ	F
F_1	▶	1.35×10^{-6}	7.75×10^{-7}	2.56	2.42×10^{-5}	2.49×10^{-5}	7.93
F_2	▶	2.11×10^{-6}	1.99×10^{-6}	4.79	2.40×10^{-6}	1.89×10^{-6}	2.31
F_3	▶	4.56×10^{0}	3.33×10^{0}	7.18	9.26×10^{0}	4.43×10^{0}	2.40
F_4	▶	5.34×10^{-3}	6.27×10^{-3}	12.67	5.81×10^{-3}	3.89×10^{-3}	1.84
F_5	▶	2.01×10^{-1}	1.12×10^{-1}	2.42	3.93×10^{-1}	1.59×10^{-1}	3.29
F_6	▶	8.24×10^{-4}	2.16×10^{-3}	12.65	2.08×10^{-3}	3.07×10^{-3}	4.39
F_7	▶	1.17×10^{-1}	7.07×10^{-2}	5.19	1.93×10^{-1}	7.28×10^{-2}	2.64
F_8	▶	$9.83 \times 10^{+1}$	6.38×10^{0}	2.97	$1.14 \times 10^{+2}$	$1.02 \times 10^{+1}$	3.09
F_9	▶	$7.28 \times 10^{+1}$	2.87×10^{0}	2.72	$8.59 \times 10^{+1}$	3.85×10^{0}	3.93
F_{10}	▶	$9.40 \times 10^{+1}$	5.86×10^{0}	3.42	$1.10 \times 10^{+2}$	5.37×10^{0}	2.33

TAB. 7.9 – Résultats comparés de API_h et API_{hd} avec $T_3 = 10\,000$ (moyennes sur 30 essais). Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction, σ représente l'écart type et F le coefficient de Fisher définit dans la section 6.7.3, page 108.

les paramètres de API_h et API_{hd}[6] : $n = 20$, $P_{\text{locale}} = 30$, $p = 2$ et $T_3 = 10\,000$. Les figures 7.18 et 7.19 montrent l'évolution moyenne du minimum à chaque itération.

[6]Dans tous les cas nous utilisons des populations hétérogènes afin de limiter la taille du jeu de paramètres à tester. De plus, nous n'utilisons pas de recrutement.

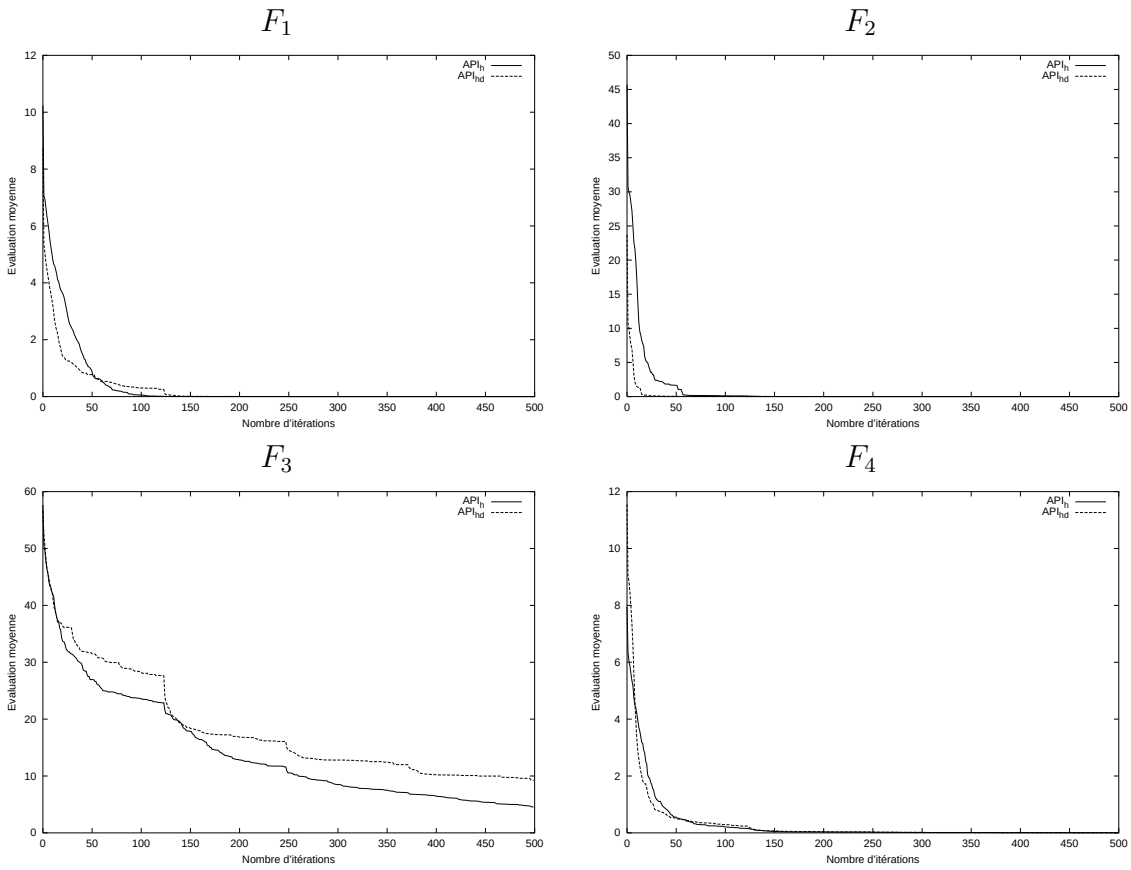


FIG. 7.18 – Courbes de convergence des fonctions F_1 à F_4 pour API$_h$ et API$_{hd}$.

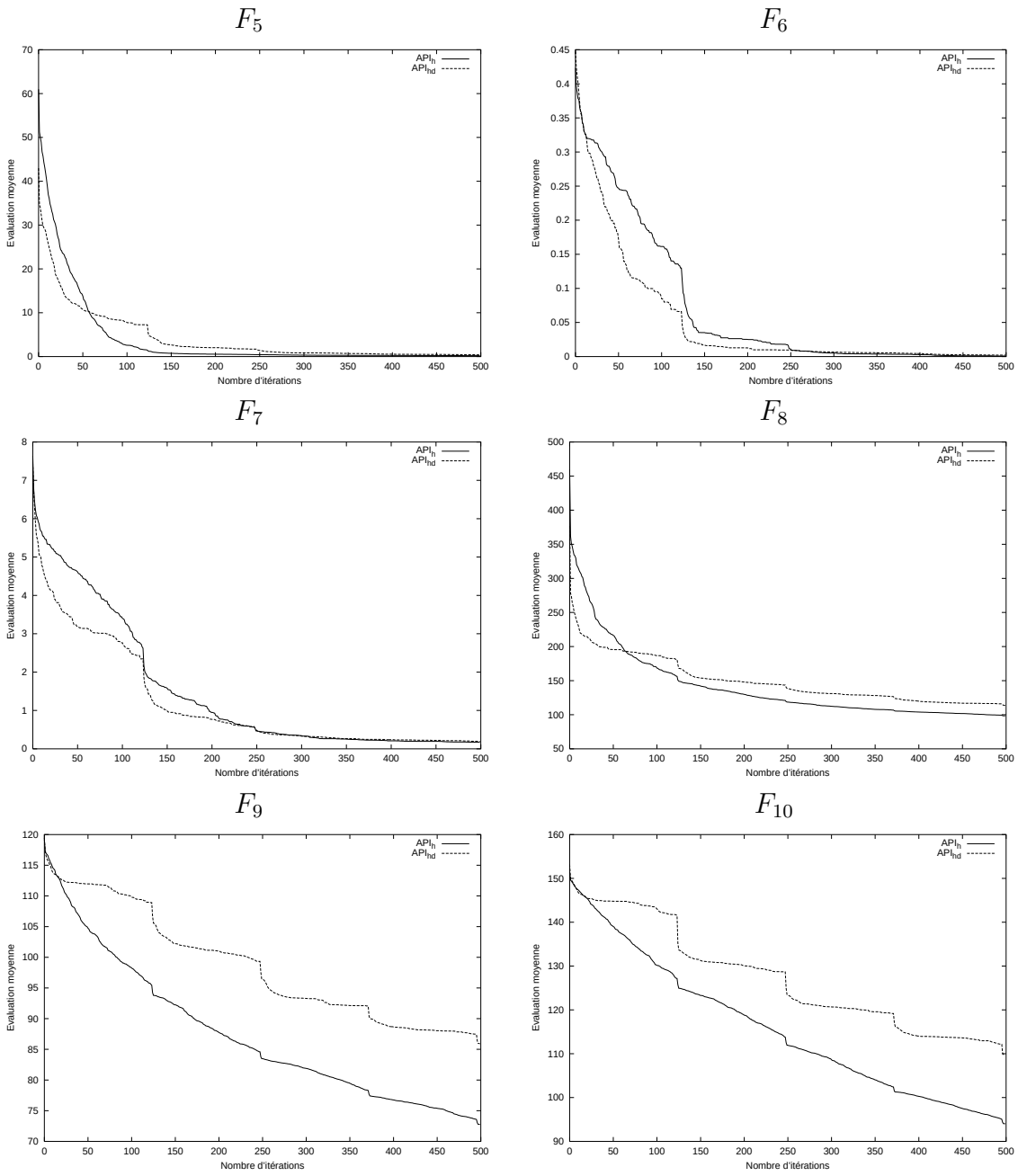


FIG. 7.19 – Courbes de convergence des fonctions F_5 à F_{10} pour API$_h$ et API$_{hd}$.

On peut regrouper ces courbes en trois catégories :

1. API$_h$ est pratiquement toujours meilleur que API$_{hd}$ (F_3, F_9 et F_{10}) ;
2. API$_{hd}$ est meilleur que API$_h$ pour au moins la première moitié des itérations (F_2, F_6 et F_7) ;
3. API$_{hd}$ est meilleur que API$_h$ pour au moins les cinquantes premières itérations (F_1, F_4, F_5 et F_8)

Ceci signifie que l'utilisation de la direction est efficace pour les premières itérations de API_{hd} mais que par la suite elle se transforme en handicap. La modification la plus immédiate est de prendre en compte la direction proportionnellement à l'amplitude A :

$$s'_i = s_i + Ad_i + A\mathcal{U}[-0.5, 0.5](B_i - b_i) \tag{7.18}$$

Les résultats obtenus avec cette variante (notée $\text{API}_{hd'}$) sont présentés dans le tableau 7.10. On peut constater que l'impact de cette variante est tout autrement plus efficace que ne l'était API_{hd} par rapport à API_h.

F_i	API_h			$\text{API}_{hd'}$		
	min	σ	F	min	σ	F
F_1	1.35×10^{-6}	7.75×10^{-7}	2.56	▶ 1.13×10^{-6}	6.96×10^{-7}	3.80
F_2	▶ 2.11×10^{-6}	1.99×10^{-6}	4.79	2.49×10^{-6}	3.67×10^{-6}	9.08
F_3	$4.56 \times 10^{+0}$	$3.33 \times 10^{+0}$	7.18	▶ $3.47 \times 10^{+0}$	$2.17 \times 10^{+0}$	2.20
F_4	5.34×10^{-3}	6.27×10^{-3}	12.67	▶ 3.77×10^{-3}	3.67×10^{-3}	4.73
F_5	▶ 2.01×10^{-1}	1.12×10^{-1}	2.42	2.20×10^{-1}	1.20×10^{-1}	1.83
F_6	▶ 8.24×10^{-4}	2.16×10^{-3}	12.65	1.92×10^{-3}	3.51×10^{-3}	3.62
F_7	1.70×10^{-1}	7.07×10^{-2}	5.19	▶ 1.63×10^{-1}	5.36×10^{-2}	3.53
F_8	▶ $9.83 \times 10^{+1}$	$6.38 \times 10^{+0}$	2.97	$9.91 \times 10^{+1}$	$8.07 \times 10^{+0}$	2.77
F_9	$7.28 \times 10^{+1}$	$2.87 \times 10^{+0}$	2.72	▶ $7.19 \times 10^{+1}$	$3.85 \times 10^{+0}$	2.61
F_{10}	$9.40 \times 10^{+1}$	$5.90 \times 10^{+0}$	3.42	▶ $8.96 \times 10^{+1}$	$4.46 \times 10^{+0}$	2.65

TAB. 7.10 – Résultats comparés de API_h et $\text{API}_{hd'}$ avec $T_3 = 10\,000$ (moyennes sur 30 essais). Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction, σ représente l'écart type et F le coefficient de Fisher défini dans la section 6.7.3, page 108.

D'une manière générale, l'utilisation d'une heuristique locale peut nettement améliorer les performances d'une méthode d'optimisation, la principale difficulté étant d'en connaître une. Nous avons montré que l'utilisation d'une information de direction par les fourmis pouvait dans un premier temps accélérer la convergence de API (pour 7 fonctions parmi les 11 utilisées) puis en l'améliorant nous avons obtenu des résultats compétitif avec API_h.

Opérateur d'exploration non uniforme

L'exploration $\mathcal{O}_{\text{explo}}$ est effectuée de façon uniforme dans l'intervalle $[s_i - \frac{A}{2}, s_i + \frac{A}{2}]$. De plus en plus de méthodes d'optimisation utilisent une distribution normale à la place de la distribution uniforme (voir par exemple (Sebag and Ducoulombier, 1998)). Nous avons introduit ce type d'exploration dans API (noté alors API_n pour signifier l'emploi d'une loi normale). L'opérateur $\mathcal{O}_{\text{explo}}$ agit alors de la façon suivante :

$$s'_i = s_i + \mathcal{N}(0, \frac{A}{2})(B_i - b_i) \tag{7.19}$$

où $\mathcal{N}(0, \frac{A}{2})$ est une loi normale centrée en 0 et d'écart type $\frac{A}{2}$.

F_i	API			API_n		
	min	σ	F	min	σ	F
F_1	9.85×10^{-3}	5.14×10^{-2}	27.99	▶ 6.86×10^{-5}	4.54×10^{-5}	3.07
F_2	2.32×10^{-1}	3.51×10^{-1}	6.45	▶ 3.42×10^{-3}	8.20×10^{-3}	11.28
F_3	$2.05 \times 10^{+1}$	$1.09 \times 10^{+1}$	2.15	▶ $7.79 \times 10^{+0}$	$4.08 \times 10^{+0}$	2.76
F_4	$1.52 \times 10^{+0}$	$1.68 \times 10^{+0}$	4.73	▶ 5.03×10^{-2}	2.89×10^{-2}	2.04
F_5	$4.08 \times 10^{+0}$	$2.91 \times 10^{+0}$	4.16	▶ 6.86×10^{-1}	2.72×10^{-1}	8.47
F_6	2.61×10^{-1}	1.91×10^{-1}	1.33	▶ 3.82×10^{-2}	6.47×10^{-2}	9.17
F_7	$6.35 \times 10^{+0}$	$1.78 \times 10^{+0}$	6.03	▶ $1.37 \times 10^{+0}$	$1.65 \times 10^{+0}$	7.06
F_8	$2.04 \times 10^{+2}$	$8.40 \times 10^{+1}$	4.27	▶ $1.36 \times 10^{+2}$	$2.87 \times 10^{+1}$	1.77
F_9	$1.06 \times 10^{+2}$	$7.92 \times 10^{+0}$	2.97	▶ $9.32 \times 10^{+1}$	$5.92 \times 10^{+0}$	2.44
F_{10}	$1.38 \times 10^{+2}$	$7.56 \times 10^{+0}$	2.56	▶ $1.25 \times 10^{+2}$	$7.79 \times 10^{+0}$	2.63

TAB. 7.11 – Résultats comparés de API (population homogène) et API_n (population homogène et $\mathcal{O}_{\text{explo}}$ non uniforme. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction, σ représente l'écart type et F le coefficient de Fisher défini dans la section 6.7.3, page 108.

Le tableau 7.11 présente les résultats obtenus dans le cas homogène en fixant $n = 20$, $A_{\text{site}} = 0.1$, $A_{\text{locale}} = 0.01$, $P_{\text{locale}} = 30$, $p = 2$ et $T_3 = 10\,000$. Le tableau 7.12 présente les résultats obtenus dans le cas hétérogène.

Il ressort de ces deux tableaux que API_n est beaucoup plus performant que API alors que la différence n'est plus perceptible entre API_{hn} et API_h. Pour une population homogène, le choix des paramètres est en général beaucoup plus sensible. L'utilisation d'une densité normale semble diminuer cette sensibilité en permettant une répartition non uniforme des essais autour d'un point de l'espace de recherche. Par contre dans le cas hétérogène cela n'a plus d'influence puisque les paramètres de la population permettent de couvrir plus de jeux de paramètres.

Comparaisons avec d'autres heuristiques

- **Recherche aléatoire pure**. Cette méthode est la plus simple que l'on puisse concevoir : générer T_3 points dans $\mathcal{S}$ et retenir le meilleur (voir l'algorithme 5.2). En fixant certains paramètres de API à des valeurs extrêmes, on peut tester cette méthode que l'on note API_a. Il suffit de poser $A_{\text{site}} = 2$, ce qui permet de générer uniformément un point dans tout l'espace de recherche et $p \geq P_N$ pour que les fourmis n'aient que le temps de construire leurs sites de chasse sans pouvoir les explorer puisque le nid devra être déplacé. Le tableau 7.13 donne les résultats obtenus avec API_a en comparaison avec une version hétérogène ($n = 20$, $p = 2$, $P_{\text{locale}} = 20$, avec recrutement). Les résultats sont nettement en faveur de API_h ce qui prouve seulement que la stratégie de recherche de API est meilleure qu'une recherche aléatoire.
- **Ascension locale stochastique avec redémarrages multiples**. Tout comme pour la recherche aléatoire, API peut être paramétré pour reproduire le compor-

F_i		API_h				API_{hn}		
		min	σ	F		min	σ	F
F_1	▶	1.05×10^{-6}	7.88×10^{-7}	3.39		2.47×10^{-6}	1.74×10^{-6}	4.07
F_2		4.98×10^{-6}	9.73×10^{-6}	18.17	▶	4.66×10^{-6}	6.39×10^{-6}	15.97
F_3	▶	$4.08 \times 10^{+0}$	$2.69 \times 10^{+0}$	3.16		$4.35 \times 10^{+0}$	$2.73 \times 10^{+0}$	4.02
F_4		4.79×10^{-3}	3.37×10^{-3}	1.48	▶	4.31×10^{-3}	4.10×10^{-3}	2.68
F_5	▶	1.60×10^{-1}	9.15×10^{-2}	2.50		2.33×10^{-1}	1.29×10^{-1}	3.07
F_6		1.07×10^{-3}	2.65×10^{-3}	8.22	▶	7.92×10^{-4}	1.89×10^{-3}	16.96
F_7	▶	1.60×10^{-1}	5.56×10^{-2}	2.90		1.76×10^{-1}	6.53×10^{-2}	5.72
F_8	▶	$9.89 \times 10^{+1}$	$6.96 \times 10^{+0}$	2.51		$1.14 \times 10^{+2}$	$7.71 \times 10^{+0}$	3.18
F_9	▶	$7.29 \times 10^{+1}$	$3.98 \times 10^{+0}$	3.04		$8.36 \times 10^{+1}$	$4.24 \times 10^{+0}$	2.83
F_{10}		$9.33 \times 10^{+1}$	$6.12 \times 10^{+0}$	3.02	▶	$9.28 \times 10^{+1}$	$5.13 \times 10^{+0}$	3.04

TAB. 7.12 – Résultats comparés de API_h (population hétérogène) et API_{hn} (population hétérogène et $\mathcal{O}_{\text{explo}}$ non uniforme) ($T_3 = 10\,000$). Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction, σ représente l'écart type et F le coefficient de Fisher défini dans la section 6.7.3, page 108.

tement de MSRHC (voir l'algorithme 5.2). Pour un nombre limité d'évaluations de chaque fonction, nous fixons les paramètres suivants :
- $n = 1$;
- $p = 1$;
- la patience du nid (P_N) pour API_h avec $n = 20$, $p = 2$, $P_{\text{locale}} = 20$, est de 84 (formule 7.1). Pour MSRHC, nous n'utilisons qu'une seule fourmi, le redémarrage de la recherche, qui correspond au déplacement du nid à une position aléatoire, est donc effectuée tous les $84 \times 20 = 1680$ itérations ;
- l'amplitude des déplacements est fixée à $A_{\text{site}} = A_{\text{locale}} = 0.5$.

