IC2

Traité Informatique et Systèmes d'Information

Fourmis artificielles 1

des bases de l'optimisation aux applications industrielles

sous la direction de
Nicolas Monmarché
Frédéric Guinand
Patrick Siarry

Hermes Lavoisier

Fourmis artificielles 1

LAVOISIER
11, rue Lavoisier
75008 Paris

www.hermes-science.com
www.lavoisier.fr

ISBN 978-2-7462-2119-2

Printed and bound in England by CPI Antony Rowe, Chippenham, October 2009.

Fourmis artificielles 1

des bases de l'optimisation
aux applications industrielles

sous la direction de

Nicolas Monmarché
Frédéric Guinand
Patrick Siarry

Hermes Science publications

Lavoisier

Il a été tiré de cet ouvrage
50 exemplaires hors commerce réservés
aux membres du comité scientifique,
aux auteurs et à l'éditeur
numérotés de 1 à 50

Fourmis artificielles 1

sous la direction de Nicolas Monmarché, Frédéric Guinand et Patrick Siarry

fait partie de la série INFORMATIQUE

dirigée par Jean-Charles Pomerol et Francis Sourd

TRAITÉS IC2 INFORMATION – COMMANDE – COMMUNICATION

sous la direction scientifique de Bernard Dubuisson

IC2 – Information, Commande, Communication – répond au besoin de disposer d'un ensemble complet des connaissances et méthodes nécessaires à la maîtrise des systèmes technologiques.

Conçu volontairement dans un esprit d'échange disciplinaire, IC2 représente l'état de l'art dans les domaines suivants retenus par le comité scientifique :

Cognition et traitement de l'information
Information et sciences du vivant
Informatique et systèmes d'information
Management et gestion des STIC
Réseaux et télécoms
Signal et Image
Systèmes automatisés
Technologies et développement durable

Chaque ouvrage décrit aussi bien les aspects fondamentaux qu'expérimentaux. Une classification des différents articles contenus dans chacun, une bibliographie et un index détaillé orientent le lecteur vers ses points d'intérêt immédiats : celui-ci dispose ainsi d'un guide pour ses réflexions ou pour ses choix.

Les savoirs, théories et méthodes rassemblés dans chaque ouvrage ont été choisis pour leur pertinence dans l'avancée des connaissances ou pour la qualité des résultats obtenus dans le cas d'expérimentations réelles.

Liste des auteurs

Patrick ALBERT
ILOG
Gentilly

Sébastien AUPETIT
Laboratoire d'informatique
Université de Tours

Raphaël BLANC
Centre hospitalier Henri Mondor
Créteil

Xavier DELORME
Ecole nationale supérieure des Mines
Saint-Etienne

Johann DRÉO
Thales Research & Technology
Palaiseau

Antoine DUTOT
LITIS
Université du Havre

Stéphane FONT
Supélec
Gif-sur-Yvette

Caroline GAGNÉ
Université du Québec
Chicoutimi
Canada

Xavier GANDIBLEUX
LINA
Université de Nantes

Marc GRAVEL
Université du Québec
Chicoutimi
Canada

Frédéric GUINAND
LITIS
Université du Havre

Laurent HENOCQUE
LSIS
Université de la Méditerranée
Marseille

Alain HERTZ
Ecole polytechnique de Montréal
GERAD
Canada

Arnaud HIRET
EDF R&D
Chatou

Julien JORGE
LINA
Université de Nantes

Mathias KLEINER
LSIS
ILOG
Marseille

Alain LENOIR
IRBI
Université de Tours

Christian MONDON
EDF R&D
Chatou

Nicolas MONMARCHÉ
Laboratoire d'informatique
Université de Tours

Amir NAKIB
LiSSi
Université de Paris 12

Damien OLIVIER
LITIS
Université du Havre

Hamouche OULHADJ
LiSSi
Université de Paris 12

Yoann PIGNÉ
LITIS
Université du Havre

Joaquin RODRIGUEZ
INRTS
Villeneuve d'Ascq

Guillaume SANDOU
Supélec
Gif-sur-Yvette

Patrick SIARRY
LiSSi
Université de Paris 12

Mohamed SLIMANE
Laboratoire d'informatique
Université de Tours

Sihem TEBBANI
Supélec
Gif-sur-Yvette

Nicolas ZUFFEREY
HEC
Université de Genève
Suisse

Table des matières

Introduction

Les fourmis artificielles ont commencé à faire parler d'elles depuis le milieu des années 1990. Après presqu'une quinzaine d'années de développements et de recherches, de thèses et de conférences, il nous a semblé opportun de faire le point sur l'état des travaux, notamment dans la communauté francophone.

Nous n'avons pas visé l'exhaustivité, en lançant ce projet de livre sur les fourmis artificielles, d'abord car cela était presque sûrement voué à l'échec, tant les projets ont été nombreux, et, aussi, parce qu'un ouvrage trop volumineux aurait été d'un attrait limité pour le lecteur débutant. L'objectif de ces deux volumes est aussi de permettre à un lecteur novice, sur la question des fourmis artificielles, de se faire une opinion, sans tomber dans le survol systématique, ou bien l'énumération mécanique, en laissant la place à l'analyse, aux résultats, aux délires parfois.

C'est pour ces raisons que les chapitres sont de portée très variable : certains offrent un panorama très large (l'optimisation combinatoire, la robotique, la classification, etc.), en se concentrant sur les notions de base, alors que d'autres se focalisent sur un domaine (les langues naturelles, le handicap, la bioinformatique, etc.), en présentant plusieurs apports des fourmis artificielles, ou sur un problème particulier (l'optimisation de la fabrication de barres d'aluminium, l'apprentissage de modèles de Markov cachés, le routage dans les réseaux mobiles, etc.).

La synthèse de tous ces travaux a été répartie sur deux volumes. Nous avons traité, pour l'essentiel, dans le volume 1, « des bases de l'optimisation aux applications industrielles », le thème de l'optimisation, car il concerne la majorité des applications des fourmis artificielles en informatique. Le volume 2, « nouvelles directions pour une intelligence collective », donne la parole à des applications plus exploratoires, dont la diffusion dans des applications industrielles est moins assurée.

Introduction rédigée par Nicolas Monmarché, Frédéric Guinand et Patrick Siarry.

A ce jour, quel bilan peut-on dresser des travaux sur les fourmis artificielles ? Tout ce que l'on pouvait en tirer pour résoudre des problèmes informatiques ou mathématiques a-t-il été exploité ? Une des principales caractéristiques chez les fourmis artificielles est l'utilisation de phéromones pour marquer son chemin : mais en a-t-on fait le tour ? Et ne reste-t-il plus qu'à l'appliquer, quasiment automatiquement, lorsqu'un nouveau problème (d'optimisation, le plus souvent) se présente ? Enfin, peut-on parler d'un domaine de recherche constitué par l'étude et l'utilisation des fourmis artificielles ?

Pour répondre à ces questions, il faut commencer par lire les deux volumes, mais surtout s'entendre sur une définition d'une fourmi artificielle. Nous proposons la définition suivante :

> *Une fourmi artificielle est un objet, virtuel ou réel (par exemple un agent logiciel ou un robot), ou encore symbolique (comme un point dans un espace de recherche) qui possède un lien, une similitude (c'est-à-dire un comportement, une caractéristique commune) avec une fourmi réelle.*

Une quasi-constante, dans la réalisation de cette définition, concerne l'aspect collectif de la vie des fourmis. La grande majorité des travaux, dont ceux abordés dans ces deux volumes, prennent en compte le travail collectif en manipulant une population, une colonie de fourmis. Cette définition, au singulier, ne se concrétise pratiquement qu'au pluriel. En ce sens, le domaine des fourmis artificielles, s'il existe, fait partie intégrante du domaine, plus vaste, de l'intelligence en essaim (*swarm intelligence*). L'intelligence en essaim se libère de la liaison avec les fourmis, et les modèles utilisés sortent parfois du cadre de l'inspiration animale (tout en restant le plus souvent dans le cadre du « bioinspiré »).

Quel est donc l'intérêt de se borner aux fourmis, sauf à limiter le champ d'investigation ? Il est probable que cette pulsion soit directement liée au sujet de l'inspiration : les fourmis ont leur autonomie sur Terre, leur diversité, leurs règles, leur réussite écologique qui nous fascinent et leur indépendance nous pousserait à les mettre à part dans la question de l'intelligence en essaim.