Les résultats obtenus sont donnés dans le tableau 7.14. Bien que les résultats obtenus par MSRHC soient meilleurs que ceux obtenus par la recherche aléatoire, API_h reste plus performant.

7.7.2 Optimisation combinatoire : Problème du voyageur de commerce

Le problème du voyageur de commerce (PVC, *Traveler Salesman Problem* : TSP) est un problème très classique en optimisation combinatoire[7]. Il est de plus en plus improbable de présenter une méthode surpassant toutes celles existantes (voir par exemple (Reinelt, 1994) pour un aperçu des différentes approches). Cependant il est intéressant de voir comment API peut s'appliquer simplement à ce problème. Comme nous l'avons vu au chapitre 2 de nombreux algorithmes inspirés des fourmis sont nés de l'analogie entre ce problème et la recherche de nourriture par les fourmis réelles (Stützle and Dorigo, 1999b). Nous avons notamment constaté que ces heuristiques sont essentiellement

[7]L'énoncé du PVC est donné au chapitre 2.

F_i	API_a			API_h			
	min	σ	F		min	σ	F
F_1	1.02×10^{-1}	6.16×10^{-2}	3.04	▶	1.71×10^{-6}	1.12×10^{-6}	4.58
F_2	6.03×10^{-3}	5.59×10^{-3}	3.11	▶	3.33×10^{-6}	5.87×10^{-6}	16.02
F_3	$1.58 \times 10^{+1}$	$3.63 \times 10^{+0}$	2.45	▶	$3.63 \times 10^{+0}$	$1.82 \times 10^{+0}$	2.33
F_4	2.23×10^{-1}	1.04×10^{-1}	2.62	▶	2.60×10^{-3}	2.90×10^{-3}	2.69
F_5	$4.03 \times 10^{+0}$	$1.46 \times 10^{+0}$	2.02	▶	1.67×10^{-1}	8.95×10^{-2}	3.49
F_6	2.12×10^{-2}	1.28×10^{-2}	2.46	▶	3.51×10^{-4}	1.20×10^{-3}	26.74
F_7	$1.29 \times 10^{+0}$	3.25×10^{-1}	2.62	▶	1.41×10^{-1}	4.05×10^{-2}	2.11
F_8	$1.46 \times 10^{+2}$	$1.05 \times 10^{+1}$	2.43	▶	$9.64 \times 10^{+1}$	$8.53 \times 10^{+0}$	3.03
F_9	$1.05 \times 10^{+2}$	$1.83 \times 10^{+0}$	3.47	▶	$7.03 \times 10^{+1}$	$3.53 \times 10^{+0}$	2.18
F_{10}	$1.24 \times 10^{+2}$	$3.27 \times 10^{+0}$	3.90	▶	$8.44 \times 10^{+1}$	$5.73 \times 10^{+0}$	3.41

TAB. 7.13 – Résultats comparés de API_a (recherche aléatoire) et API_h avec $T_3 = 10\,000$. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction, σ représente l'écart type et F le coefficient de Fisher défini dans la section 6.7.3, page 108.

constructives

API pour le PVC

L'adaptation la plus immédiate de API à la résolution de problèmes combinatoires est de procéder par la modification successive d'un ensemble de solutions, contrairement à une approche constructive des solutions. Ce type d'approche a par exemple été utilisé pour le problème de l'assignement quadratique exposé au chapitre 2. La position d'une fourmi dans l'espace de recherche est équivalente à un chemin hamiltonien dans le graphe des villes. Le déplacement d'une fourmi correspond alors à une modification de ce chemin. Un certain nombre d'heuristiques sont envisageables pour accomplir cette modification. Nous en avons expérimenté quelques unes avec la contrainte qu'elles soient paramétrables par une amplitude. Cela signifie qu'une valeur élevée de l'amplitude permet de modifier assez fortement une solution alors qu'une valeur faible ne doit pas trop perturber un chemin. La plupart des heuristiques efficaces utilisent des heuristiques proches du problème permettant de modifier un chemin pour en trouver un plus court. L'algorithme ACS, par exemple (voir le chapitre 2), construit des solutions en utilisant la coopération des fourmis puis chaque chemin est amélioré (si possible) par l'heuristique 2-opt ou 3-opt. Voici les techniques de modification d'un chemin, correspondant à l'opérateur $\mathcal{O}_{\text{explo}}$, que nous avons expérimentées :

- la permutation simple : deux villes sont échangées au hasard dans la séquence. L'amplitude correspond dans ce cas au nombre de permutations effectuées (une amplitude de 0.1 pour un problème à 100 villes occasionera $0.1 \times 100 = 10$ permutations).
- la permutation circulaire : plusieurs villes sont permutées dans la séquence, le nombre de villes permutées représente l'amplitude du déplacement d'une fourmi.
- l'insertion : cette méthode consiste à choisir une ville au hasard et à l'insérer entre deux autres villes de la séquence. L'amplitude correspond alors au nombre

F_i		MSRHC				API_h		
		min	σ	F		min	σ	F
F_1		2.00×10^{-2}	1.34×10^{-2}	4.71	▶	1.84×10^{-6}	1.15×10^{-6}	3.38
F_2		3.71×10^{-3}	3.53×10^{-3}	2.66	▶	2.37×10^{-6}	2.07×10^{-6}	2.25
F_3		$1.01 \times 10^{+1}$	$2.30 \times 10^{+0}$	2.04	▶	$3.61 \times 10^{+0}$	$2.23 \times 10^{+0}$	4.18
F_4		1.03×10^{-1}	4.44×10^{-2}	1.87	▶	2.78×10^{-3}	3.11×10^{-3}	3.54
F_5		$1.70 \times 10^{+0}$	3.46×10^{-1}	2.56	▶	1.57×10^{-1}	7.85×10^{-2}	1.78
F_6		1.06×10^{-2}	1.80×10^{-3}	10.14	▶	8.47×10^{-4}	2.41×10^{-3}	12.21
F_7		8.47×10^{-1}	2.47×10^{-1}	1.90	▶	1.52×10^{-1}	6.17×10^{-2}	3.98
F_8		$1.07 \times 10^{+2}$	$5.86 \times 10^{+0}$	2.04	▶	$9.79 \times 10^{+1}$	$5.83 \times 10^{+0}$	2.94
F_9		$8.14 \times 10^{+1}$	$1.78 \times 10^{+0}$	2.35	▶	$7.59 \times 10^{+1}$	$3.92 \times 10^{+0}$	2.59
F_{10}	▶	$8.02 \times 10^{+1}$	$2.05 \times 10^{+0}$	3.30		$9.04 \times 10^{+1}$	$6.57 \times 10^{+0}$	2.09

TAB. 7.14 – Résultats comparés de MSRHC et API_h avec $T_3 = 10\,000$. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque fonction, σ représente l'écart type et F le coefficient de Fisher défini dans la section 6.7.3, page 108.

d'insertions effectuées,

– permutation probabiliste : si la permutation de deux villes choisies au hasard dans la séquence augmente la longueur du chemin, la probabilité que cette permutation soit effectuée est inversement proportionnelle à l'augmentation occasionnée. Si la permutation diminue la longueur du chemin, elle est systématiquement effectuée.

Ces méthodes de perturbation d'une séquence ont le mérite d'être simples et peu coûteuses au niveau du temps de calcul. On peut cependant utiliser des méthodes améliorant localement une solution comme technique de déplacement d'une fourmi, par exemple on peut appliquer une méthode gloutonne sur une portion du chemin. On peut également envisager de différencier la création d'un site de chasse de l'exploration d'un site et associer à ces deux tâches des techniques différentes de modification de la séquence.

Expérimentations

Afin de tester cette version combinatoire d'API nous nous sommes basés sur un ensemble de jeux de tests très utilisés dans le domaine. Les problèmes décrits dans le tableau 7.15 sont tirés de la liste de problèmes du voyageur de commerce TSPLIB (Reinelt, 1995).

En ce qui concerne l'opérateur $\mathcal{O}_{\text{explo}}$ il est apparu que les meilleurs résultats étaient obtenus en utilisant l'insertion, à la fois pour la création de site que de l'exploration locale. Nous avons comparé API à ACS afin d'en mesurer les performances. Comme ACS utilise une heuristique supplémentaire à chaque création d'un chemin, nous avons utilisé dans les deux cas, pour API et ACS, l'heuristique 2-Opt. Dans un objectif d'équité, les paramètres de chaque algorithme ont été ajustés pour que le même nombre d'appels à 2-Opt soit effectué. Le tableau 7.16 donne les résultats obtenus par ACS dont les paramètres sont les suivants : $n = 20$, $T_3 = 200$, $\beta = 2.0$, $q_0 = 0.98$, $\rho = 0.1$ et $\tau_0 = 0.2$. Les listes candidates n'ont pas été utilisées. API a été utilisé en version

Problème	V	Optimum
P16	16	16
P49	49	49
EIL51	51	415
EIL76	76	538
KROA100	100	21282
D198	198	15780
LIN318	318	42029
PCB442	442	50778
RAT783	783	8806

TAB. 7.15 – Jeux de test pour l'évaluation de API sur le PVC. V correspond au nombre de villes du problème.

Problème	$\bar{s}$	$[\sigma]$	s^+	Durée
P16	16.00	[0.00]	16	0.51
P49	49.00	[0.00]	49	6.27
EIL51	430.76	[6.75]	419	7.26
EIL76	554.88	[4.85]	542	18.84
KROA100	50995.44	[10098.60]	26501	37.28
D198	16977.78	[144.18]	16606	215.30
LIN318	47043.08	[369.42]	46455	781.75
PCB442	57476.56	[695.15]	56026	1966.75
RAT783	10103.17	[71.80]	9880	11349.85

TAB. 7.16 – Résultats obtenus par ACS. $\bar{s}$ est la longueur moyenne du plus court chemin trouvé sur 50 essais et σ, l'écart type correspondant. s^+ est la longueur du plus court chemin trouvé et la durée, donnée en secondes, est donnée en moyenne sur les 50 essais.

hétérogène pour produire les résultats présentés dans le tableau 7.17. Les paramètres sont les suivants : $P_N = 44$, $p = 2$, $n = 20$ et $T_3 = 200$. Dans la colonne API$_h$+glouton, nous avons fait apparaître les résultats obtenus quand le nid est initialisé par une heuristique gloutonne[8] au lieu d'une initialisation aléatoire.

On constate tout d'abord que l'initialisation gloutonne de la position du nid confère un avantage à API surtout perceptible pour les problèmes comportant un nombre important de villes.

Les résultats obtenus par API sont inférieurs en qualité à ceux obtenus par ACS. Il faut cependant tenir compte des remarques suivantes :

– ACS a été développé spécifiquement pour le PVC et l'optimisation combinatoire ;
– ACS utilise l'information de distance entre les villes comme heuristique locale

[8]La première ville est choisie aléatoirement, puis la séquence est construite en choisissant pour chaque position la ville la plus proche de la dernière choisie parmi les villes qui n'ont pas encore été traversées.

Problème	API_h				API_h+glouton		
	$\bar{s}$	$[\sigma]$	s^+	Durée	$\bar{s}$	$[\sigma]$	s^+
P16	16.00	[0.00]	16	0.06	16.00	[0.00]	16
P49	49.14	[0.40]	49	0.39	49.14	[0.40]	49
EIL51	439.50	[9.45]	419	0.41	435.98	[7.36]	424
EIL76	591.48	[14.54]	566	0.99	573.12	[11.33]	553
KROA100	59581.20	[1332.12]	56879	1.85	55969.04	[660.94]	54987
D198	21775.02	[1594.90]	18769	7.67	18047.42	[518.47]	17368
LIN318	75989.66	[3598.25]	67902	22.32	52970.88	[1111.55]	50587
PCB442	94632.06	[3527.09]	85131	58.61	61324.48	[1089.31]	58002
RAT783	18893.86	[846.62]	17081	272.35	10944.18	[234.73]	10472

TAB. 7.17 – Résultats obtenus par API pour le PVC. $\bar{s}$ est la longueur moyenne du plus court chemin trouvé sur 50 essais et σ, l'écart type correspondant. s^+ est la longueur du plus court chemin trouvé et la durée, donnée en secondes, est donnée en moyenne sur les 50 essais.

alors que API n'utilise pas cette information ;
- le temps de calcul de API est très inférieur au temps de calcul de ACS. Par exemple, pour un problème à 783 villes, API nécessite environ 272 secondes alors que ACS en nécessite 11350 pour le même nombre d'itérations ;

Pour accélérer ACS, il faudrait par exemple utiliser des listes candidates, ce qui réduirait le temps de construction d'une séquence et pour améliorer API, il faudrait que l'opérateur $\mathcal{O}_{\text{explo}}$ soit un peu plus perfectionné qu'une simple série d'insertions aléatoires.

7.7.3 Apprentissage de Chaînes de Markov Cachées

Les Chaînes de Markov Cachées

Les Chaînes de Markov Cachées (CMC), ou Modèles de Markov Cachés (*Hidden Markov Models : HMM*) sont des processus stochastiques simples en applications, riches en propriétés et fondés sur des bases mathématiques solides (Slimane and Asselin de Beauville, 1996). Les CMC sont des outils de modélisation très utilisés en reconnaissance des formes (Rabiner, 1989), et en particulier appliqués à la reconnaissance de visages (Samaria, 1994; Slimane et al., 1996a) ou la prévision de série temporelles (Slimane et al., 1998).

> « On emploie les CMC pour modéliser des phénomènes dont on estime n'observer que des manifestations ou des réalisations. Le phénomène responsable de ces réalisations reste caché. »(Brouard, 1999)

Dans ce document, nous ne décrivons pas en détail les CMC et leurs algorithmes, nous nous contenterons d'y voir un problème d'optimisation. Nous nous intéressons plus particulièrement aux CMC discrètes et stationnaires. Formellement, une CMC λ est représentée par trois matrices A, B et Π. A correspond à la matrice de transition entre les états cachés. S'il y a N états cachés $s_1, \ldots, s_N$ alors $(a_{ij})_{1 \leq i,j \leq N} = A \in$

$\mathcal{M}_{N\times N}(\mathbb{R})$ et A est stochastique. a_{ij} est la probabilité de passer de l'état i à l'état j. La matrice B fait correspondre les N états cachés aux M symboles observables $v_1, \ldots, v_M$. $(b_{jk})_{1\leq j\leq N, 1\leq k\leq M} = B \in \mathcal{M}_{N\times M}(\mathbb{R})$ est aussi stochastique et b_{jk}, que l'on note aussi $b_j(v_k)$, donne la probabilité que la CMC se trouvant dans l'état j génère le symbole v_k. Enfin, le vecteur $\Pi = (\pi_1, \ldots, \pi_N)$, lui aussi stochastique, donne la probabilité que la CMC soit dans un état donné à son initialisation.

Il y a trois problèmes fondamentaux concernant les CMC :

- le problème de l'évaluation. Étant donnée une observation o (composée de K symboles appartenant à $\{v_1, \ldots, v_M\}$), on désire connaitre la probabilité $P(o|\lambda)$ que la CMC λ génère cette observation. L'algorithme Forward effectue ce calcul en N^2K opérations ;
- le problème de la détermination du chemin d'état. Étant données une CMC λ et une observation o, on cherche la succession des états cachés la plus probablement suivie par λ lorsque celle-ci génère o. Ce problème est résolu par l'algorithme de Viterbi ;
- le problème d'apprentissage. Étant donnée une observation o on cherche la CMC λ^* maximisant la probabilité de générer o. Deux cas se présentent : (i) l'architecture de la CMC est connue (i.e. N, le nombre d'états), il s'agit de découvrir les valeurs des matrices de λ. On parle alors d'apprentissage supervisé. (ii) le nombre d'états cachés est inconnu et sa détermination fait donc partie du problème, l'apprentissage est alors dit non supervisé. Quand le nombre d'états est fixé, l'algorithme Baum-Welch (BW) permet d'obtenir les valeurs des matrices A^*, B^* et Π^* en partant d'une CMC. Le principal inconvénient de cet algorithme est qu'il se comporte comme un algorithme de descente de gradient et ne garantit en conséquence que la découverte d'un optimum local.

C'est le problème d'apprentissage qui nous intéresse ici puisqu'il s'agit d'un problème d'optimisation : maximiser $P(o|\lambda)$. Les algorithmes génétiques ont par exemple été utilisés pour « servir » à l'algorithme BW des CMC d'initialisation (Slimane et al., 1996b). Le principe est le suivant : une population de CMC est générée puis évaluée relativement à l'observation que l'on désire apprendre (par l'algorithme Forward). Les CMC sont ensuite sélectionnées, croisées et mutées en adaptant les opérateurs génétiques classiques aux CMC. L'algorithme BW est utilisé en tant qu'opérateur de recherche locale et il peut être utilisé à chaque itération de l'algorithme génétique (AG∪BW)[9] ou à la fin du processus génétique (AG+BW) afin d'améliorer la solution obtenue. Le cas non-supervisé est pris en compte en manipulant une population de CMC dont le nombre d'états cachés est variable (algorithme GHOSP : *Genetic Hybrid Optimization and Search of Parameters*)(Slimane et al., 1999).

API pour l'optimisation des CMC

Nous proposons de remplacer les algorithmes génétiques par l'algorithme API. L'apprentissage des CMC est un problème intéressant pour API puisque la dimension de l'espace de recherche est assez importante : il faut trouver les valeurs des composantes

[9] AG∪BW est considéré comme lamarkien puisque les améliorations obtenues par BW seront conservées pour la génération suivante.

des matrices A, B et Π de la CMC λ. La position d'une fourmi est une CMC ce qui revient à un vecteur composé de $N^2 + N \times M + N$ valeurs réelles. La fonction f à maximiser correspond ici à $P(o|\lambda)$. Les opérateurs $\mathcal{O}_{\text{rand}}$ et $\mathcal{O}_{\text{explo}}$ développés pour l'optimisation de fonctions numériques peuvent être utilisés tels quels. C'est à ce niveau que se situe le principal avantage d'API sur les AG : la conception des opérateurs de croisement ou de mutation n'est pas toujours évidente, surtout si la population est constituée de CMC comportant un nombre d'états variable.

Expérimentations

API a été implanté en C++ pour réutiliser les classes de gestion des CMC ainsi que les algorithmes associés développés par Thierry Brouard dans le cadre de sa thèse (Brouard, 1999).

Les tests portent sur 12 observations artificielles présentées dans le tableau 7.18.

Obs.	Description	T	M
o_1	1, 2, 3, 1, 4, 2, 4, 4	8	4
o_2	1, 2, 2, 1, 1, 1, 2, 2, 2, 1	10	2
o_3	1, 2, 3, 2, 1, 2, 5, 4, 1, 2, 4	11	5
o_4	1, 1, 1, 2, 2, 1, 2, 3	8	3
o_5	1, 1, 1, 2, 2, 2, 3, 3, 3	9	3
o_6	1, 2, 3, 1, 2, 3, 1, 2, 3	9	3
o_7	1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3	12	3
o_8	1, 1, 2, 2, 3, 3, 4, 4, 1, 2, 3, 4	12	4
o_9	1, 1, 1, 1, 2, 2, 2, 2	8	2
o_{10}	1, 5, 4, 6, 3, 2, 5, 4, 1, 2, 3, 6, 5, 4, 1, 2, 5, 6, 3, 1	20	6
o_{11}	2, 3, 4, 6, 5, 4, 1, 1, 1, 6, 1, 5, 1, 1, 5	15	6
o_{12}	5, 4, 5, 4, 4, 5, 4, 8, 5, 4, 1, 1, 1, 2, 2	15	8

Tab. 7.18 – Les 12 observations utilisées comme jeu de tests pour API appliqué aux CMC. T correspond à la taille de l'observation et M au nombre de symboles utilisés.

Les résultats présentés dans le tableau 7.19 sont tirés de (Brouard, 1999) pour les algorithmes RAND, AG, AG+BW et AG∪BW. Chaque valeur correspond à la moyenne des meilleures valeurs de $P(o_i|\lambda)$ obtenues sur 50 essais pour un nombre d'états variant de deux à dix. Voici la description de chacun de ces algorithmes et de leurs paramètres :

- RAND (recherche aléatoire) : pour chaque nombre d'états, 20 CMC sont générées et le résultat renvoyé correspond à l'évaluation par l'algorithme Forward de la meilleure CMC trouvée.
- AG (algorithme génétique) : pour chaque nombre d'états, 20 individus sont manipulés et 10 itérations sont données à l'AG. Parmi les 20 individus, 16 sont utilisés comme parents et 8 enfants sont générés à chaque itération ce qui implique qu'à chaque itération l'AG génère 8 CMC. Quatre enfants remplacent alors les quatre parents les moins forts. À l'initialisation, les 20 parents générés sont évalués. Au

total, l'AG génère et évalue $20 + 10 \times 8 = 100$ CMC pour un nombre d'états cachés donné.
- AG+BW : hybridation séquentielle de l'AG et BW : l'algorithme BW est appliqué sur la meilleure CMC trouvée par l'AG.
- AG∪BW : hybridation coopérative de l'AG et BW : l'algorithme BW est appliqué sur chaque CMC construite par l'AG.

La version de API (API_h∪BW) utilisée s'inspire de l'hybridation coopérative de l'AG et BW : chaque CMC construite par les fourmis est modifiée par BW pour en maximiser l'évaluation. Nous utilisons une population hétérogène de 10 fourmis pour qu'avec 10 itérations, 100 CMC soit générées[10], les autres paramètres ont les valeurs suivantes : $P_{\text{locale}} = 3$, $p = 2$ et $T_2 = 10$ (le recrutement n'est pas utilisé).

Obs.	RAND	AG	AG+BW	AG∪BW	API_h∪BW
o_1	3.47×10^{-5}	5.51×10^{-5}	9.47×10^{-2}	1	1
o_2	3.61×10^{-4}	6.21×10^{-4}	1.78×10^{-2}	1	9.44×10^{-1}
o_3	2.12×10^{-8}	1.38×10^{-7}	8.73×10^{-3}	2.5×10^{-1}	3.71×10^{-1}
o_4	4.79×10^{-4}	5.18×10^{-4}	1.61×10^{-2}	9.95×10^{-1}	1
o_5	6.87×10^{-5}	2.06×10^{-4}	5.76×10^{-2}	1	1
o_6	1.28×10^{-4}	1.53×10^{-4}	1	1	1
o_7	2.52×10^{-6}	7.03×10^{-6}	1.34×10^{-3}	2.70×10^{-1}	5.82×10^{-1}
o_8	7.41×10^{-8}	1.72×10^{-7}	8.58×10^{-3}	2.5×10^{-1}	4.44×10^{-1}
o_9	1.83×10^{-3}	4.61×10^{-3}	1.70×10^{-1}	1	1
o_{10}	4.24×10^{-16}	1.60×10^{-15}	1.09×10^{-7}	1.34×10^{-3}	2.32×10^{-3}
o_{11}	1.65×10^{-11}	5.36×10^{-11}	4.11×10^{-5}	9.24×10^{-3}	3.51×10^{-2}
o_{12}	3.52×10^{-13}	1.63×10^{-11}	1.94×10^{-4}	1.85×10^{-2}	5.19×10^{-2}

TAB. 7.19 – Résultats obtenus par différentes méthodes pour l'apprentissage des CMC. Chaque valeur correspond à la moyenne sur 50 essais de l'évaluation de la meilleure CMC ($P(o_i|\lambda)$). Les meilleures valeurs atteignent la valeur 1.

Nous constatons que API_h∪BW donne de meilleurs résultats que l'ensemble des autres méthodes et notamment par rapport à AG∪BW qui en est le plus proche. Le même type d'expérience a été mené avec une version de API_h sans utiliser BW (donc comparable à l'AG seul). Les résultats de cette campagne de tests ne sont pas présentés ici mais API_h s'était montré équivalent à l'AG seul. Nous pouvons en conclure que API_h semble plus adapté à l'initialisation de BW que l'AG. Par rapport à l'AG, API ne croise pas les solutions entre elles, ce qui signifie que l'apport du croisement dans l'AG n'est pas significatif pour l'initialisation de BW.

Ces résultats encourageants sont à modérer en considérant que les espaces de recherche représentés par le jeu de tests utilisé sont relativement réduits si on les compare aux espaces de recherche couramment manipulés pour des applications réelles des CMC. Dans le cas de l'apprentissage d'images, par exemple, la taille d'une observation correspond au nombre de pixels dans l'image et le nombre de symboles correspond au nombre

[10]En fait il y aura 101 CMC générées puisqu'à l'initialisation du nid une CMC est crée et évaluée.

de niveaux de gris considérés. Il conviendrait donc de tester $\text{API}_h \cup \text{BW}$ sur des espaces de dimension de cet ordre de grandeur pour pouvoir en affirmer la supériorité face à AG∪BW car dans ce cas l'opérateur de croisement peut se révéler utile à l'exploration de l'espace de recherche. On peut envisager d'utiliser une variante muti-population de API pour répondre à cette difficulté (voir la section 7.8.1).

Une extension possible de $\text{API}_h \cup \text{BW}$ pourrait s'inspirer de l'algorithme GHOSP. Cet algorithme est identique à AG∪BW à la différence près que la population de CMC est composée de CMC ayant un nombre d'états pouvant être différent. Au départ la population est composée de représentants de chaque nombre d'états puis au cours des itérations, les CMC les plus adaptées sont sélectionnées ce qui fait qu'un certain nombre d'états peuvent ne plus être représentés dans la population. L'avantage de GHOSP sur AG∪BW est qu'il détermine automatiquement le nombre d'états cachés nécessaires à l'apprentissage. L'adaptation de cette technique à $\text{API}_h \cup \text{BW}$ est plus facile puisqu'il n'est pas utile de concevoir un opérateur génétique de croisement permettant de croiser deux CMC comportant un nombre d'états différents. Il suffit de définir des opérateurs $\mathcal{O}_{\text{rand}}$ et $\mathcal{O}_{\text{explo}}$ permettant, pour le premier de générer des CMC avec un nombre d'états variable et pour le deuxième d'ajouter ou de retirer des états à une CMC.

7.7.4 Apprentissage de Réseaux de Neurones Artificiels

Nous abordons ici le problème de l'approximation de fonctions : étant donné une fonction F dont les valeurs ne sont connues qu'en un nombre fini de k points $\{p_1, \cdots, p_k\}$, on veut construire un modèle qui approximera F en tout point. Les Réseaux de Neurones Artificiels (RNA) sont des modèles possibles pour l'approximation de fonctions (voir (Widrow, 1990) pour un exposé synthétique des RNA). Nous nous proposons de résoudre le problème de l'apprentissage des poids d'un réseau par l'algorithme API.

L'approximation de fonctions par RNA a été couplée au calcul évolutionnaire à la fois pour apprendre les poids du réseau (Yao, 1993) mais aussi sa topologie (Angeline et al., 1993; Mandischer, 1995). Ce type de problème a été abordé par exemple avec une stratégie d'évolution (SE) et en programmation génétique (PG)(Sebag et al., 1997). La SE a été utilisée pour l'approximation paramétrique (recherche des poids) et la PG pour l'approximation non paramétrique (recherche des poids et de la topologie). Si le modèle est bien choisi, la SE donne des résultats plus précis et plus rapidement que la PG qui a tendance à se perdre dans l'espace de recherche qui est beaucoup plus grand. Il ressort de cet article que dans les deux cas (SE et PG), les données expérimentales sont cruciales.

Le modèle que nous utilisons ici est le Perceptron Multi-Couches (*Multi Layer Perceptron* : MLP) (Rumelhart et al., 1986). La figure 7.20 présente un MLP à quatre entrées, une couche cachée et deux sorties.

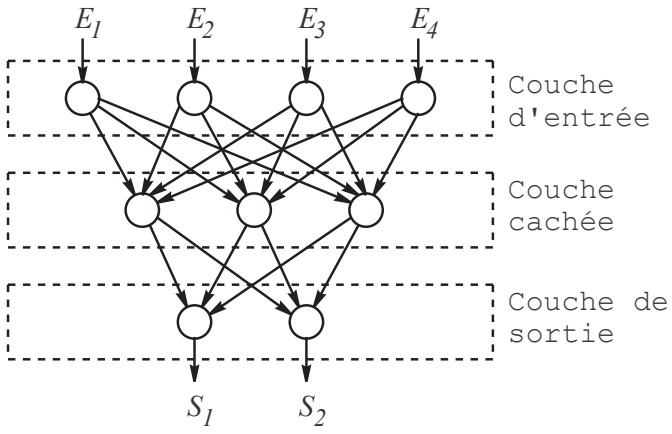


FIG. 7.20 – Perceptron multi-couche (MLP) à quatre entrées $(E_1, \ldots, E_4)$, une couche cachée et deux sorties (S_1, S_2).

La sortie de chaque cellule est calculée en fonction de ses entrées. Chaque arc possède un poids w_{ij}. La sortie d'une cellule est donnée par sa fonction de transfert. Nous avons utilisé la fonction sigmoïde :

$$sgm(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{7.20}$$

la valeur de x est donnée par les entrées de la cellule j :

$$x = \sum_{i \in E} w_{ij} y_i \tag{7.21}$$

où E est l'ensemble des cellules de la couches inférieure et y_i leur sortie.

Nous ne considérons pas le problème de la détermination de la topologie du réseau, les solutions seraient de toutes les façons assez proches de celles utilisées pour les chaînes de Markov cachées (cf 7.7.3).

API pour l'apprentissage du MLP

L'objectif est de découvrir les valeurs w_{ij} des poids du MLP afin de minimiser l'erreur quadratique E_q calculée sur l'ensemble de la base d'apprentissage :

$$E_q = \sum_{i=1}^{k} |F(p_i) - \hat{F}(p_i)| \tag{7.22}$$

où $\hat{F}$ correspond à l'approximation du MLP.

Du point de vue de API, la position s d'une fourmi correspond à un vecteur de $\mathbb{R}^W$ où W est le nombre d'arc du MLP. L'évaluation de cette position correspond à l'erreur quadratique obtenue par un MLP ayant comme poids les valeurs de s.

E_1	E_2	XOR(E_1,E_2)
$[-1,0[$	$[-1,0[$	$[-1,0[$
$[-1,0[$	$[0,1]$	$[0,1]$
$[0,1]$	$[-1,0[$	$[0,1]$
$[0,1]$	$[0,1]$	$[-1,0]$

TAB. 7.20 – Sorties de la fonction XOR.

k	N_c	RPG $\delta = 0.005$	RPG $\delta = 0.05$	RPG $\delta = 0.5$	API	API$_h$
10	3	15.99	16.05	25.08	16.60	▶ 15.79
10	5	17.01	17.95	27.84	15.41	▶ 14.55
10	7	20.79	19.90	37.73	17.45	▶ 15.80
50	3	9.94	12.48	41.99	11.73	▶ 9.18
50	5	7.14	▶ 6.65	48.87	11.13	6.81
50	7	▶ 6.62	8.63	49.60	8.05	6.79
100	3	▶ 7.38	8.07	37.7	10.04	7.50
100	5	▶ 4.16	6.09	49.65	7.29	5.26
100	7	▶ 3.58	4.60	62.75	6.30	4.61

TAB. 7.21 – Comparaison de API et de la RPG pour l'apprentissage de la fonction XOR. Les valeurs indiquées correspondent à la moyenne des pourcentages d'erreur quadratique. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque couple (k, N_c).

Expérimentations

Nous avons considéré un problème classique dans le domaine de l'approximation de fonctions : la fonction XOR. L'apprentissage de la fonction XOR (ou exclusif) consiste à découvrir les poids d'un MLP prenant en entrée deux valeurs réelles E_1 et E_2 toutes deux comprises dans $[-1, 1]$. La sortie est déterminée par le tableau 7.20. L'algorithme API a été dans un premier temps comparé à l'algorithme de Rétro-Propagation de Gradient (RPG) (Rumelhart et al., 1986). Cet algorithme nécessite de définir un paramètre de pas (δ), trois valeurs sont testées : 0.005, 0.05 et 0.5. Nous avons utilisé trois tailles (k) pour la base d'apprentissage : 10, 50 et 100. Nous avons aussi fait varier la structure du MLP : 3, 5 et 7 neurones ont été utilisés pour une couche cachée unique (N_c). Les résultats obtenus correspondent à l'erreur quadratique moyenne obtenue pour dix essais et sur une base de test de 100 valeurs. Pour chaque essai, API dispose de 10 000 évaluations de l'erreur quadratique. Les résultats sont présentés dans le tableau 7.21. Les paramètres de API sont les suivants : $n = 18$, $A_{\text{locale}} = 0.01$, $A_{\text{site}} = 0.1$, $P_{\text{locale}} = 10$, $p = 2$ et $P_N = 20$. Les poids du réseau sont dans l'intervalle $[-2, 2]$.

Les résultats obtenus semblent indiquer que API$_h$ est plus performant que la RPG pour des tailles d'apprentissage réduites. Cela laisse présager que API est intéressant

k	N_c	API∪RPG			API_h∪RPG		
		$\delta = 0.005$	$\delta = 0.05$	$\delta = 0.5$	$\delta = 0.005$	$\delta = 0.05$	$\delta = 0.5$
10	3	14.89	16.79	▶ 13.07	15.55	14.37	13.09
10	5	15.15	16.28	11.40	15.02	17.49	▶ 9.80
10	7	14.59	17.86	▶ 10.40	16.48	18.00	13.12
50	3	8.11	▶ 6.68	12.19	8.14	6.81	11.85
50	5	6.68	▶ 6.57	9.24	6.60	6.85	10.65
50	7	▶ 6.16	6.98	9.24	6.83	7.32	8.77
100	3	7.50	7.67	13.17	▶ 7.37	7.76	12.36
100	5	4.48	4.62	13.72	▶ 4.29	4.59	12.02
100	7	▶ 3.64	4.43	11.83	4.15	4.33	13.82

TAB. 7.22 – Résultats obtenus par l'hybridation de API et de la RPG pour l'apprentissage de la fonction XOR. Les valeurs indiquées correspondent à la moyenne des pourcentages d'erreur quadratique. Le symbole ▶ signale les meilleurs résultats obtenus pour chaque couple (k, N_c).

quand il est difficile d'obtenir des données expérimentales. De plus, plus le nombre de neurones cachés est important, plus les résultats de la RPG se dégradent alors que le phénomène est moins marqué pour API. Enfin, le pas δ de 0.005 pour la RPG est le plus performant.

Tout comme cela a été fait pour les chaînes de Markov cachées avec l'algorithme Baum-Welch, nous proposons d'hybrider API avec la RPG : API∪RPG. Le tableau 7.22 donne les résultats obtenus en suivant le même protocole de tests que précédemment.

L'hybridation donne des résultats qui s'opposent sur deux points aux constatations que nous avions formulées pour les résultats des méthodes séparées :

- la version hétérogène de API s'est montrée dans tous les cas plus performante que la version homogène (tableau 7.21). Quand on hybride API et la RPG cette tendance est inversée : API∪RPG donne plus souvent de meilleurs résultats que API_h∪RPG ;
- le pas de 0.005 est le plus intéressant quand la RPG opère seule. Quand API est utilisé pour initialiser la RPG cette conclusion n'est plus valable puisque les trois valeurs de pas testées ont toutes réussi à être performantes pour un couple de paramètres (k, N_c).

Les résultats obtenus sur l'apprentissage de réseaux de neurones nous ont montré que le succès de l'hybridation de API avec une heuristique issue du domaine n'est pas toujours évident, même dans le cas d'un problème simple. Les résultats présentés dans les tableaux 7.21 et 7.22 montrent que globalement les résultats sont meilleurs quand on hybride les méthodes. Le principal inconvénient est que l'on ne peut dégager de conclusion nette sur le paramètrage de l'hybridation (pas de la RPG et population homogène/hétérogène de API).

7.8 Discussion

7.8.1 Améliorations du modèle

Hybridation avec le recuit simulé

On peut envisager de changer la règle de décision d'une fourmi concernant la capture d'une proie. Classiquement, une proie est capturée si la fonction est améliorée par rapport au site exploré par la fourmi. L'idée est de considérer qu'une proie a été capturée si la fonction n'a pas été améliorée avec une certaine probabilité dépendant d'une variable de température. Le recuit simulé se comporte en effet d'une façon très similaire (cf chapitre 5).

Modèle multi-populations

La première étape d'API consiste à placer le nid à une position aléatoire (étape 2 de l'algorithme 7.1) qui sert de point de départ à la recherche des fourmis. L'inconvénient évident de cette opération est que les performances de l'algorithme vont être fortement dépendantes de cette initialisation. La parade classique en optimisation consiste à faire démarrer l'algorithme en différents points de l'espace de recherche. On initialise successivement le point de départ à des positions aléatoires ou permettant de quadriller l'espace. La première méthode permet alors de contourner la difficulté de la deuxième concernant la dimensionalité de l'espace mais ne garantit que statistiquement une bonne couverture de cet espace de recherche. La métaphore biologique la plus apte à représenter cette stratégie est d'utiliser plusieurs colonies de fourmis, chacune avec un nid à une position différente. Cette approche a par exemple été adoptée pour le PVC et l'agorithme AS (Kawamura et al., 2000). Si on exclut toute interaction entre les colonies (coopération ou compétition), cela revient à lancer séquentiellement plusieurs fois API en modifiant l'emplacement initial du nid. Cette variante de API, que l'on pourrait noter API_m, est donc une forme de parallélisation de la recherche (nous en présentons plus loin (section 7.8.3) une implémentation parallélisant la recherche de chaque fourmi). Le principal avantage est que l'on parallélise la recherche très simplement, l'inconvénient est que l'on donne moins d'explorations à une colonie si on impose un nombre fixe d'évaluations de la fonction. Cette technique s'apparente à l'utilisation de plusieurs populations dans les algorithmes génétiques (*Island Model*, voir à ce propos une discussion dans (Whitley et al., 1997)). La coopération entre les populations est alors formalisée par des migrations entre populations. La coopération que l'on peut introduire dans API pourait prendre la forme suivante : régulièrement, des fourmis, et donc leurs sites de chasse, peuvent être échangées entre deux colonies. La compétition peut aussi permettre de rendre API_m adaptatif : on peut introduire la notion de viabilité pour chaque colonie : chaque fourmi dispose d'une espérance de vie et chaque naissance d'une fourmi dépend d'une probabilité fixée par rapport à la performance du nid.

Hétérogénéité des paramètres avancée

L'hétérogénité de API_h peut être poussée plus loin : les amplitudes A_{site} et A_{locale} peuvent prendre des valeurs différentes suivant la coordonnée du vecteur à laquelle elle s'applique. Ce genre de disparité se retrouve par exemple dans les stratégies d'évolution adaptatives où le taux de mutation est modifié au cours de l'évolution de l'algorithme et de façon indépendante pour chaque dimension de l'espace de recherche (voir par exemple (Hansen et al., 1995)).

7.8.2 Autres problèmes

Nous n'avons pas présenté d'adaptation de API aux fonctions binaires. Nous pourrions par exemple nous inspirer des sociétés de Hill-Climbers de Sebag et Schoenauer (Sebag and Schoenauer, 1997). La méthode qu'ils proposent consiste à manipuler une population de solutions dont le déplacement s'apparente à une mutation qui tient compte des individus inadaptés précédemment rencontrés. Cet historique est stocké sous la forme d'un vecteur réel appelé *repoussoir* qui est mis à jour de la même façon que dans PBIL (cf chapitre 6). Les bits proches en valeur de la composante correpondante du repoussoir ont une probabilité d'être mutés plus forte. Pour API, le nombre de bits à muter pourrait représenter l'amplitude de recherche et donc être variable suivant les fourmis d'une population hétérogène[11].