Toutes ces questions méritent plusieurs exemples concrets et les avis de plusieurs chercheurs manipulant cette notion de fourmi artificielle. La suite de ce volume est là pour y contribuer.

Nous complétons cette introduction en résumant les douze chapitres formant le volume 1 du livre « Fourmis artificielles ».

Dans le premier chapitre de l'ouvrage, Alain Lenoir et Nicolas Monmarché décrivent succinctement l'organisation de la vie sociale des fourmis réelles et les comportements qui ont été transposés dans les algorithmes de fourmis artificielles. Les

auteurs présentent d'abord la morphologie des vraies fourmis et quelques caractéristiques de leur vie sociale. Puis, ils exposent quelques exemples de modélisation d'agissements auto-organisés : les pistes de phéromones, le tri d'objets, les activités de creusement du nid, l'agrégation d'individus et la modélisation de l'odeur coloniale. Enfin, ils esquissent le passage à la fourmi artificielle, agent créé par l'informaticien pour résoudre ses propres problèmes, en s'efforçant de reproduire l'intelligence collective des fourmis.

Dans le second chapitre, Antoine Dutot, Frédéric Guinand, Damien Olivier et Yoann Pigné exposent les principes généraux mis en œuvre pour la résolution, à l'aide d'algorithmes de colonies de fourmis, de problèmes d'optimisation combinatoire. Une caractéristique des fourmis retient particulièrement l'attention des auteurs : leur capacité à construire et à maintenir des *structures* dans leur environnement. Dans ce chapitre, la structure est considérée comme l'élément central dans le processus de résolution par colonie de fourmis de problèmes d'optimisation combinatoire. La solution à un problème donné peut, en effet, s'exprimer comme une structure dans l'espace de recherche. A la manière des sociétés de fourmis, les systèmes à base de fourmis artificielles ont la capacité de construire, de mettre en évidence et de maintenir des structures. L'optimisation de celles-ci requiert toutefois des éléments additionnels, propres au problème traité.

Le chapitre 3 propose un tour d'horizon des problèmes combinatoires traités *via* les fourmis artificielles. Antoine Dutot et Yoann Pigné passent d'abord en revue les principales approches de colonies de fourmis que l'on trouve dans la littérature, en distinguant la famille des ACOs de l'algorithme *Ant-Based Control*. Les différentes classes de problèmes combinatoires résolus à l'aide de ces approches sont décrites dans la seconde partie du chapitre. Enfin, les auteurs s'intéressent aux problèmes ouverts, dans lesquels les objectifs de l'optimisation sont multiples, ou encore des contraintes mal formalisées – liées notamment aux environnements dynamiques et décentralisés, aux incertitudes, ou aux erreurs dans les données – doivent être prises en compte.

Dans le chapitre 4, Patrick Siarry, Johann Dréo et Nicolas Monmarché analysent l'exploitation des fourmis artificielles pour l'optimisation en variables continues. Plusieurs démarches ont été proposées dans la littérature. La plupart des approches s'inspirent des caractéristiques d'auto-organisation et de mémoire externe des colonies de fourmis, laissant de côté la construction itérative de la solution. Il y a cependant aussi des techniques plus récentes, qui exploitent le caractère probabiliste du formalisme ACO. Le chapitre dresse un panorama des méthodes publiées, en les classant en quatre familles : les algorithmes découlant d'une modélisation directe du comportement des fourmis, les algorithmes à base de discrétisation binaire, les algorithmes à base d'échantillonnage probabiliste et les approches hybrides.

Le chapitre 5 s'intéresse à l'optimisation par colonie de fourmis de la « configuration », en programmation par contraintes : il s'agit d'un formalisme permettant de

représenter des problèmes combinatoires. Patrick Albert, Laurent Henocque et Mathias Kleiner montrent que les colonies de fourmis permettent d'augmenter la taille des problèmes de configuration susceptibles d'être traités par une recherche énumérative des solutions. Les auteurs mettent au point un algorithme pour adapter ACO aux variables ensemblistes sur des domaines ouverts et pour construire des solutions. L'étude expérimentale concluant le chapitre montre que la taille de l'espace multidimensionnel des jeux de paramètres de ACO soulève une difficulté : le calage des paramètres de ACO est alors réalisé en recourant à l'optimisation par essaim particulaire.