7.8.3 Parallélisation de API

Nous avons présenté à plusieurs reprises API comme une heuristique facilement parallélisable du point de vue de son implémentation. Nous présentons ici une expérience qui a été faite dans ce sens.

Si on ne considère pas le recrutement et le déplacement périodique du nid, chaque fourmi effectue sa recherche de façon totalement indépendante des autres. Cette caractéristique peut être considérée comme une faiblesse par rapport aux heuristiques telles que les algorithmes génétiques qui sont pour une part basés sur la recombinaison des caractéritiques des solutions qu'ils manipulent. Cependant cette faiblesse est discutable du point de vue de la qualité de l'échantillonage de l'espace de recherche (voir la section 7.6) et surtout de la facilité à la parallélisation. La parallélisation, pour les algorithmes évolutionnaires, consiste par exemple à manipuler plusieurs sous-populations et de limiter les interactions à des mécanismes d'immigration. Nous avons envisagé ce type de modèle pour API (API_m, section 7.8.1). La parallélisation que nous proposons ici se situe à un niveau plus bas : nous avons parallélisé le déplacement de chaque fourmi.

La mise en œuvre de cette parallélisation associe une fourmi à un processeur tout en conservant un processeur dédié à la synchronisation des fourmis. Ce choix peut se représenter sous la forme d'un modèle maître-esclaves où le maître correspond au nid et

[11] C'est justement ce paramètre qui est considéré comme critique dans (Sebag and Schoenauer, 1997).

les esclaves aux fourmis. La figure 7.21 présente le déroulement de API avec 2 fourmis sur trois processeurs. Les différentes phases et messages echangés sont les suivants :

- La section a du processeur maître est une phase d'initialisation ;
- la section b des processeurs esclaves correspond aux explorations indépendantes des fourmis ;
- le message 1 permet l'initialisation d'une fourmi (c'est-à-dire le positionnement sur le nid et l'oubli de tous ses sites de chasse antérieurs) ;
- le message 2 est la réponse de la fourmi au bout de P_N sorties du nid (c'est-à-dire la meilleure évaluation de la fonction qu'elle a trouvée jusqu'ici) ;
- la section c du maître correspond au déplacement du nid sur la meilleure position renvoyée par les fourmis.

Les temps de communication peuvent être variables (comme ils le sont représentés) à cause des variations éventuelles du traffic en communication entre chaque processeur.

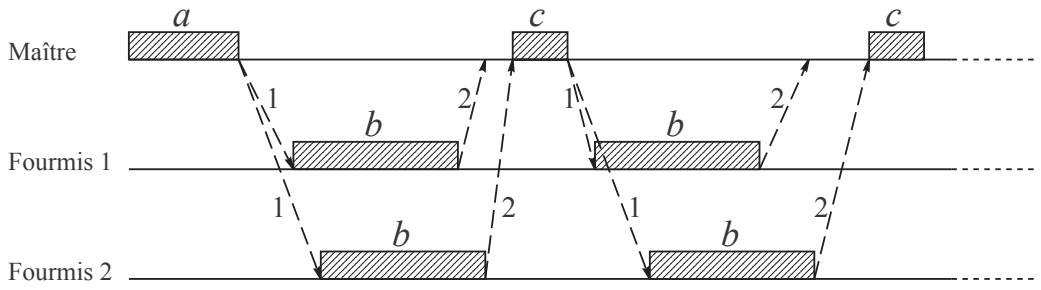


FIG. 7.21 – Parallélisation de API : déroulement sur trois processeurs.

L'implémentation de ce modèle a été faite sur un réseau local avec la bibliothèque PVM (*Parallel Virtual Machine*) (Dongarra et al., 1995) permettant une grande souplesse de gestion des processeurs et des messages.

Les premiers résultats que nous avons pu obtenir ont montré que la version parallèle améliore la vitesse de API en particulier quand le nombre d'évaluations de la fonction (paramètre T_3) est élevé. Nous avons aussi constaté que la vitesse de la version parallèle augmente en fonction du nombre de fourmis et obtient un maximum pour 10 fourmis. Ensuite, sûrement à cause des temps de communication, la durée augmente pour 15 fourmis. Pour 10 fourmis la version parallèle est environ 4 fois plus rapide que la version séquentielle pour la fonction F_{10}.

7.9 Conclusion

Dans ce chapitre nous avons exposé la modélisation des fourmis de l'espèce *Pachycondyla apicalis* en l'appliquant à un certain nombre de problèmes d'optimisation. Au delà de la stimulation que représente l'étude du comportement des fourmis, nous avons montré que la source d'inspiration biologique, aussi originale soit elle, pouvait

nous conduire à une heuristique (l'algorithme API) proche, par certains de ses aspects, des heuristiques existantes. L'étude expérimentale nous a montré que les points de différences sont sûrement influents pour les performances. Un des atouts de cette modélisation est que sa description est relativement indépendante de l'espace de recherche considéré, ce qui permet facilement de l'appliquer à de nouveaux problèmes. De plus, les résultats obtenus par API sont souvent compétitifs avec ceux obtenus par des métaheuristiques plus classiques. Les principales évolutions de API que l'on peut espérer sont celles qui préoccupent depuis quelque temps déjà les méta-heuritiques plus installées. Par exemple, il sera intéressant d'adapter API au problème d'optimisation avec contraintes, à l'optimisation multicritère, à l'optimisation dynamique ...

L'algorithme API a donné lieu à l'encadrement de plusieurs projets de fin d'études à l'E3i (Meslet, 1998; Laurin, 1998; Durand, 2000) et à l'encadrement d'un stage de DESS (Desbarats, 1998). Un certain nombre de communications ou publications ont aussi ponctué les différentes étapes de cette étude (Venturini et al., 1997; Monmarché et al., 1997; Monmarché et al., 1998; Monmarché et al., 1998; Monmarché et al., 1999c; Monmarché et al., 2000b).

Conclusion

Ce qu'il faut retenir

Dans ce travail de thèse, nous présentons les travaux s'inspirant du comportement des fourmis réelles pour la résolution de problèmes en informatique. Les fourmis possèdent en effet de nombreuses caractéristiques collectives et individuelles pouvant nous aider à la résolution de problèmes complexes (chapitre 1). Les premiers travaux apparus dans ce domaine se sont inspirés de l'intelligence collective des fourmis pour la résolution de problèmes combinatoires comme le problème du voyageur de commerce ou encore de problèmes de robotique comme l'exploration ou l'exploitation d'environnements hostiles à l'homme (chapitre 2). Nous proposons deux approches supplémentaires de cette nouvelle source d'inspiration biomimétique.

La capacité des fourmis à rassembler et à trier des objets est à l'origine d'un algorithme pour la classification non supervisée. Un ensemble d'objets, définis par un certain nombre de caractéristiques numériques, doit être partitionné de la manière la plus naturelle possible et la plus pertinente aux yeux d'un expert du domaine. Nous ne disposons d'aucune information sur le nombre de groupes à constituer. Les fourmis artificielles ont été utilisées pour proposer une solution à ce problème (Lumer and Faieta, 1994) : les objets sont répartis sur une grille à deux dimensions indépendemment de leurs caractéristiques et les agents-fourmis ramassent et déposent ces objets suivant une mesure de similarité entre les objets de cases voisines. Le comportement local d'une fourmi (la décision de ramasser ou de déposer un objet) amène la grille vers un état stable où les objets proches par leurs caractéristiques le sont aussi sur la grille et forment ainsi des groupes. Nous avons repris ces travaux et nous les avons étendus dans plusieurs directions. La grille utilisée jusqu'alors ne pouvait accueillir qu'un objet par case, en autorisant les fourmis et les cases de la grille à recevoir plusieurs objets, nous avons simplifié et accéléré la convergence. L'algorithme AntClass, développé à cette occasion, est hybride dans le sens où la recherche du nombre de classes est effectuée par les fourmis artificielles et qu'un algorithme classique en classification, les centres mobiles, est utilisé pour gommer les erreurs de classification inhérentes à une méthode stochastique telle que les fourmis artificielles. Par rapport aux travaux initiaux, nous obtenons de bons résultats avec beaucoup moins d'itérations.

L'optimisation est un problème général qui connaît actuellement de nombreux développements issus de modélisation biomimétiques variées (Corne et al., 1999). Dans un premier temps nous explorons les approches BSC, PBIL et ACO pour l'optimisation de fonctions binaires et nous proposons un modèle commun qui a été évalué sur

plusieurs problèmes (chapitre 6). Ensuite, la deuxième partie principale de ce travail de thèse s'inspire directement de la stratégie de recherche de nourriture d'une espèce de fourmis (*Pachycondyla apicalis*) (Fresneau, 1994) pour résoudre des problèmes d'optimisation globale. En plus de la nouveauté de l'approche, l'apport de cette adaptation réside principalement dans sa simplicité. Les fourmis de cette espèce possèdent en effet la caractéristique de chasser en solitaire et d'utiliser un nombre réduit de règles comportementales. À titre d'exemples nous appliquons l'algorithme qui en découle, appelé API, à des problèmes variés tels que l'optimisation de fonctions numériques, l'apprentissage de chaînes de Markov cachées ou des poids d'un réseau de neurones artificiels, ou encore à un problème d'optimisation combinatoire classique : le problème du voyageur de commerce. Nous explorons plusieurs directions, soit en s'inspirant des caractéristiques naturelles des fourmis, soit en ajoutant à API des étapes de recherches locales liées au problème traité. L'algorithme API, bien que totalement inspiré des fourmis *Pachycondyla apicalis*, peut être comparé à plusieurs techniques d'optimisation tels que les algorithmes génétiques (Michalewicz, 1996). L'intérêt de cette modélisation est qu'elle reste proche de sa source d'inspiration tout en donnant des résultats compétitifs par rapport à d'autres techniques d'optimisation biomimétiques.

Ce travail de thèse se positionne comme une contribution au développement d'algorithmes inspirés des fourmis. Cette contribution prend deux formes : dans un premier temps, nous améliorons un algorithme existant en y introduisant à la fois des innovations inspirées des fourmis ainsi qu'une hybridation avec un algorithme issu du domaine de la classification. Dans un deuxième temps, nous proposons une modélisation nouvelle du comportement d'une espèce de fourmi spécifique pour le problème général de l'optimisation. Deux objectifs ont été suivis : améliorer l'existant et proposer une nouvelle source d'inspiration.

Perspectives

Les perspectives sont nombreuses et ont été pour la plupart évoquées à la fin de chaque chapitre. Au sujet de la classification non supervisée (chapitres 3 et 4), il paraît judicieux de considérer comme limitée l'utilisation de données uniquement numériques. L'adaptation des principes développés dans AntClass à des données symboliques, comme peuvent l'être des données textuelles, trouvera certainement de nombreuses applications dans le domaine de la découverte de connaissances et la recherche d'informations sur Internet. La nature éminemment distribuée d'Internet et des réseaux de stockage d'informations en général, serait en effet toute promise à des méthodes de recherche et de classification inspirées de l'activité des fourmis, par essence distribuées.

La difficulté des méthodes exactes et classiques d'optimisation pour résoudre des problèmes industriels sera dans le futur à l'origine d'innovations les plus diverses et originales (chapitre 5). Les modèles basés sur une population de solutions comme les algorithmes génétiques et centrés sur l'apprentissage d'un modèle probabiliste (chapitre 6) sont par exemple en plein développement. Il y a essentiellement deux voies pour l'innovation dans ce domaine :

– rechercher de nouvelles sources d'inspiration dans la nature. C'est que nous avons

tenté avec les fourmis de l'espèce *Pachycondyla apicalis* ;
- associer et hybrider différentes méthodes d'optimisation entre elles.

La première voie comporte ses dangers, on risque constamment de réinventer ce qui existe déjà sous une autre forme. Nous avons ainsi fait remarquer que l'algorithme API possédait un certain nombre d'analogies avec des techniques existantes. Cependant il reste deux aspects propres à API : les différentes propriétés qui le caractérisent n'ont, à notre connaissance, jamais été rassemblées dans une même méthode et ce rassemblement existe « à l'état naturel ». Les sources d'inspiration que peuvent nous fournir les fourmis ne sont pas taries. Par exemple, la capacité d'une fourmis à différencier les individus de sa colonie par rapport aux individus des colonies voisines nécessite une forme d'apprentissage probablement évolué et sûrement assez proche des capacités du système immunitaire. Ce genre de caractéristiques pourrait s'apparenter à la résolution d'un problème de classification non supervisée : chaque individu résoud un problème de discrimination. La deuxième voie, à savoir l'hybridation des méthodes entre elles, a aussi largement été explorée pour différents types de problèmes. Utiliser les techniques existantes et adaptées à un problème est certainement une voie prometteuse. Cela positionne les méthodes biomimétiques en tant que métaheuristiques d'optimisation. Un travail en cours avec Vincent T'Kindt, Daniel Laugt et Fabrice Tercinet sur l'optimisation bicritère d'un problème de flowshop à deux machines par un algorithme de type ACO en est une illustration et donne des résultats très prometteurs (T'Kindt et al., 2000; Laügt et al., 2000). Le développement des algorithmes de type ACO a atteint en une dizaine d'années un stade de maturation où ces principes sont maintenant appliqués à de nombreux problèmes (chapitre 2). La tendance sera donc à la fois de diversifier les problèmes traités comme de rassembler les heuristiques proches pour construire des métaheuristiques.

Épilogue

La Fourmi

Une fourmi de dix-huit mètres
Avec un chapeau sur la tête,
Ça n'existe pas, ça n'existe pas.
Une fourmi trainant un char
Plein de pingouins et de canards,
Ça n'existe pas, ça n'existe pas.
Une fourmi parlant français,
Parlant latin et javanais,
Ça n'existe pas, ça n'existe pas.
Eh ! Pourquoi pas ?

Robert Desnos
Extrait de *Chantefables et Chantefleurs*, Gründ.

Remerciements

Bien que cela ne soit pas dans les habitudes, je tiens à remercier le lecteur patient qui aurait atteint, ne serait-ce que par le hasard, cette page. Cela me laisse l'occasion de lui signifier que ce document n'est que la partie émergée (émergente ?) d'un travail de thèse. Cette page n'aurait en effet pas vu le jour sans une succession d'étapes avant tout humaines.

C'est ainsi qu'il convient d'attribuer une partie de ce travail à Christian Proust, grâce à qui le « montage financier » a été possible.

L'enthousiasme scientifique de mes encadrants, Gilles Venturini et Mohamed Slimane, au delà de l'amitié que je leur témoigne, doit aussi être considéré comme le point de départ de cette réalisation. Voilà pour le commencement.

Par la suite, je mentirais en remerciant des personnes pour leur aide dans, ce que l'on a coutume de désigner, les moments difficiles de la thèse puisque c'est précisément grâce à de nombreuses personnes qu'il y a eu peu de moments difficiles.

Tout d'abord, je voudrais mentionner les biologistes, spécialistes des fourmis, pour la source d'inspiration qu'ils véhiculent. Être la source d'innovations et d'inventions scientifiques par la passion qui les anime est pour moi une des premières qualités scientifiques. Je souhaite donc à Alain Lenoir, Annie et Guy Leroux, Guy Beugnon, Bertrand Schatz et Dominique Fresneau de continuer à observer les fourmis le plus longtemps possible.

D'un point de vue informatique, c'est toute l'E3i qu'il faut citer, tant par les nombreux projets menés par les étudiants (notamment Sébastien Laurin, Olivier Meslet, David Steinberg, Laurent Françoise, Guillaume Dromel, Damien Duvaux, Daniel Laügt et Ludovic Mathé par leur travail de projet de fin d'études) que par la qualité de l'ambiance qui y règne du point de vue pédagogique et scientifique (Ameur, André, Christian, Christine, Christophe, Claudine, Colette, Emmanuel, Françoise, Gilles, Jean-Charles, Jean-Louis, Jean-Louis, Jean-Pierre, Lionel, Manoochere, Manu, Marie-Do, Michel, Mikaël, Mohand, Nathalie, Nicolas, Nicole, PMak, Patrick, Pierrot, Romual, Sébastien, Sylvain, Thierry, Thierry, Vincent, ... [12]).

Enfin, je dois bien avouer que si l'orthographe n'a pas été trop malmenée, celà revient à ma mère.

La dernière étape de ce travail me permet de citer et de remercier Jin-Kao Hao et Philippe Preux pour leur travail de rapporteur, ainsi que Paul Bourgine, Jean-Louis Deneubourg et Marc Schoenauer pour leur rôle d'examinateur. J'aimerais qu'ils mesurent ici l'importance de leur contribution dans cette réalisation collective.

[12] Aux regards de l'Histoire, toute cette assemblée pourra témoigner qu'en informatique, il n'y a pas que des puces : il y a aussi des fourmis.

Bibliographie

Abbatista, F., Abbatista, N., and Caponetti, L. (1995). « An Evolutionary an Cooperative Agents Model for Optimization ». In (IEEE, 1995).

Abbattista, F. and Dalbis, D. (1996). « Improving the Genetic Algorithms by means of a Cooperative Model ». In *Proceedings of the Second Online Workshop on Evolutionary Computation(WEC2)*, pages 61–64.

Alexandrov, D. (2000). « Randomized Algorithms for the Minmax Diameter k-Clustering Problem ». In *Proceedings of ECCO 13*, pages 193–194, Capri, Italy.

Angeline, P., Saunders, G., and Pollack, J. (1993). « An evolutionary algorithm that constructs recurrent neural networks ». *IEEE Transactions on Neural Networks*, 5(2) :86–91.

Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors (1999). *Congress on Evolutionary Computation*. IEEE Press.

Arkin, R. (1998). *Behavior-based robotics*. MIT Press, Cambridge, Massachusetts.

Arkin, R. and Balch, T. (1998). « *Artificial Intelligence and Mobile Robots* », Chapitre Cooperative Multiagent Robotic Systems, pages 277–296. MIT Press, Cambridge, Massachusets.

Assad, A. and Packard, H. (1991). « Emergent Colonization in an Artificial Ecologie ». In (Varela and Bourgine, 1991), pages 143–152.

Bäck, T., Hoffmeister, F., and Schwefel, P. (1991). « A Survey of Evolution Strategies ». In (Belew and Booker, 1991), pages 2–9.

Bagnères, A., Rivière, G., and Clément, J. (1998). « Artificial neural network modeling of caste odor discrimination based on cuticular hydrocarbons in termites ». *Chemoecology*, 8 :201–209.

Ball, G. and Hall, D. (1965). « ISODATA, a novel method of data analysis and pattern classification ». Technical Report, Stanford Research Institute.

Ball, G., Hall, D., Wolf, D., and Eusebio, J. (1969). « Promenade an Improved Interctive-Graphics Man/Machine System for pattern Recognition ». Technical Report, Stanford Research Institute.

Baluja, S. (1994). « Population-Based Incremental Learning : A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning ». Technical Report CMU-CS-94-163, Carnegie Mellon University.

Baluja, S. (1995). « An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics ». Technical Report CMU-CS-95-193, Carnegie Mellon University.

Baluja, S. and Caruana, R. (1995). « Removing the Genetics from the Standard Genetic Algorithm ». In (Prieditis and Russell, 1995), pages 38–46.

Banzhaf, W., Daida, J., Eiben, A., Garzon, M., Honavar, V., Jakiela, M., and Smith, R., editors (1999). *Genetic and Evolutionary Computation Conference (GECCO-99)*, Orlando, Florida USA. Morgan Kaufmann, San Francisco, CA.

Bauer, A., Bullnheimer, B., Hartl, R., and Strauss, C. (1999). « An Ant Colony Approach for the Single Machine Total Tardiness Problem ». POM Working Paper 5/99, Department of Management Science, University of Vienna.

Beckers, R., Holland, O., and Deneubourg, J. (1994). « From local actions to global tasks : stigmergy and collective robotics ». In *Proceedings of Artificial Life 4*, pages 181–189. MIT Press.

Belew, R. and Booker, L., editors (1991). *Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA.

Bilchev, G. and Parmee, I. (1995). The Ant Colony Metaphor for Searching Continuons Design Spaces. In Fogarty, T., editor, *Lecture Notes in Computer Science*, volume 993, pages 24–39. Springer Verlag.

Bilchev, G. and Parmee, I. (1996a). « Constrained Optimisation with an Ant Colony Search Model ». In *Second Conference on Adaptive Computing in Engineering Design and Control*, Plymouth, UK.

Bilchev, G. and Parmee, I. (1996b). « Evolutionary Metaphors for the Bin Packing Problem ». In *Fifth Annual Conference on Evolutionary Programming*, San Diego, California, USA.

Blake, C. and Merz, C. (1998). « UCI Repository of machine learning databases, University of California, Irvine, Dept. of Information and Computer Sciences ». `http ://www.ics.uci.edu/~mlearn/MLRepository.html`.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence : From Natural to Artificial Systems*. Oxford University Press, New York.

Bonabeau, E., Henaux, F., Guerin, S., Snyers, D., Kuntz, P., and Theraulaz, G. (1998a). « Routing in Telecommunications Networks with "Smart" Ant-Like Agents ». Working Paper 98-01-003, Santa Fe Institute.

Bonabeau, E., Sobkowski, A., Theraulaz, G., and Deneubourg, J. (1998b). « Adaptive Task Allocation Inspired by a Model of Division of Labor in Social Insects ». Working Paper 98-01-004, Santa Fe Institute.

Bonabeau, E. and Theraulaz, G. (1994). *Intelligence Collective*. Hermes.

Bonabeau, E., Theraulaz, G., and Deneubourg, J. (1998c). « Fixed Response Thresholds and the Regulation of Division of Labor in Insect Societies ». Working Paper 98-01-009, Santa Fe Institute.

Botee, H. and Bonabeau, E. (1999). « Evolving Ant Colony Optimization ». Working Paper 99-01-009, Santa Fe Institute.

Bow, S. (1984). *Pattern Recognition, Application to Large Data-Set Problems*. Marcel Dekker inc, New York.

Brooks, R. (1986). « A robot layered control system for a mobile robot ». *IEEE Journal of robotics and automation*, 2 :14–23.

Brossut, R. (1996). *Phéromones, la communication chimique chez les animaux*. CNRS editions, Belin.

Brouard, T. (1999). « *Algorithmes hybrides d'apprentissage de chaînes de Markov cachées : conception et application à la reconnaissance des formes* ». PhD thesis, Université de Tours.

Bullnheimer, B. (1999). « *Ant Colony Optimization in Vehicle Routing* ». Doctoral thesis, University of Vienna.

Bullnheimer, B., Hartl, R., and Strauss, C. (1997a). « Applying the Ant System to the Vehicle Routing Problem ». In *Second International Conference on Metaheuristcs*, Sophia-Antipolis, France.

Bullnheimer, B., Hartl, R., and Strauss, C. (1997b). « A New Rank Based Version of the Ant System - A Computational Study ». Working Paper 3/97, Institute of Management Science, University of Vienna, Austria.

Bullnheimer, B., Hartl, R., and Strauss, C. (1999a). Applying the Ant System to the Vehicle Routing Problem. In Voss, S., Martello, S., Osman, I., and Roucairol, C., editors, *Meta-Heuristics : Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Boston.

Bullnheimer, B., Hartl, R., and Strauss, C. (1999b). An Improved Ant system Algorithm for the Vehicle Routing Problem. In Dawid, Feichtinger, and Hartl, editors, *Annals of Operations Research*, Nonlinear Economic Dynamics and Control.

Bullnheimer, B., Hartl, R., and Strauss, C. (1999c). « A New Rank Based Version of the Ant System : A Computational Study ». *Central European Journal for Operations Research and Economics*, 7(1) :25–38.

Bullnheimer, B., Kotsis, G., and Strauss, C. (1997c). Parallelization Strategies for the Ant System. In De Leone, R., Murli, A., Pardalos, P., and Toraldo, G., editors, *High Performance Algorithms and Software in Nonlinear Optimization*, volume 24 of *Applied Optimization*, pages 87–100. Kluwer, Dordrecht.

Byrne, R. (1997). « *Global Optimisation in Process Design* ». PhD thesis, University of London.

Chagné, P., Beugnon, G., and Dejean, A. (1999). « Fourragement chez *Gigantiops destructor* (Fabricius) (Formicidae : Formicinae) ». In *Actes des Colloques Insectes Sociaux*, volume 13, pages 21–26, Tours, France.

Chapuisat, M. and Keller, L. (1997). « Les fourmis sont-elles encore en froid avec Darwin ? ». *La Recherche*, 196 :90–93.

Cliff, D., Husbands, P., Meyer, J., and W., S., editors (1994). *Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3*. MIT Press, Cambridge, Massachusetts.

Collins, R. and Jefferson, D. (1992). « AntFarm : Towards Simulated Evolution ». In (Langton et al., 1992), pages 579–601.

Colorni, A., Dorigo, M., and Maniezzo, V. (1991). « Distributed Optimization by Ant Colonies ». In (Varela and Bourgine, 1991), pages 134–142.

Colorni, A., Dorigo, M., and Maniezzo, V. (1992). « An investigation of some properties of an "Ant algorithm" ». In (Männer and Manderick, 1992), pages 509–520.

Colorni, A., Dorigo, M., and Maniezzo, V. (1995). « New Results of an Ant System Approach Applied to the Asymmetric TSP ». In I.H.Osman and Kelly, J., editors, *Metaheuristics International Conference*, pages 356–360, Hilton Breckenridge, Colorado. Kluwer Academic Publishers.

Colorni, A., Dorigo, M., Maniezzo, V., and Trubian, M. (1994). « Ant System for Job-shop Scheduling ». *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1) :39–53.

Cordón, O., Herrera, F., and Moreno, L. (1999). « Integración de conceptos de computatión evolutiva en un nuevo modelo de colonias de hormigas ».

Corne, D., Dorigo, M., and Glover, F., editors (1999). *New Ideas in Optimisation*. McGraw-Hill, London, UK.

Costa, D. and Hertz, A. (1997). « Ants Can Colour Graphs ». *Journal of the Operational Research Society*, 48 :295–305.

Cucchiara, R. (1993). « Analysis and comparison of different genetic models for the clustering problem in image analysis ». In Albrecht, R., Reeves, C., and Steele, N., editors, *International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 423–427. Springer-Verlag.

Culioli, J. (1994). *Introduction à l'optimisation*. Ellipse, seconde edition.

Darwin, C. (1859). *On the Origin of Species*. John Murray, London.

De Jong, K. (1975). « *An analysis of the behavior of a class of genetic adaptive systems* ». PhD thesis, University of Michigan.

den Besten, M., Stützle, T., and Dorigo, M. (2000). « Ant Colony Optimization for the Total Weighted Tardiness Problem ». In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature*, volume 1917 of *Lecture Notes in Computer Science*, pages 611–?? Springer Verlag.

Deneubourg, J. (1977). « Application de l'ordre par fluctations à la description de certaines étapes de la construction du nid chez les termites ». *Insectes Sociaux*, 24 :117–130.

Deneubourg, J., Goss, S., Pasteels, J., Fresneau, D., and Lachaud, J. (1987). Self-organization mechanisms in ant societies (II) : learning in foraging and division of labor. In Pasteels, J. and Deneubourg, J., editors, *From individual to collective behavior in social insects, Experientia supplementum*, volume 54, pages 177–196. Bikhauser Verlag.

Deneubourg, J.-L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C., and Chretien, L. (1990). « The dynamics of collective sorting : robot-like ant and ant-like robots ». In (Meyer and Wilson, 1990), pages 356–365.

Desbarats, L. (1998). « Apprentissage d'un réseau de Neurones par une Population de Fourmis *Pachycondyla apicalis* pour l'approximation de fonctions ». Rapport de stage de DESS, École d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

Di Caro, G. and Dorigo, M. (1997). « AntNet : A Mobile Agents Approach to Adaptive Routing ». Technical Report 97-12, IRIDIA, Université Libre de Bruxelles, Belgium.

Di Caro, G. and Dorigo, M. (1998). « Ant Colonies for Adaptive Routing in Packet-switched Communications Networks ». In (Eiben et al., 1998a).

Di Caro, G. and Dorigo, M. (1998a). « AntNet : Distributed Stigmergetic Control for Communications Networks ». *Journal of Artificial Intelligence Research*, 9 :317–365.

Di Caro, G. and Dorigo, M. (1998b). « Mobile Agents for Adaptive Routing ». In *31st Hawaii International Conference on System*, Big Island of Hawaii, January 6-9.

Di Caro, G. and Dorigo, M. (1998). Two Ant Colony Algorithms for Best-Effort Routing in Datagram Networks. In *Proceedings of PDCS'98 - 10th International Conference on Parallel and Distributed Computing and Systems, Las Vegas, Nevada, October 28-31.*

Diday, E. and Simon, J. (1976). Cluster analysis. In Fu, K., editor, *Digital Pattern Recognition*, pages 47–94. Springer-Verlag, Berlin.

Dongarra, J., Gengler, M., Tourancheau, B., and Vigouroux, X. (1995). *EuroPVM'95*. Hermes.

Dorigo, M. (1992). « *Optimization, Learning and Natural Algorithms (in Italian)* ». PhD thesis, Politecnico di Milano, Italy.

Dorigo, M., editor (1998). *ANTS'98 - From Ant Colonies to Artificial Ants : First International Workshop on Ant Colony Optimization, Brussels, Belgium, October 15-16*. `http ://iridia.ulb.ac.be/ants/ants98/ants98.html`.

Dorigo, M. and Di Caro, G. (1999a). « Ant Colony Optimization : A New Meta-Heuristic ». In (Angeline et al., 1999).

Dorigo, M. and Di Caro, G. (1999b). The Ant Colony Optimization Meta-Heuristic. In (Corne et al., 1999), pages 11–32.

Dorigo, M., Di Caro, G., and Gambardella, L. (1999). « Ant Algorithms for Discrete Optimization ». *Artificial Life*, 5(2).

Dorigo, M. and Gambardella, L. (1996). « A Study of some properties of Ant-Q ». In (Voigt et al., 1996), pages 656–665.

Dorigo, M. and Gambardella, L. (1997a). « Ant Colonies for the Traveling Salesman Problem ». *BioSystems*, 43 :73–81.

Dorigo, M. and Gambardella, L. (1997b). « Ant Colony Sytem : A Cooperative Learning Approach to the Travelling Salesman Problem ». *IEEE Transactions on Evolutionary Computation*, 1(1) :53–66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1991). « Positive feedback as a search strategy ». Technical Report 91-016, Politecnico di Milano, Italy.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). « The Ant System : Optimization by a Colony of Cooperating Agents ». *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1) :29–41.

Dorigo, M., Middendoff, M., and Stützle, T., editors (2000). *ANTS'2000 - From Ant Colonies to Artificial Ants : Second International Workshop on Ant Algorithms, Brussels, Belgium, September 8-9.* `http ://iridia.ulb.ac.be/~ants/ants2000/index.html`.

Drogoul, A., Ferber, J., Corbara, B., and Fresneau, D. (1991). « A Behavioral Simulation Model for the study of Emergent Social Structures ». In (Varela and Bourgine, 1991), pages 161–170.

Dromel, G. (1999). « Comparaison de systèmes d'optimisation à base de population ». Rapport de projet de fin d'études, École d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

Durand, G. (2000). « Parallélisation de l'algorithme API ». Rapport de projet de fin d'études, École d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

Duvaux, D. (2000). « Robot-fourmi ». Rapport de projet de fin d'études, École d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

Eiben, A., Bäck, T., Schwefel, H.-P., and Schoenauer, M., editors (1998a). *Fifth International Conference on Parallel Problem Solving from Nature (PPSN V).* Springer-Verlag, New York.

Eiben, A., Hinterding, R., and Michalewicz, Z. (1998b). « Parameter Control in Evolutionary Algorithms ». Technical Report TR98-07, Department of Computer Science of the Leiden University.

Epstein, J. and Axtell, R. (1996). *Growing Artificial Societies.* MIT Press.

Eshelman, L., editor (1995). *Sixth International Conference on Genetic Algorithms.* Morgan Kaufmann, San Francisco, CA.

Falkenauer, E. (1994). « A new representation and operators for genetic algorithms applied to grouping problems ». *Evolutionary Computation*, 2(2) :123–144.

Ferber, J. (1995). *Les Systèmes multi-agents, vers une intelligence collective.* Inter-Edition.

Floreano, D., Nicoud, J., and Mondana, F., editors (1999). *Fifth European Conference on Artificial Life, Lausanne, Switzerland.* Springer-Verlag, Heildelberg, Germany.

Fogel, L., Owens, A., and Walsh, M. (1966). *Artificial Intelligence Through Simulated Evolution.* Wiley, J., Chichester, UK.

Forsyth, P. and Wren, A. (1997). « An Ant System for Bus Driver Scheduling ». In *7th International Workshop on Computer-Aided Scheduling of Public Transport*, Boston.

Françoise, L. (1999). « Robotique mobile PC/104 ». Rapport de projet de fin d'études, École d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

Françoise, L., Monmarché, N., and Venturini, G. (1999). « Vers un robot modélisant la perception visuelle des fourmis ». In *Actes des Colloques Insectes Sociaux*, volume 13, pages 169–172, Tours, France.

Franks, N. and Sendova-Franks, A. (1992). « Brood sorting by ants : distributing the workload over the work surface ». *Behav. Ecol. Sociobiol.*, 30 :109–123.

Fresneau, D. (1985). « Individual foraging and path fidelity in a ponerine ant ». *Insectes Sociaux, Paris*, 32(2) :109–116.

Fresneau, D. (1994). « *Biologie et comportement social d'une fourmi ponérine néotropicale (Pachycondyla apicalis)* ». Thèse d'etat, Université de Paris XIII, Laboratoire d'Ethologie Expérimentale et Comparée, France.

Gambardella, L. and Dorigo, M. (1995). « Ant-Q : A Reinforcement Learning Approach to the Travelling Salesman Problem ». In (Prieditis and Russell, 1995), pages 252–260.

Gambardella, L. and Dorigo, M. (1996). « Solving Symmetric and Asymmetric TSPs by Ant Colonies ». In IEEE, editor, *Proceedings of the third IEEE International Conference on Evolutionary Computation*, pages 622–627, Nagoya, Japan. IEEE Press.

Gambardella, L. and Dorigo, M. (1997). « HAS-SOP : An Hybrid Ant System for the Sequential Ordering Problem ». Technical Report 97-11, IDSIA, Lugano, Switzerland.

Gambardella, L., Taillard, E., and Dorigo, M. (1997). « Ant Colonies for the QAP ». Technical Report 97-4, IDSIA, Lugano, Switzerland.

Gambardella, L. M., Taillard, E., and Agazzi, G. (1999a). MACS-VRPTW : A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. In (Corne et al., 1999), pages 63–76.

Gambardella, L. M., Taillard, E., and Dorigo, M. (1999b). « Ant Colonies for the QAP ». *Journal of the Operational Research Society*, 50 :167–176.

Garey, M. and Johnson, D. (1979). *Computers and Intractability : A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, California.

Glover, F. (1989). « Tabu Search - Part I ». *ORSA journal of computing*, 1(3) :190–206.

Glover, F. (1990). « Tabu Search - Part II ». *ORSA journal of computing*, 2(1) :4–32.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Goss, S., Aron, S., Deneubourg, J., and Pasteels, J. (1989). « Self-Organized Shortcuts in the Argentine Ant ». *Naturwissenchaften*, 76 :579–581.

Goss, S. and Deneubourg, J.-L. (1991). « Harvesting by a group of robots ». In (Varela and Bourgine, 1991), pages 195–204.

Grassé, P. (1959). « La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes Natalensis* et *Cubitermes sp.* La théorie de la stigmergie : essai d'interprétation du comportement des termites contructeurs ». *Insectes Sociaux*, 6 :41–80.

Gutjahr, W. (2000). « A Graph-based Ant System and its convergence ». *Future Generation Computer Systems*, 16(8) :873–888.

Hamilton, W. (1964). « The genetical evolution of social behaviour I, II ». *Journal of Theorical Biology*, 7 :152.

Hansen, N., Ostermeier, A., and Gawelczyk, A. (1995). « On the adaptation of arbitrary normal mutation distributions in evolution strategies : The generating set adaptation ». In (Eshelman, 1995), pages 57–64.

Hertz, J., Krog, A., and Palmer, G. (1993). *Introduction to the Theory of Neural Computation*. Addison-Wesley.

Heusse, M., Snyers, D., Guérin, S., and Kuntz, P. (1998). « Adaptive Agent-Driven Routing and Load Balancing in Communication Networks ». Technical Report RR-98001-IASC, ENST de Bretagne.

Holland, J. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor.

Hölldobler, B. and Wilson, E. (1990). *The Ants*. Springer Verlag, Berlin, Germany.

Hölldobler, B. and Wilson, E. (1996). *Voyage chez les Fourmis*. Seuil.

Husbands, P. and Harvey, I., editors (1997). *Fourth European Conference on Artificial Life*. MIT Press, Boston.

IEEE, editor (1995). *Second IEEE International Conference on Evolutionary Computation*. IEEE Press.

IEEE, editor (1997). *Fourth IEEE International Conference on Evolutionary Computation*. IEEE Press.

Jain, A. and Dubes, R. (1988). *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series.

Jaisson, P. (1993). *La fourmi et le sociobiologiste*. Odile Jacob, Paris.

Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., and Wang, A. (1992). « Evolution as a Theme in Artificial Life : The Genesys/Tracker System ». In (Langton et al., 1992), pages 549–578.

Jones, D. and Beltrano, M. (1991). « Solving partitioning problems with genetic algorithms ». In (Belew and Booker, 1991), pages 442–449.

Kaelbling, L. (1993). *Learning in Embedded Systems*. MIT Press.

Kawamura, H., Yamamoto, M., Suzuki, K., and Ohuchi, A. (1998). « Multiple ant colonies approach for search space sharing ».

Kawamura, H., Yamamoto, M., Suzuki, K., and Ohuchi, A. (1999). « Ants War with Evolutive Pheromone Style Communication ». In (Floreano et al., 1999), pages 639–643.

Kawamura, H., Yamamoto, M., Suzuki, K., and Ohuchi, A. (2000). « Multiple Ant Colonies Algorithm Based on Colony Level Interaction ». *IEICE Transactions Fundamentals*, E83-A(2) :371–379.

Kennedy, J. and Eberhart, R. (1999). « *The particle Swarm : Social Adaptation in Information-Processing Systems* », pages 379–388. In (Corne et al., 1999).

Kettaf, F. (1997). « *Contributions des algorithmes évolutionnaires au partitionnement des données* ». PhD thesis, Université de Tours.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). « Optimization by Simulated Anealing ». *Science*, 220 :671–680.

Kohonen, T. (1988). *Self-Organization and Associative Memory*. Springer-Verlag, second edition.

Kortenkamp, D., Bonasso, R., and R., M., editors (1998). *Artificial Intelligence and Mobile Robots*. MIT Press.

Koza, J. (1992). « Genetic Evolution and Co-Evolution of Computer Programs ». In (Langton et al., 1992), pages 603–629.

Koza, J. (1994). *Genetic Programming : On the Programming of Computers by means of Natural Evolution*. MIT Press.

Krieger, M. and Billeter, J. (1999). « Self-Organized Task Allocation in a Population up to Twelve Mobile Robots ». Technical Report, LAMI-EPFL, Lausanne, Switzerland.

Kube, C. and Bonabeau, E. (1999). « Cooperative Transport By Ants and Robots ». Working Paper 99-01-008, Santa Fe Institute.

Kube, R. and Zhang, H. (1992). « Collective Robotics : From Social Insects to Robots ». *Adaptive Behavior*, 2 :189–218.