Dans le chapitre 6, Guillaume Sandou, Stéphane Font, Sihem Tebbani, Christian Mondon et Arnaud Hiret présentent une application mise en œuvre à EDF : celle de l'affectation optimale d'unités au sein d'un site de production d'énergie. Il s'agit d'un problème d'optimisation « mixte » de très grande dimension, les variables binaires caractérisant l'état de marche/arrêt des installations et les variables réelles les quantités d'énergie produites. Les auteurs mettent au point un algorithme de colonie de fourmis en variables réelles adapté au problème, et montrent l'intérêt de son couplage avec un algorithme génétique. A l'issue de cette phase d'optimisation, la solution trouvée constitue une commande en boucle ouverte du système de production. Or, celui-ci est soumis à de nombreuses incertitudes, notamment sur la demande des consommateurs. Les principes de la commande prédictive sont alors mis en œuvre pour étendre les résultats d'optimisation, dans un contexte de boucle fermée.

Une autre application industrielle est décrite dans le chapitre 7 : il s'agit d'un problème d'ordonnancement posé par la fabrication de barres d'aluminium, dans une usine, au Québec. Les principales difficultés résident dans le nombre élevé de contraintes et dans le temps de calcul, l'évaluation d'une solution demandant une simulation complète de la production. Caroline Gagné et Marc Gravel montrent l'importance des termes de visibilité, pour le guidage efficace du processus de construction des solutions par les fourmis artificielles. La principale innovation tient ici dans l'utilisation de plusieurs visibilités liées au problème. En outre, une procédure spécifique de traitement des objectifs est élaborée pour obtenir des solutions de compromis : cette approche est préférée à celle du traitement lexicographique des objectifs, car elle correspond davantage aux préoccupations d'un planificateur de production.

Les mêmes auteurs, Caroline Gagné et Marc Gravel, décrivent, dans le chapitre 8, la résolution par colonie de fourmis d'un problème d'ordonnancement de chaîne d'assemblage automobile. Il s'agit de déterminer l'ordre dans lequel un ensemble de voitures sont fabriquées sur une ligne composée de trois ateliers consécutifs (tôlerie, peinture et montage). Ce problème industriel a été proposé par Renault, dans le cadre du challenge ROADEF'2005, si bien que l'on dispose des résultats procurés par plusieurs algorithmes concurrents. Les auteurs ont adapté la structure de la trace de phéromone

dans la règle de transition et ils ont eu à nouveau recours à plusieurs termes de visibilité. Les expérimentations numériques ont montré la suprématie des fourmis artificielles pour la construction de très bonnes solutions, vis-à-vis de l'objectif principal, pour les instances de grande taille, pouvant comporter plus de 1 000 véhicules.

Le chapitre 9 présente un algorithme de colonie de fourmis, mis au point par Xavier Gandibleux, Julien Jorge, Xavier Delorme et Joaquin Rodriguez, pour mesurer et optimiser la capacité d'un réseau ferroviaire. Trois caractéristiques sont à souligner dans cette application : la technique de perturbation des phéromones, la stratégie d'autoparamétrage, qui gère la convergence de l'algorithme, et un double mode de construction des solutions. L'algorithme proposé fait appel à un ensemble de recherches locales *ad hoc* au regard du problème de *set packing* à résoudre. Le modèle d'optimisation considéré permet de traiter avec une grande flexibilité les heures d'entrée des trains, ainsi que les parcours autorisés dans l'infrastructure. Il permet aussi de manipuler des préférences sur les parcours empruntés, de distinguer différentes catégories de « trains saturants » et de traiter les contraintes particulières rencontrées dans les gares (arrêt à un quai de capacité suffisante, par exemple).