Kuntz, P., Layzell, P., and Snyers, D. (1997). « A Colony of Ant-like Agents for Partitioning in VLSI Technology ». In (Husbands and Harvey, 1997), pages 417–424.

Kuntz, P. and Snyers, D. (1994). « Emergent Colonization and Graph Partitioning ». In (Cliff et al., 1994), pages 494–500.

Kuntz, P., Snyers, D., and Layzell, P. (1998). « A stochastic heuristic for visualising graph clusters in a bi-dimensional space prior to partitioning ». *Journal of Heuritics*.

Lambrinos, D., Maris, M., Kobayashi, H., Labhart, T., Pfeifer, R., and Wehner, R. (1997). « An Autonomous Agent Navigating with a Polarized Light Compass ». *Adaptive Behavior*, 6(1) :131–161.

Langham, A. and Grant, P. (1999a). « A Multilevel k-way Partitioning Algorithm for Finite Element Meshes using Competing Ant Colonies ». In (Banzhaf et al., 1999), pages 1602–1608.

Langham, A. and Grant, P. (1999b). « Using Competing Ant Colonies to Solve k-way Partitioning Problems with Foraging and raiding strategies ». In (Floreano et al., 1999), pages 621–625.

Langton, C. (1989). « Artificial Life ». In Langton, C., editor, *Proccedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (Artificial Life I)*, volume 6, pages 1–47.

Langton, C., Taylor, C., Farmer, J., and Rasmussen, S., editors (1992). *Artificial Life II*, volume 10. Addison-Wesley.

Laügt, D., Tercinet, F., Monmarché, N., and T'Kindt, V. (2000). « Une heuristique basée sur les colonies de fourmis pour résoudre un problème d'ordonnancement bicritère ». Rapport interne 231, Laboratoire d'Informatique de l'Université de Tours, E3i Tours.

Laurin, S. (1998). « Extention de l'algorithme API au Problème du Voyageur de Commerce ». Rapport de projet de fin d'études, École d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

Lebœuf, C., Roque, J., and Guegand, J. (1983). *Cours de probabilités et de statistiques*. Ellipse, seconde edition.

Leguizamón, G. and Michalewicz, Z. (1999). « A New Version of Ant System for Subset Problems ». In (Angeline et al., 1999).

Lumer, E. and Faieta, B. (1994). « Diversity and Adaptation in Populations of Clustering Ants ». In (Cliff et al., 1994), pages 501–508.

Mandischer, M. (1995). « Evolving recurrent neural networks with non-binary encoding ». In (IEEE, 1995), pages 584–588.

Maniezzo, V. (1998). « Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem ». Technical Report CSR 98-1, C. L. in Scienze dell'Informazione, Università di Bologna, Sede di Cesena, Italy.

Maniezzo, V. and Carbonaro, A. (1998). « An ANTS heuristic for the Frequency Assignment Problem ». Technical Report CSR 98-4, C. L. in Scienze dell'Informazione, Università di Bologna, Sede di Cesena, Italy.

Maniezzo, V. and Carbonaro, A. (2000). « An ANTS heuristic for the frequency assignment problem ». *Future Generation Computer Systems*, 16(8) :927–935.

Maniezzo, V. and Colorni, A. (1999). The Ant System Applied to the Quadratic Assignment Problem. In *IEEE Transactions on Knowledge and Data Engineering*. to appear.

Maniezzo, V., Colorni, A., and Dorigo, M. (1994). « The Ant System Applied to the Quadratic Assignment Problem ». Technical Report 94-28, IRIDIA, Université Libre de Bruxelles, Belgium.

Männer, R. and Manderick, B., editors (1992). *Second International Conference on Parallel Problem Solving from Nature (PPSN II)*. Elsevier Science.

Mariano, C. and Morales, E. (1999). « MOAQ an Ant-Q Algorithm for Multiple Objective Optimization Problems ». In (Banzhaf et al., 1999), pages 894–901.

Martinoli, A., Ijspeert, A., and Gambardella, L. (1999). « A Probabilistic Model for Understanding and Comparing Collective Aggregation Mechanisms ». In (Floreano et al., 1999), pages 575–584.

McCulloch, W. and Pitts, W. (1943). « A Logical Calculus of the Ideas Immanent in Nervous Activity ». *Bulletin of Mathematical Biophysics*, 5 :115–133.

McLurkin, J. « Using Cooperative Robots for Explosive Ordnance Disposal ». `http ://www.ai.mit.edu/people/jdmac/old-jdmac.html`.

Melhuish, C. (1999). « Exploiting Domain Physics : Using Stigmergy to Control Cluster Building with Real Robots ». In (Floreano et al., 1999), pages 585–595.

Melhuish, C., Holland, O., and S.E.J., H. (1998). « Collective sorting and segregation in robots with minimal sensing ». In (Pfeifer et al., 1998), pages ? ?– ? ?

Meslet, O. (1998). « Extention de l'algorithme API : Multipopulation et Recrutement pour l'optimisation numérique ». Rapport de projet de fin d'études, École d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

Meyer, J. and Wilson, S., editors (1990). *First International Conference on Simulation of Adaptive Behavior*. MIT Press, Cambridge, Massachusetts.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, third edition.

Michel, R. and Middendorf, M. (1998). « An Island Model Based Ant System with Lookahead for the Shortest Supersequence Problem ». In (Eiben et al., 1998a), pages 692–701.

Michel, R. and Middendorf, M. (1999). An ACO Algorithm for the Shortest Common Supersequence Problem. In (Corne et al., 1999), pages 51–61.

Monmarché, N. (1999). « On data clustering with artificial ants ». In Freitas, A., editor, *AAAI-99 & GECCO-99 Workshop on Data Mining with Evolutionary Algorithms : Research Directions*, pages 23–26, Orlando, Florida.

Monmarché, N., Desbarats, L., Slimane, M., and Venturini, G. (1998). « Etude d'un nouvel algorithme d'optimisation et d'apprentissage inspiré d'une colonie de fourmis *Pachycondyla apicalis* ». In *Colloque Intelligence Artificielle et Complexité*, pages 114–119, Paris, France.

Monmarché, N., Nocent, G., Venturini, G., and Santini, P. (1999). « On generating HTML style sheets with an interactive genetic algorithm based on gene frequencies. ». In *Artificial Evolution 99, Lecture Notes in Artificial Intelligence*, volume 1829, pages 99–110, Dunkerque, France. Springer-Verlag.

Monmarché, N., Ramat, E., Desbarats, L., and Venturini, G. (2000a). « Probabilistic Search with Genetic Algorithms and Ant Colonies ». In Wu, A., editor, *Proceedings of the Optimization by building and using probabilistic models workshop, Genetic and Evolutionary Computation Conference*, pages 209–211, Las Vegas, Nevada.

Monmarché, N., Ramat, E., Dromel, G., Slimane, M., and Venturini, G. (1999a). « On the similarities between AS, BSC and PBIL : toward the birth

of a new meta-heuristic ». Technical Report 215, Laboratoire d'Informatique de l'Université de Tours, E3i Tours. 14 pages.

Monmarché, N., Slimane, M., and Venturini, G. (1999b). « AntClass : discovery of clusters in numeric data by an hybridization of an ant colony with the Kmeans algorithm ». Technical Report 213, Laboratoire d'Informatique de l'Université de Tours, E3i Tours. 21 pages.

Monmarché, N., Slimane, M., and Venturini, G. (1999c). « On how the ants *Pachycondyla apicalis* are suggesting a new search algorithm ». Technical Report 214, Laboratoire d'Informatique de l'Université de Tours, E3i Tours. 17 pages.

Monmarché, N., Slimane, M., and Venturini, G. (1999a). « On Improving Clustering in Numerical Databases with Artificial Ants ». In Floreano, D., Nicoud, J., and Mondala, F., editors, *Lecture Notes in Artificial Intelligence*, pages 626–635, Swiss Federal Institute of Technology, Lausanne, Switzerland. Springer-Verlag.

Monmarché, N., Venturini, G., and Slimane, M. (1997). « Optimisation de Chaînes Markov Cachées par une population de fourmis *Pachycondyla apicalis* ». Rapport interne 195, Laboratoire d'Informatique de l'Université de Tours, E3i Tours. 24 pages.

Monmarché, N., Venturini, G., and Slimane, M. (1998). « On how the ants *Pachycondyla apicalis* are suggesting a new search algorithm ». Presented at the First International Workshop on Ant Colony Optimization, Brussels, Belgium.

Monmarché, N., Venturini, G., and Slimane, M. (1999b). « AntClass, Découverte de classes dans des données numériques grâce à l'hybridation d'une colonie de fourmis et l'algorithme des centres mobiles ». In *Conférence d'Apprentissage*, pages 169–176, Ecole Polytechnique, Palaiseau, France.

Monmarché, N., Venturini, G., and Slimane, M. (1999c). « Classification non supervisée par une population de fourmis artificielles ». In *Actes des Colloques Insectes Sociaux*, volume 13, pages 43–52, Tours, France.

Monmarché, N., Venturini, G., and Slimane, M. (2000b). « On how *Pachycondyla apicalis* ants suggest a new search algorithm ». *Future Generation Computer Systems*, 16(8) :937–946.

Mühlenbein, H., Schomisch, M., and Born, J. (1991). « The parallel genetic algorithm as function optimizer ». In (Belew and Booker, 1991), pages 271–278.

Navarro Varela, G. and Sinclair, M. (1999). « Ant Colony Optimisation for Virtual-Wavelength-Path Routing and Wavelength Allocation ». In *Congress on Evolutionary Computation*, Washington DC, USA.

Pelikan, M., Goldberg, D., and Lobo, F. (1999). « A Survey of Optimization by Building and Using Probabilistic Models ». IlliGAL Report 99018, Illinois Genetic Algorithms Laboratory, University of Illinois.

Pfeifer, R., Blumberg, B., Meyer, J., and Wilson, S., editors (1998). *Fith International Conference on Simulation of Adaptive Behavior*. MIT Press.

Pimon, S. and Solnon, C. (2000). « A Generic Ant Algorithm for Solving Constraint Satisfaction Problems ». Technical Report, LISI, Université Lyon1.

PRICE, K. (1999). « *An Introduction to Differential Evolution* », pages 79–108. In (Corne et al., 1999).

PRIEDITIS, A. and RUSSELL, S., editors (1995). *Twelfth International Conference on Machine Learning, Lake Tahoe, California*. Morgan Kaufmann, San Mateo, California.

QUINN, R. and ESPENSCHIELD, K. (1993). Control of a Hexapod Robot using a Biologically Inspired Neural Network. In BEER, R., RITZMANN, R., and T., M., editors, *Biological Neural Networks in Invertebrate Neuroethothology and Robotics*, pages 365–381. Academic Press, San Diego, CA.

RABINER, L. (1989). « A tutorial on hidden Markov models and selected applications in speech recognition ». In *Proceedings of IEEE*, volume 77, pages 257–286.

RAMAT, E., VENTURINI, G., LENTE, C., and SLIMANE, M. (1997). « Solving the Multiple Resource Constrained Project Scheduling Problem with Hybrid Genetic Algorithm ». In BÄCK, T., editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 489–496. Morgan Kaufmann, San Francisco, CA.

REINELT, G. (1994). The traveling salesman, Computational solutions for TSP applications. In *Lecture Notes in Computer Science*, volume 840. Springer Verlag, Heidelberg, Germany.

REINELT, G. (1995). « TSPLIB ». Universität Heidelberg, Institut für Angewandte Mathematik, `http ://www.iwr.uni-heidelberg.de/iwr/comopt/TSPLIB95/TSPLIB.html`.

ROUX, O., FONLUPT, C., TALBI, E.-G., and ROBILLARD, D. (1999). « ANTabu - enhanced version ». Technical Report LIL-99-1, Laboratoire d'Informatique du Litoral.

RUDLOF, S. and KOEPPEN, M. (1996). « Stochastic Hill Climbing with Learning by Vectors of Normal Distributions ». In *Proceedings of the first Online Workshop on Soft Computing*. `http ://www.bioele.nuee.nagoya-u.ac.jp/wsc1/papers/p077.html`.

RUMELHART, D., HINTON, G., and WILLIAMS, R. (1986). « Learning Internal Representations by Error Propagation ». *Parallel Distributed Processing*, 1 :318–362.

RUSSELL, A., D., T., and MACKAY-SIM, A. (1994). « Sensing Odour Trails for Mobile Robot Navigation ». In *IEEE International Conference on Robotics and Automation*, pages 2672–2677, San Diego, CA.

SAMARIA, F. (1994). « *Face recognition using HMMs* ». PhD thesis, Cambridge University.

SCHATZ, B., CHAMERON, S., BEUGNON, G., and COLLET, T. (1999). « Path integration as a source of reinforcement signals for visual sequence learning in the ant *Cataglyphis cursor* ». *Nature*, 399 :769–772.

SCHATZ, B., LACHAUD, J., and BEUGNON, G. (1997). « Graded recruitment and hunting strategies linked to prey weight and size in the ponerine ant *Ectatomma ruidum* ». *Behav. Ecol. Sociobiol.*, 40 :337–349.

Schoonderwoerd, R., Holland, O., Bruten, J., and Rothkrantz, L. (1997). « Ant-based Load Balancing in Telecommunications Networks ». *Adaptive Behavior*, 5(2) :169–207.

Schwefel, H. (1995). *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York, second edition.

Schwefel, H. (1998). « Evolution Strategies, Origin, Contemporary Incarnations, and Applications ». Evonet Summer School on Evolutionary Computation, E3i, University of Tours, France, G. Venturini and A.E. Eiben (Editors).

Sebag, M., editor (1999). *Conférence d'apprentissage, Palaiseau, France*. Association Française pour l'Intelligence Artificielle.

Sebag, M. and Ducoulombier, A. (1998). « Extending Population-Based Incremental Learning to Continuous Spaces ». In (Eiben et al., 1998a).

Sebag, M. and Schoenauer, M. (1997). « A Society of Hill-Climbers ». In *IEEE International Conference on Evolutionary Computation, Indianapolis*, pages 319–324.

Sebag, M., Schoenauer, M., and Maitournam, H. (1997). Parametric and non-parametric identification of macro mechanical models. In Quadraglia, D., Periaux, J., Poloni, C., and Winter, G., editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Sciences*, pages 327–340. Wiley, J.

Sendova-Franks, A. and Franks, N. (1992). « Task allocation in ant colonies within variable environments (a study of temporal polyethism) ». *Bulletin of Mathematical Biology*, 55 :75–96.

Sigel, E., Denby, B., and Le Hégarat-Mascle, S. (2000). « Application of Ant Colony Optimization to Adaptive Routing in LEO Telecommunications Satellite Network ». submitted to IEEE Transactions on Networking.

Slimane, M. and Asselin de Beauville, J.-P. (1996). « Introduction aux modèles de Markov cachés (1ère partie) ». Technical Report 171, Laboratoire d'Informatique de l'Université de Tours, E3i Tours.

Slimane, M., Asselin de Beauville, J.-P., Brouard, T., Venturini, G., and Sealelli, J.-M. (1996a). « Hybridation d'une Chaîne de Markov Cachée et d'un algorithme génétique : application à la reconnaissance des formes dans les images ». In *Proceedings of Automatisme et Génie Informatique, Tours*, pages 435–438.

Slimane, M., Brouard, T., Venturini, G., and Asselin de Beauville, J.-P. (1999). « Apprentissage non-supervisé d'images par hybridation génétique d'une chaîne de Markov cachée ». *Traitement du signal*, 16(6) :461–475.

Slimane, M., Venturini, G., Asselin de Beauville, J.-P., and Brouard, T. (1998). « Hybrid Genetic Learning of Hidden Markov Models for Time Series Prediction ». *Biomimetic approaches in management science, Kluwer Academics*.

Slimane, M., Venturini, G., Asselin de Beauville, J.-P., Brouard, T., and Brandeau, A. (1996b). « Optimizing Hidden Markov Models with a Genetic Algorithm ». In *Artificial Evolution, Lecture Notes in Computer Science*, volume 1063, pages 384–396. Springer Verlag.

Solnon, C. (2000a). « Ant-P-solveur : un solveur de contraintes à base de fourmis artificielles ». In *Journées Francophones sur la Programmation Logique et par Contraintes*. Hermes. à paraitre.

Solnon, C. (2000b). « Solving Permutation Constraint Satisfaction Problems with Artificial Ants ». In *Prooceedings of the 14th European Conference on Artificial Intelligence*.

Steinberg, D. (1998). « Découverte de classe dans des données par une colonie de fourmis ». Rapport de projet de fin d'études, École d'Ingénieurs en Informatique pour l'Industrie (E3i), Université de Tours, France.

Steinberg, D., Monmarché, N., Slimane, M., and Venturini, G. (1998a). « Discovery of clusters in numeric data by an hybridization of an ant colony with the minimum distance classification ». Presented at the First International Workshop on Ant Colony Optimization, Brussels, Belgium.

Steinberg, D., Monmarché, N., Slimane, M., Venturini, G., and Guinot, C. (1998b). « Découverte de classes dans des données numériques par hybridation d'une colonie de fourmis avec les centres mobiles ». In *Sixièmes rencontres de la Société Francophone de Classification*, pages 211–214, Montpellier, France.

Stützle, T. (1998a). « An Ant Approach to the Flow Shop Problem ». In *EUFIT'98*, pages 1560–1564, Aachen.

Stützle, T. (1998b). « Parallelisation Strategies for Ant Colony Optimization ». In (Eiben et al., 1998a).

Stützle, T. and Dorigo, M. (1999a). ACO Algorithms for the Quadratic Assignment Problem. In (Corne et al., 1999), pages 33–50.

Stützle, T. and Dorigo, M. (1999b). ACO Algorithms for the Traveling Salesman Problem. In Miettinen, K., Mäkelä, M., Neittaanmäki, P., and Periaux, J., editors, *Evolutionary Algorithms in Engineering and Computer Science : Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications*. John Wiley & Sons.

Stützle, T. and Hoos, H. (1997a). « The $\mathcal{MAX} - \mathcal{MIN}$ Ant System and local Search for Combinatorial Optimization Problems : Towards Adaptive Tools for Global Optimization ». In *2nd Metaheuristics International Conference*, Sophia-Antipolis, France.

Stützle, T. and Hoos, H. (1997b). « $\mathcal{MAX} - \mathcal{MIN}$ Ant System and Local Search for the Traveling Salesman Problem ». In (IEEE, 1997), pages 308–313.

Stützle, T. and Hoos, H. (1997c). « Improvements on the Ant System : Introducing the $\mathcal{MAX} - \mathcal{MIN}$ Ant System ». In *Third International Conference on Artificial Neural Networks and Genetic Algorithms*, University of East Anglia, Norwich, UK. Springer Verlag.

Stützle, T. and Hoos, H. (2000). « $\mathcal{MAX} - \mathcal{MIN}$ Ant System ». *Future Generation Computer Systems*, 16(8) :889–914.

Syswerda, G. (1993). « Simulated Crossover in Genetic Algorithms ». In Whitley, L., editor, *Second workshop on Foundations of Genetic Algorithms*, pages 239–255, San Mateo, California. Morgan Kaufmann.

TAILLARD, É. (1998). « FANT : Fast Ant System ». Technical Report 46-98, IDSIA, IDSIA, Lugano.

TAILLARD, É. (1999). Ant Systems. In PARDALOS, P. and RESENDE, M., editors, *Handbook of Applied Optimization*.

TAILLARD, E. and GAMBARDELLA, L. M. (1997). « An Ant Approach for Structured Quadratic Assignment Problems ». In *2nd Metaheuristics International Conference*, Sophia-Antipolis, France.

TALBI, E.-G., ROUX, O., FONLUPT, C., and ROBILLARD, D. (1999). « Parallel Ant Colonies for Combinatorial Optimization Problems ». In ROLIM, J., editor, *Workshop on Biologically Inspired Solutions to Parallel Processing Systems*. Springer-Verlag.

THERAULAZ, G., BONABEAU, E., and DENEUBOURG, J. (1998). « Response thresold reinforcement and division of labour in social insect societies ». Working Paper 98-01-006, Santa Fe Institute.

T'KINDT, V., MONMARCHÉ, N., LAÜGT, D., and TERCINET, F. (2000). « Combining Ant Colony Optimization and Simulated Annealing to solve a 2-machine flowshop bicriteria scheduling problem ». In *Proceedings of the European Chapter on Combinatorial Optimization*, pages 129–130, Capri, Italy.

TORRE, F. (1999). « Les Vraizamis ». In (Sebag, 1999), pages 177–184.

TOU, J. and GONZALES, R. (1974). *Pattern recognition Principles*. Addison-Wesley.

ÜNSAL, C. and BAY, J. « Spacial Self-Organization in Large Population of Mobile Robots ». `http ://armyant.ee.vt.edu/unsalWWW/spacial.html`.

VANDER MEER, R., BREED, M., K.E., E., and M.L., W., editors (1998). *Pheromone Communication in Social Insects*. Westview Press.

VARELA, F. and BOURGINE, P., editors (1991). *First European Conference on Artificial Life, Paris, France*. MIT Press, Cambridge, Massachusetts.

VENTURINI, G. (1997). « *Apport des algorithmes génétiques à l'apprentissage et à l'optimisation* ». Habilitation à diriger les recherches, Université de Tours.

VENTURINI, G., SLIMANE, M., MONMARCHÉ, N., and FRESNEAU, D. (1997). « Modélisation des fourmis *Pachycondyla apicalis* appliquée à l'optimisation numérique ». Rapport interne 194, Laboratoire d'Informatique de l'Université de Tours, E3i Tours. 15 pages.

VOIGT, H.-M., EBELING, W., RECHENBERG, I., and SCHWEFEL, H.-P., editors (1996). *Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV)*. Springer-Verlag, Berlin.

VON LASZEWSKI, G. (1991). « Intelligent Structural Operators for the k-Way Graph Partioning Problem ». In (Belew and Booker, 1991), pages 45–52.

WAGNER, I. and BRUCKSTEIN, A. (1995). « Cooperative Clearners : a Study in Ant-Robotics ». Technical Report CIS-9512, Center for Intelligent Systems.

WAGNER, I. and BRUCKSTEIN, A. (1999). « Hamiltonian(t) - An Ant-Inspired Heuristic for Recognizing Hamiltonian Graphs ». In *Conference on Evolutionary Computation*.

Wagner, I., Lindenbaum, M., and Bruckstein, A. (1999). « Distributed Covering by Ant-Robots Using Evaporating Traces ». *IEEE Transactions on Robotics and Automation*, 15(5) :918–933.

Wagner, I., Lindenbaum, M., and Bruckstein, A. (2000). « ANTS : Agent, Networks, Trees, and Subgraphs ». *Future Generation Computer Systems*, 16(8) :915–926.

Wemmert, C., Gancarskii, P., and Korczak, J. (1999). « Un système de raffinement non-supervisé d'un ensemble de hiérarchies de classes ». In (Sebag, 1999), pages 153–160.

White, T., Pagurek, B., and Oppacher, F. (1998). « Connection management using adaptive mobile agents ». In Arabnia, H., editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pages 802–809. CSREA Press.

Whitley, D. (1991). « Fundamental principles of deception in genetic search ». In Rawlins, G., editor, *First workshop on Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann.

Whitley, D., Mathias, K., and Fitzhorn, P. (1991). « Delta Coding : an iterative search strategy for Genetic Algorithms ». In (Belew and Booker, 1991), pages 77–84.

Whitley, D., Mathias, K., Rana, S., and Dzubera, J. (1995). « Building better test functions ». In (Eshelman, 1995), pages 239–246.

Whitley, D., Mathias, K., Rana, S., and Dzubera, J. (1996). « Evaluating Evolutionary Algorithms ». *Artificial Intelligence*, 85 :245–276.

Whitley, D., Rana, S., and Heckendorn, R. B. (1997). « Island Model Genetic Algorithms and Linearly Separable Problems ». In *Proceedings of the AISB Workshop on Evolutionary Computation*.

Widrow, B. (1990). « 30 Years of Adaptive Neural Networks : Perceptron, Madaline and Backpropagation ». In *Proceedings of the IEEE*, volume 78, pages 1415–1442.

Wilson, E. (1975). *Sociobiology, The New Synthesis*. Harward University Press.

Wodrich, M. (1996). « *Ant Colony Optimisation, an empirical investigation into an ant colony metaphor for continuous function optimisation* ». Undergraduate thesi, Department of Electrical and Electronic Engineering, University of Cape Town.

Yao, X. (1993). « A review of evolutionary artificial neural networks ». *International Journal of Intelligent Systems*, 8 :539–567.

Annexe A

L'algorithme ISODATA

A.1 Introduction

Cette annexe décrit l'algorithme ISODATA[1](Ball and Hall, 1965; Ball et al., 1969; Tou and Gonzales, 1974) de classification non supervisée. Cette méthode permet de classer un ensemble O d'individus (objets) caractérisés par des variables quantitatives. On note n le nombre d'objets à classer, possédant chacun p paramètres numériques notés $(x_i^1, x_i^2, \ldots, x_i^p)$ pour l'objet x_i.

A.2 Notations

Voici les valeurs utilisées par la suite :
- la distance d utilisée est la distance euclidienne ;
- la distance maximale entre deux objets de O :

$$d_{\max} = \max_{(i,j)\in[1,n]^2, i\neq j} d(x_i, x_j) \tag{A.1}$$

- la distance minimale entre deux objets de O :

$$d_{\min} = \min_{(i,j)\in[1,n]^2, i\neq j} d(x_i, x_j) \tag{A.2}$$

- la distance moyenne entre deux objets de O :

$$d_{\text{mean}} = \frac{2}{n(n-1)} \sum_{(i,j)\in[1,n]^2, i\neq j} d(x_i, x_j) \tag{A.3}$$

- la dispersion maximale $\text{Disp}_{\max}(j)$ de la classe j est définie de la façon suivante :

$$\text{Disp}_{\max}(j) = \max_{1\leq k\leq p} \left\{ \sqrt{\frac{1}{n_j}} \sum_{i\in I_j} \left(x_i^k - z_j^k\right)^2 \right\} \tag{A.4}$$

  où n_j est l'effectif de la classe j, I_j est l'ensemble des indices des objets de la classe j, $z_j = (z_j^1, \ldots, z_j^p)$ est le centre de gravité de la classe j ;

[1]Iterative Self Organizing Data Analysis Techniques A

– la dispersion moyenne $\text{Disp}_{\text{mean}}$ de O définie de la façon suivante :

$$\text{Disp}_{\text{mean}} = \frac{1}{p}\sum_{k=1}^{p}\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i^k - z^k\right)^2} \tag{A.5}$$

où $z = (z^1, \ldots, z^p)$ est le centre de gravité de O.

A.3 Description de l'algorithme ISODATA

L'algorithme construit une partition initiale et affecte un individu à la classe dont le centre est le plus proche. Ensuite, les classes sont recomposées en suivant un certain nombre de règles :

– éclatement d'une classe si cette classe a une dispersion maximale (Disp_{max}) dépassant un seuil donné (D_{max}).

$$\text{Disp}_{\text{max}} > D_{\text{max}} \tag{A.6}$$

Si une classe j doit être éclatée, on construit deux centres de classe, $z_{j'}$ et $z_{j''}$ identiques à z_j sauf pour la coordonnée k' qui maximise la quantité $\sqrt{\frac{1}{n_j}\sum_{i\in I_j}\left(x_i^k - z_j^k\right)^2}$ et dans ce cas

$$z_{j'}^{k'} \leftarrow z_j^{k'} + D_{\text{max}}\sqrt{\frac{1}{n_j}\sum_{i\in I_j}\left(x_i^{k'} - z_j^{k'}\right)^2} \tag{A.7}$$

$$z_{j''}^{k'} \leftarrow z_j^{k'} - D_{\text{max}}\sqrt{\frac{1}{n_j}\sum_{i\in I_j}\left(x_i^{k'} - z_j^{k'}\right)^2} \tag{A.8}$$

– agglomération de deux classes j' et j'' si la distance entre leurs centres est inférieure à un seuil donné (D_{min}) :

$$d(z_{j'}, z_{j''}) < D_{\text{min}} \tag{A.9}$$

– suppression d'une classe j si son effectif n_j est inférieur à un seuil donné (N_{min}) :

$$n_j > N_{\text{min}} \tag{A.10}$$

L'algorithme A.1 résume le fonctionnement d'Isodata. La principale difficulté que l'on rencontre avec ISODATA est qu'il faut déjà avoir une certaine connaissance des données pour fixer les paramètres. En particulier le paramètre Disp_{max} fixant l'éclatement d'une classe en deux, ainsi que le paramètre D_{min} fixant le rapprochement de deux classes.

Algorithme A.1: Isodata
Entrée: O : l'ensemble des données
Sortie: partition obtenue
ISODATA(O)

(1) Normaliser les données.
(2) Affecter les n objets à c classes au hasard.
(3) Calculer les centres des classes.
(4) Affecter chaque objet à la classe dont le centre est le plus proche.
(5) Eliminer les classes comportant moins de $N_{\min}$ objets, les individus alors orphelins sont affectés aux classes dont les centres sont les plus proches.
(6) Calculer les centres des classes.
(7) Si deux classes sont suffisament proches, les rassembler.
(8) Si la dispersion des objets autour d'une classe est trop importante, éclater la classe en deux classes.
(9) **si** (il y a eu des modifications) OU (un certain nombre T d'itérations n'a pas été atteint) **alors**
(10) **aller en** en 4
(11) **finsi**
(12) Affecter chaque objet à la classe dont le centre est le plus proche.
(13) **retourner** la partition obtenue.

Annexe B

BSC, PBIL, AS_b et ACS_b : Exemples détaillés

Dans cette annexe, nous allons dérouler les quatre algorithmes du chapitre 6 sur un exemple simple afin de comparer les mises à jour du vecteur V.

B.1 Problème et paramètres

La fonction f à minimiser, définie sur quatre bits, est la suivante :

$$f(s) = 2 + s(1) - s(2) + s(3) - s(4) \tag{B.1}$$

l'optimum global est $s^* = 0101$ car $f(s^*) = 0 \leq f(s) \forall s \in \{0000, 0001, \ldots, 1111\}$.

Les paramètres sont les suivants :

- commun : $n = 4$, $T_3 = 12$ (la fonction f est évaluée 12 fois et chaque méthode effectura $12/4 = 3$ itérations),
- BSC : $p_m = 0.05$,
- PBIL : $LR = 0.1$, $p_m = 0.01$, $\delta_m = 0.05$, $LR_{neg} = 0.025$,
- AS_b : $\rho = 0.1$, $\tau^0 = 0.5$,
- ACS_b : $\rho = 0.1$, $\tau^0 = 0.5$, $q_0 = 0.8$, $\alpha = 0.01$.

Nous imposons la génération de la même population pour les trois itérations de chaque méthode afin de mettre en valeur les modifications de V.

B.2 Déroulement de BSC

L'algorithme BSC est décrit dans la section 6.3.

1. **Itération 1** : Génération de la population suivant $V = (0.5, 0.5, 0.5, 0.5)$:

$$P = (1001, 0010, 1110, 0100)$$

2. Evaluation de P :

$$f(s_1) = 2, f(s_2) = 3, f(s_3) = 3, f(s_4) = 1$$

3. Mise à jour de V :
   (a) calcul des rangs :

$$Rank(s_1) = 2, Rank(s_2) = 3, Rank(s_3) = 3, Rank(s_4) = 1,$$

   (b) calcul des poids :

$$\omega(s_1) = 2, \omega(s_2) = 1, \omega(s_3) = 1, \omega(s_4) = 3$$

   (c) calcul de V :

$$V^t = \begin{pmatrix} \frac{2\times1+1\times0+1\times1+3\times0}{2+1+1+3} \\ \frac{2\times0+1\times0+1\times1+3\times1}{2+1+1+3} \\ \frac{2\times0+1\times1+1\times1+3\times0}{2+1+1+3} \\ \frac{2\times1+1\times0+1\times0+3\times0}{2+1+1+3} \end{pmatrix} = \begin{pmatrix} \frac{3}{7} \\ \frac{4}{7} \\ \frac{2}{7} \\ \frac{2}{7} \end{pmatrix} = \begin{pmatrix} 0.43 \\ 0.57 \\ 0.29 \\ 0.29 \end{pmatrix}$$

   (d) mutation :

$$V = (0.44, 0.56, 0.31, 0.31)$$

4. **Itération 2** : Génération de la population suivant $V = (0.44, 0.56, 0.31, 0.31)$:

$$P = (0001, 0110, 1100, 0100)$$

5. Evaluation de P :

$$f(s_1) = 1, f(s_2) = 2, f(s_3) = 2, f(s_4) = 1$$

6. Mise à jour de V :
   (a) calcul des rangs :

$$Rank(s_1) = 1, Rank(s_2) = 2, Rank(s_3) = 2, Rank(s_4) = 1,$$

   (b) calcul des poids :

$$\omega(s_1) = 3, \omega(s_2) = 2, \omega(s_3) = 2, \omega(s_4) = 3$$

   (c) calcul de V :

$$V^t = \begin{pmatrix} \frac{3\times0+2\times0+2\times1+3\times0}{3+2+2+3} \\ \frac{3\times0+2\times1+2\times1+3\times1}{3+2+2+3} \\ \frac{3\times0+2\times1+2\times0+3\times0}{3+2+2+3} \\ \frac{3\times1+2\times0+2\times0+3\times0}{3+2+2+3} \end{pmatrix} = \begin{pmatrix} \frac{2}{10} \\ \frac{7}{10} \\ \frac{2}{10} \\ \frac{3}{10} \end{pmatrix} = \begin{pmatrix} 0.20 \\ 0.70 \\ 0.20 \\ 0.30 \end{pmatrix}$$

   (d) mutation :

$$V = (0.23, 0.68, 0.23, 0.32)$$

7. **Itération 3** : Génération de la population suivant $V = (0.23, 0.68, 0.23, 0.32)$:

$$P = (0101, 0111, 0000, 0100)$$

8. Evaluation de P :

$$f(s_1) = 0, f(s_2) = 1, f(s_3) = 2, f(s_4) = 1$$

9. Mise à jour de V :
   (a) calcul des rangs :

$$Rank(s_1) = 1, Rank(s_2) = 2, Rank(s_3) = 3, Rank(s_4) = 2,$$

   (b) calcul des poids :

$$\omega(s_1) = 3, \omega(s_2) = 2, \omega(s_3) = 1, \omega(s_4) = 2$$

   (c) calcul de V :

$$V^t = \begin{pmatrix} \frac{3\times0+2\times0+1\times0+2\times0}{3+2+1+2} \\ \frac{3\times1+2\times1+1\times0+2\times1}{3+2+1+2} \\ \frac{3\times0+2\times1+1\times0+2\times0}{3+2+1+2} \\ \frac{3\times1+2\times1+1\times0+2\times0}{3+2+1+2} \end{pmatrix} = \begin{pmatrix} \frac{0}{8} \\ \frac{7}{8} \\ \frac{2}{8} \\ \frac{5}{8} \end{pmatrix} = \begin{pmatrix} 0.00 \\ 0.86 \\ 0.25 \\ 0.63 \end{pmatrix}$$

   (d) mutation :

$$V = (0.05, 0.82, 0.28, 0.62)$$

B.3 Déroulement de PBIL

L'algorithme PBIL est décrit dans la section 6.4.

1. **Itération 1** : Génération de la population suivant $V = (0.5, 0.5, 0.5, 0.5)$:

$$P = (1001, 0010, 1110, 0100)$$

2. Evaluation de P :

$$f(s_1) = 2, f(s_2) = 3, f(s_3) = 3, f(s_4) = 1$$

3. Mise à jour de V :
   (a) $s^+ = s_4 = 0100$, $s^- = s_2 = 0010$[1]
   (b) calcul de V :
   – apprentissage positif :

$$V^t = \begin{pmatrix} 0.5 \times 0.9 + 0 \times 0.1 \\ 0.5 \times 0.9 + 1 \times 0.1 \\ 0.5 \times 0.9 + 0 \times 0.1 \\ 0.5 \times 0.9 + 0 \times 0.1 \end{pmatrix} = \begin{pmatrix} 0.45 \\ 0.55 \\ 0.45 \\ 0.45 \end{pmatrix}$$

[1] On aurait pu tout aussi bien prendre s_3.

– apprentissage négatif : seuls les bits 2 et 3 de s^+ et s^- sont différents :

$$V^t = \begin{pmatrix} 0.45 \\ 0.55 \times 0.975 + 1 \times 0.025 \\ 0.45 \times 0.975 + 0 \times 0.025 \\ 0.45 \end{pmatrix} = \begin{pmatrix} 0.45 \\ 0.56 \\ 0.44 \\ 0.45 \end{pmatrix}$$

(c) mutation :

$$V^t = \begin{pmatrix} 0.45 \\ 0.56 \\ 0.44 \\ 0.45 \times 0.95 + 1 \times 0.05 \end{pmatrix} = \begin{pmatrix} 0.45 \\ 0.56 \\ 0.44 \\ 0.48 \end{pmatrix}$$

4. **Itération 2** : Génération de la population suivant $V = (0.45, 0.56, 0.44, 0.48)$:

$$P = (0001, 0110, 1100, 0100)$$

5. Evaluation de P :

$$f(s_1) = 1, f(s_2) = 2, f(s_3) = 2, f(s_4) = 1$$

6. Mise à jour de V :
   (a) $s^+ = s_1 = 0001$, $s^- = s_2 = 0110$[2]
   (b) calcul de V :
   – apprentissage positif :

$$V^t = \begin{pmatrix} 0.45 \times 0.9 + 0 \times 0.1 \\ 0.56 \times 0.9 + 0 \times 0.1 \\ 0.44 \times 0.9 + 0 \times 0.1 \\ 0.48 \times 0.9 + 1 \times 0.1 \end{pmatrix} = \begin{pmatrix} 0.41 \\ 0.50 \\ 0.40 \\ 0.53 \end{pmatrix}$$

– apprentissage négatif : les bits 2, 3 et 4 de s^+ et s^- sont différents :

$$V^t = \begin{pmatrix} 0.41 \\ 0.50 \times 0.975 + 0 \times 0.025 \\ 0.40 \times 0.975 + 0 \times 0.025 \\ 0.53 \times 0.975 + 1 \times 0.025 \end{pmatrix} = \begin{pmatrix} 0.41 \\ 0.49 \\ 0.39 \\ 0.54 \end{pmatrix}$$

(c) mutation : aucune

$$V = (0.41, 0.49, 0.39, 0.54)$$

7. **Itération 3** : Génération de la population suivant $V = (0.41, 0.49, 0.39, 0.54)$:

$$P = (0101, 0111, 0000, 0100)$$

[2]On aurait pu tout aussi bien prendre s_4 et s_3.

8. Evaluation de P :

$$f(s_1) = 0, f(s_2) = 1, f(s_3) = 2, f(s_4) = 1$$

9. Mise à jour de V :
   (a) $s^+ = s_1 = 0101$, $s^- = s_3 = 0000$
   (b) calcul de V :
   – apprentissage positif :

$$V^t = \begin{pmatrix} 0.41 \times 0.9 + 0 \times 0.1 \\ 0.49 \times 0.9 + 1 \times 0.1 \\ 0.39 \times 0.9 + 0 \times 0.1 \\ 0.54 \times 0.9 + 1 \times 0.1 \end{pmatrix} = \begin{pmatrix} 0.37 \\ 0.54 \\ 0.35 \\ 0.59 \end{pmatrix}$$

   – apprentissage négatif : seuls les bits 2 et 4 de s^+ et s^- sont différents :

$$V^t = \begin{pmatrix} 0.37 \\ 0.54 \times 0.975 + 1 \times 0.025 \\ 0.35 \\ 0.59 \times 0.975 + 1 \times 0.025 \end{pmatrix} = \begin{pmatrix} 0.37 \\ 0.55 \\ 0.35 \\ 0.60 \end{pmatrix}$$

   (c) mutation : aucune

$$V = (0.41, 0.49, 0.39, 0.54)$$

B.4 Déroulement de AS$_b$

L'algorithme AS$_b$ est décrit dans la section 6.5.
Les phéromones sont initialisée à $\tau_0 = 0.5$:

$$\tau = \begin{pmatrix} \tau_{0_1} & \tau_{1_1} \\ \tau_{0_2} & \tau_{1_2} \\ \tau_{0_3} & \tau_{1_3} \\ \tau_{0_4} & \tau_{1_4} \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

on définit les matrices Δ^j :

$$\Delta^j = \begin{pmatrix} \Delta^j_{0_1} & \Delta^j_{1_1} \\ \Delta^j_{0_2} & \Delta^j_{1_2} \\ \Delta^j_{0_3} & \Delta^j_{1_3} \\ \Delta^j_{0_4} & \Delta^j_{1_4} \end{pmatrix}$$

1. **Itération 1** : Génération de la population suivant $V = (0.5, 0.5, 0.5, 0.5)$:

$$P = (1001, 0010, 1110, 0100)$$

2. Evaluation de P :

$$f(s_1) = 2, f(s_2) = 3, f(s_3) = 3, f(s_4) = 1$$

3. Mise à jour de V :
   (a) Calcul des Δ^j :

$$\Delta^1 = \begin{pmatrix} 0 & 1/3 \\ 1/3 & 0 \\ 1/3 & 0 \\ 0 & 1/3 \end{pmatrix}, \Delta^2 = \begin{pmatrix} 1/4 & 0 \\ 1/4 & 0 \\ 0 & 1/4 \\ 1/4 & 0 \end{pmatrix},$$

$$\Delta^3 = \begin{pmatrix} 1/4 & 0 \\ 1/4 & 0 \\ 1/4 & 0 \\ 0 & 1/2 \end{pmatrix}, \Delta^4 = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \\ 1/2 & 0 \\ 1/2 & 0 \end{pmatrix}$$

   (b) calcul de τ :

$$\tau = \begin{pmatrix} 0.9 \times 0.5 + 1/4 + 1/4 + 1/2 & 0.9 \times 0.5 + 1/3 \\ 0.9 \times 0.5 + 1/3 + 1/4 + 1/4 & 0.9 \times 0.5 + 1/2 \\ 0.9 \times 0.5 + 1/3 + 1/4 + 1/2 & 0.9 \times 0.5 + 1/4 \\ 0.9 \times 0.5 + 1/4 + 1/2 & 0.9 \times 0.5 + 1/3 + 1/2 \end{pmatrix}$$

$$= \begin{pmatrix} 1.45 & 0.78 \\ 1.28 & 0.95 \\ 1.53 & 0.70 \\ 1.2 & 1.28 \end{pmatrix}$$

   (c) calcul de V :

$$V = (\frac{0.78}{0.78 + 1.45}, \frac{0.95}{0.95 + 1.28}, \frac{0.70}{0.70 + 1.53}, \frac{1.28}{1.28 + 1.2})$$

$$V = (0.35, 0.43, 0.31, 0.52)$$

4. **Itération 2** : Génération de la population suivant $V = (0.35, 0.43, 0.31, 0.52)$:

$$P = (0001, 0110, 1100, 0100)$$

5. Evaluation de P :

$$f(s_1) = 1, f(s_2) = 2, f(s_3) = 2, f(s_4) = 1$$

6. Mise à jour de V :
   (a) Calcul des Δ^j :

$$\Delta^1 = \begin{pmatrix} 1/2 & 0 \\ 1/2 & 0 \\ 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}, \Delta^2 = \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \\ 0 & 1/3 \\ 1/3 & 0 \end{pmatrix},$$

$$\Delta^3 = \begin{pmatrix} 0 & 1/3 \\ 0 & 1/3 \\ 1/3 & 0 \\ 1/3 & 0 \end{pmatrix}, \Delta^4 = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \\ 1/2 & 0 \\ 1/2 & 0 \end{pmatrix}$$

(b) calcul de τ :

$$\tau = \begin{pmatrix} 0.9 \times 0.35 + 1/2 + 1/3 + 1/2 & 0.9 \times 0.35 + 1/3 \\ 0.9 \times 0.43 + 1/2 & 0.9 \times 0.43 + 1/3 + 1/3 + 1/2 \\ 0.9 \times 0.31 + 1/2 + 1/3 + 1/2 & 0.9 \times 0.31 + 1/3 \\ 0.9 \times 0.52 + 1/3 + 1/3 + 1/2 & 0.9 \times 0.52 + 1/2 \end{pmatrix}$$

$$= \begin{pmatrix} 1.65 & 0.65 \\ 0.89 & 1.55 \\ 1.61 & 0.61 \\ 1.63 & 0.97 \end{pmatrix}$$

(c) calcul de V :

$$V = (\frac{0.65}{0.65 + 1.65}, \frac{1.55}{1.55 + 0.89}, \frac{0.61}{0.61 + 1.61}, \frac{0.97}{0.97 + 1.63})$$

$$V = (0.28, 0.64, 0.27, 0.37)$$

7. **Itération 3** : Génération de la population suivant $V = (0.28, 0.64, 0.27, 0.37)$:

$$P = (0101, 0111, 0000, 0100)$$

8. Evaluation de P :

$$f(s_1) = 0, f(s_2) = 1, f(s_3) = 2, f(s_4) = 1$$

9. Mise à jour de V :

(a) Calcul des Δ^j :

$$\Delta^1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \Delta^2 = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \\ 0 & 1/2 \\ 0 & 1/2 \end{pmatrix},$$

$$\Delta^3 = \begin{pmatrix} 1/3 & 0 \\ 1/3 & 0 \\ 1/3 & 0 \\ 1/3 & 0 \end{pmatrix}, \Delta^4 = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \\ 1/2 & 0 \\ 1/2 & 0 \end{pmatrix}$$

(b) calcul de τ :

$$\tau = \begin{pmatrix} 0.9 \times 0.28 + 1 + 1/2 + 1/3 + 1/2 & 0.9 \times 0.28 \\ 0.9 \times 0.64 + 1/3 & 0.9 \times 0.64 + 1 + 1/2 + 1/2 \\ 0.9 \times 0.27 + 1 + 1/3 + 1/2 & 0.9 \times 0.27 + 1/2 \\ 0.9 \times 0.37 + 1/3 + 1/2 & 0.9 \times 0.37 + 1 + 1/2 \end{pmatrix}$$

$$= \begin{pmatrix} 2.59 & 0.25 \\ 0.91 & 2.58 \\ 2.08 & 0.74 \\ 1.17 & 1.83 \end{pmatrix}$$

(c) calcul de V :

$$V = (\frac{0.25}{0.25+2.59}, \frac{2.58}{2.58+0.91}, \frac{0.74}{0.74+2.08}, \frac{1.83}{1.83+1.17})$$
$$V = (0.09, 0.74, 0.26, 0.61)$$

B.5 Déroulement de ACS$_b$

L'algorithme ACS$_b$ est décrit dans la section 6.5.
Les phéromones sont initialisée à $\tau^0 = 0.5$:

$$\tau = \begin{pmatrix} \tau_{0_1} & \tau_{1_1} \\ \tau_{0_2} & \tau_{1_2} \\ \tau_{0_3} & \tau_{1_3} \\ \tau_{0_4} & \tau_{1_4} \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

on définit la matrice Δ :

$$\Delta = \begin{pmatrix} \Delta_{0_1} & \Delta_{1_1} \\ \Delta_{0_2} & \Delta_{1_2} \\ \Delta_{0_3} & \Delta_{1_3} \\ \Delta_{0_4} & \Delta_{1_4} \end{pmatrix}$$

1. **Itération 1** : Génération de la population suivant $V = (0.5, 0.5, 0.5, 0.5)$: Nous ne présentons pas ici le détail de la génération des solutions, à savoir le choix à chaque bit entre l'exploration et l'exploitation. La règle de mise à jour locale est appliquée aprés chaque solution de P générée : pour la première itération cela n'a aucun effet puisque $\tau_{k_i} = \tau^0 : (1-\alpha)\tau_{k_i} + \alpha\tau^0 = 0.99 \times 0.5 + 0.01 \times 0.5 = 0.5$. Considérons que la population générée est alors :

$$P = (1001, 0010, 1110, 0100)$$

2. Evaluation de P :

$$f(s_1) = 2, f(s_2) = 3, f(s_3) = 3, f(s_4) = 1$$

3. Mise à jour de V : $s^{++} = 0100$
   (a) Calcul de Δ :

$$\Delta = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \\ 1/2 & 0 \\ 1/2 & 0 \end{pmatrix}$$

   (b) calcul de τ :

$$\tau = \begin{pmatrix} 0.9 \times 0.5 + 0.1 \times 0.5 & 0.9 \times 0.5 \\ 0.9 \times 0.5 & 0.9 \times 0.5 + 0.1 \times 0.5 \\ 0.9 \times 0.5 + 0.1 \times 0.5 & 0.9 \times 0.5 \\ 0.9 \times 0.5 + 0.1 \times 0.5 & 0.9 \times 0.5 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.45 \\ 0.45 & 0.5 \\ 0.5 & 0.45 \\ 0.5 & 0.45 \end{pmatrix}$$

(c) calcul de V :

$$V = (\frac{0.45}{0.45+0.5}, \frac{0.5}{0.5+0.45}, \frac{0.45}{0.45+0.5}, \frac{0.45}{0.45+0.5})$$
$$= (0.47, 0.53, 0.47, 0.47)$$

4. **Itération 2** : Génération de la population suivant $V = (0.47, 0.53, 0.47, 0.47)$: mises à jour locales :
   – avec $s_1 = 0001$:

$$\tau = \begin{pmatrix} (1-\alpha)\tau_{0_1} + \alpha\tau^0 & \tau_{1_1} \\ (1-\alpha)\tau_{0_2} + \alpha\tau^0 & \tau_{1_2} \\ (1-\alpha)\tau_{0_3} + \alpha\tau^0 & \tau_{1_3} \\ \tau_{0_4} & (1-\alpha)\tau_{1_4} + \alpha\tau^0 \end{pmatrix}$$

$$= \begin{pmatrix} 0.99 \times 0.5 + 0.01 \times 0.5 & 0.45 \\ 0.99 \times 0.45 + 0.01 \times 0.5 & 0.5 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.45 \\ 0.5 & 0.99 \times 0.45 + 0.01 \times 0.5 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.45 \\ 0.4505 & 0.5 \\ 0.5 & 0.45 \\ 0.5 & 0.4505 \end{pmatrix}$$

   – avec $s_2 = 0110$:

$$\tau = \begin{pmatrix} 0.99 \times 0.5 + 0.01 \times 0.5 & 0.45 \\ 0.4505 & 0.99 \times 0.5 + 0.01 \times 0.5 \\ 0.5 & 0.99 \times 0.45 + 0.01 \times 0.5 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4505 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.45 \\ 0.4505 & 0.5 \\ 0.5 & 0.4505 \\ 0.5 & 0.4505 \end{pmatrix}$$

   – avec $s_3 = 1100$:

$$\tau = \begin{pmatrix} 0.5 & 0.99 \times 0.45 + 0.01 \times 0.5 \\ 0.4505 & 0.99 \times 0.5 + 0.01 \times 0.5 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4505 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4505 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.4505 \\ 0.4505 & 0.5 \\ 0.5 & 0.4505 \\ 0.5 & 0.4505 \end{pmatrix}$$

   – avec $s_4 = 0100$:

$$\tau = \begin{pmatrix} 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4505 \\ 0.4505 & 0.99 \times 0.5 + 0.01 \times 0.5 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4505 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4505 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.4505 \\ 0.4505 & 0.5 \\ 0.5 & 0.4505 \\ 0.5 & 0.4505 \end{pmatrix}$$

$$P = (0001, 0110, 1100, 0100)$$

   V n'a pas changé significativement.

5. Evaluation de P :

$$f(s_1) = 1, f(s_2) = 2, f(s_3) = 2, f(s_4) = 1$$

6. Mise à jour de V : $s^{++} = 0100$

(a) Calcul de Δ :

$$\Delta = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \\ 1/2 & 0 \\ 1/2 & 0 \end{pmatrix}$$

(b) calcul de τ :

$$\tau = \begin{pmatrix} 0.9 \times 0.5 + 0.1 \times 0.5 & 0.9 \times 0.4505 \\ 0.9 \times 0.4505 & 0.9 \times 0.5 + 0.1 \times 0.5 \\ 0.9 \times 0.5 + 0.1 \times 0.5 & 0.9 \times 0.4505 \\ 0.9 \times 0.5 + 0.1 \times 0.5 & 0.9 \times 0.4505 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.4055 \\ 0.4055 & 0.5 \\ 0.5 & 0.4055 \\ 0.5 & 0.4055 \end{pmatrix}$$

(c) calcul de V :

$$V = (\frac{0.4055}{0.4055 + 0.5}, \frac{0.5}{0.5 + 0.4055}, \frac{0.4055}{0.4055 + 0.5}, \frac{0.4055}{0.4055 + 0.5})$$

$$= (0.45, 0.55, 0.45, 0.45)$$

7. **Itération 3** : Génération de la population suivant $V = (0.45, 0.55, 0.45, 0.45)$:
   - avec $s_1 = 0101$:

$$\tau = \begin{pmatrix} 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4055 \\ 0.4055 & 0.99 \times 0.5 + 0.01 \times 0.5 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4055 \\ 0.5 & 0.99 \times 0.4055 + 0.01 \times 0.5 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.4055 \\ 0.4055 & 0.5 \\ 0.5 & 0.4055 \\ 0.5 & 0.4064 \end{pmatrix}$$

   - avec $s_2 = 0111$:

$$\tau = \begin{pmatrix} 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4055 \\ 0.4055 & 0.99 \times 0.5 + 0.01 \times 0.5 \\ 0.5 & 0.99 \times 0.4055 + 0.01 \times 0.5 \\ 0.5 & 0.99 \times 0.4064 + 0.01 \times 0.5 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.4055 \\ 0.4055 & 0.5 \\ 0.5 & 0.4064 \\ 0.5 & 0.4073 \end{pmatrix}$$

   - avec $s_3 = 0000$:

$$\tau = \begin{pmatrix} 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4055 \\ 0.99 \times 0.4055 + 0.01 \times 0.5 & 0.5 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4064 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4073 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.4055 \\ 0.4064 & 0.5 \\ 0.5 & 0.4064 \\ 0.5 & 0.4073 \end{pmatrix}$$

   - avec $s_4 = 0100$:

$$\tau = \begin{pmatrix} 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4055 \\ 0.4064 & 0.99 \times 0.5 + 0.01 \times 0.5 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4064 \\ 0.99 \times 0.5 + 0.01 \times 0.5 & 0.4073 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.4055 \\ 0.4064 & 0.5 \\ 0.5 & 0.4064 \\ 0.5 & 0.4073 \end{pmatrix}$$

$$P = (0101, 0111, 0000, 0100)$$

V n'a pas changé significativement.

8. Evaluation de P :

$$f(s_1) = 0, f(s_2) = 1, f(s_3) = 2, f(s_4) = 1$$

9. Mise à jour de V : $s^{++} = 0101$
   (a) Calcul de Δ :

$$\Delta = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

   (b) calcul de τ :

$$\tau = \begin{pmatrix} 0.9 \times 0.5 + 0.1 \times 1 & 0.9 \times 0.4055 \\ 0.9 \times 0.4064 & 0.9 \times 0.5 + 0.1 \times 1 \\ 0.9 \times 0.5 + 0.1 \times 1 & 0.9 \times 0.4064 \\ 0.9 \times 0.5 & 0.9 \times 0.4073 + 0.1 \times 1 \end{pmatrix} = \begin{pmatrix} 0.55 & 0.3650 \\ 0.3658 & 0.55 \\ 0.45 & 0.3658 \\ 0.5 & 0.4666 \end{pmatrix}$$

   (c) calcul de V :

$$V = (\frac{0.3650}{0.3650 + 0.55}, \frac{0.55}{0.55 + 0.3658}, \frac{0.3658}{0.3658 + 0.55}, \frac{0.4666}{0.4666 + 0.45})$$
$$= (0.40, 0.60, 0.40, 0.51)$$

B.6 Commentaires

Le tableau B.1 récapitule les vecteurs de probabilité obtenus par chaque méthode.

Algorithme	V
BSC	$(0.05, 0.82, 0.28, 0.62)$
PBIL	$(0.41, 0.49, 0.39, 0.54)$
AS$_b$	$(0.09, 0.74, 0.26, 0.61)$
ACS$_b$	$(0.40, 0.60, 0.40, 0.51)$

Tab. B.1 – Vecteurs de probabilité obtenus par chaque méthode en trois itérations.

Étant donné la taille réduite de la population, il ne faut pas tirer de conclusions hatives sur les résultats obtenus. En effet, ces algorithmes basent leur efficacité sur leur capacité à échantilloner la population qu'il manipulent, la taille de cette population est donc primordiale à considérer pour assurer de bon résultats. On peut cependant déjà constater que deux groupes se dessinent : (BSC, AS$_b$) et (PBIL, ACS$_b$). Les deux premières méthodes semble faire converger le vecteur V plus rapidement que les deuxièmes.

Index

Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation

Résumé

Dans ce travail de thèse, nous présentons les travaux s'inspirant des fourmis réelles pour la résolution de problèmes en informatique. Nous proposons deux approches supplémentaires de ces nouvelles inspirations biomimétiques. La première reprend certains travaux en classification non supervisée et étend ces principes dans plusieurs directions. L'algorithme AntClass développé à cette occasion, est hybride dans le sens où la recherche du nombre de classes est effectué par des fourmis artificielles et qu'un algorithme classique en classification, les centres mobiles, est utilisé pour gommer les erreurs de classification inhérentes à une méthode stochastique telle que les fourmis artificielles. Après avoir souligné les ressemblances et différences entre les approches évolutionnaires et celles à base de population de fourmis et proposé un modèle commun, nous nous inspirons de la stratégie de recherche de nourriture d'une espèce de fourmis (*Pachycondyla apicalis*) pour résoudre des problèmes d'optimisation globale. L'apport de cette adaptation réside principalement dans sa simplicité. Nous appliquons l'algorithme qui en découle, appelé API, à des problèmes variés tels que l'optimisation de fonctions numériques, l'apprentissage de chaînes de Markov cachées ou des poids d'un réseau de neurones artificiels, ou encore à un problème d'optimisation combinatoire classique : le problème du voyageur de commerce.

Mots clés

fourmis artificielles, algorithmes, optimisation, classification, *Pachycondyla apicalis*.

Artificial ant based algorithms applied to clustering and optimization problems

Abstract

In this thesis, we present works inspired by real ants for the resolution of well known problems in computer science. We propose two supplementary approaches to these new biomimetic inspirations. The first resumes works on clustering and widens their principles in several directions. The AntClass algorithm, developed for occasion, is hybrid in the way that the artificial ants search for the number of clusters and a classic classification algorithm, the K-means algorithm, is used to reduce classification errors inherent to a stochastic method such as artificial ants. After underlying similarities and differences between the evolutionnary approaches and those based on a population of ants, we propose a common model. We finally use the *Pachycondyla apicalis* ants' strategy to search for food to solve global optimization problems. The contribution of this adaptation mainly lies in its simplicity. We apply the ensuring algorithm, called API, to various problems such as the optimization of numerical functions, the learning of hidden Markov models, the learning of the weights of an artificial neural network, or also to a classical combinatorial optimization problem : the traveling salesman problem.

Keywords

artificial ants, algorithms, optimization, clustering, *Pachycondyla apicalis*.

Laboratoire d'Informatique de Tours, UPRES-EA 2101, Equipe Reconnaissance de Formes et Analyse d'Images, École d'Ingénieurs en Informatique pour l'Industrie, 64 avenue J. Portalis, 37200 Tours (`http ://www.e3i.univ-tours.fr`).