Dans le chapitre 10, Amir Nakib, Raphaël Blanc, Hamouche Oulhadj et Patrick Siarry exposent une nouvelle méthode de segmentation d'images par résonance magnétique (IRM), basée sur le critère de « variance intraclasse modifiée ». Ce critère prend en considération non seulement l'information sur les niveaux de gris, mais aussi la distribution spatiale des niveaux de gris, en exploitant les surfaces des régions segmentées. La complexité algorithmique résultante soulève une difficulté, notamment pour les applications temps réel. Le problème est résolu par une technique de fourmis artificielles, qui procure une segmentation satisfaisante des images IRM : la validation est effectuée sur des images de cerveau, provenant du *CHU Henri Mondor* de Créteil. Le faible temps de calcul de cette approche autorise son extension à un problème d'optimisation dynamique : celui de la segmentation de séquences d'images IRM.

Le chapitre 11 s'intéresse au coloriage des sommets d'un graphe par colonie de fourmis. Depuis 1997, plusieurs chercheurs se sont penchés sur l'utilisation des fourmis artificielles pour traiter ce problème classique, et force est de constater que les algorithmes de fourmis sont désormais compétitifs avec les meilleurs algorithmes de coloration connus à ce jour. Le but de ce chapitre est de faire un historique de l'évolution de ces algorithmes de fourmis. Alain Hertz et Nicolas Zufferey commencent par une définition précise du problème à résoudre et décrivent ensuite trois approches de résolution, qui diffèrent par le rôle attribué à chaque fourmi. Les résultats numériques présentés permettent de comparer les différentes approches. Les auteurs observent, à ce propos, que la communauté scientifique n'est pas unanime sur le nom à donner aux diverses métaheuristiques employées : certains considèrent, en effet, les algorithmes de fourmis comme des cas particuliers d'autres algorithmes plus anciens, alors que d'autres les considèrent comme des généralisations ou des variations.

Le dernier chapitre de ce volume (chapitre 12), rédigé par Sébastien Aupetit, Nicolas Monmarché et Mohamed Slimane, est consacré à l'apprentissage des *modèles de Markov cachés*. Il s'agit d'outils statistiques permettant de modéliser, sous la forme de processus stochastiques, des phénomènes temporels dont une partie est observable. Par exemple, dans le cas d'une image représentant une voiture, la partie observable correspond aux pixels, tandis que la partie non observable concerne l'organisation générale de l'image ; une étape d'apprentissage est alors nécessaire pour ajuster le modèle à la séquence des observations, de manière à conceptualiser la voiture. Les auteurs préconisent l'algorithme de fourmis artificielles API pour mener à bien cet apprentissage. Une analyse statistique de API montre, en effet, que ses opérateurs peuvent être généralisés et redéfinis, de façon à améliorer l'efficacité de API dans l'apprentissage d'un modèle de Markov caché.

Chapitre 1

Des fourmis réelles aux fourmis artificielles

La vie sociale des fourmis a fasciné les hommes depuis l'antiquité pour leur courage, leur sagesse et leur aptitude au travail. On leur attribue même des sentiments. Elles sont un modèle d'organisation. Cependant, quand on les observe, on a souvent plutôt l'impression qu'elles s'agitent dans tous les sens et font n'importe quoi. Et pourtant, finalement la brindille est bien entrée dans le nid, ou la proie rapportée au nid rapidement. En réalité, les fourmis ne sont pas des automates programmés tous identiques ; elles sont différentes et ne font pas toutes la même chose. C'est ainsi que dans un système apparemment chaotique, une organisation stable se maintient. Pour expliquer ce phénomène, on a imaginé la notion d'auto-organisation. Les fourmis sont devenues un objet d'étude pour les informaticiens qui modélisent leur comportement afin de mieux comprendre leur organisation sociale. Dans ce chapitre, nous rappellerons d'abord ce que sont les vraies fourmis. Ensuite, nous avons choisi de présenter quelques exemples de modélisation. Dans une troisième partie, nous verrons que la fourmi devient purement virtuelle quand les informaticiens ont utilisé le modèle fourmi pour résoudre leurs propres problèmes.

1.1. Les vraies fourmis

Les fourmis représentent un univers à elles seules. « *The ants* », la bible des myrmécologues, est un pavé de 780 pages [HOL 90], de nombreuses publications sont

Chapitre rédigé par Alain Lenoir et Nicolas Monmarché.

accessibles au grand public [KEL 06, PAS 05] et on retrouve très souvent des illustrations des capacités des fourmis dans les reportages animaliers[1]. Elles occupent en effet [PAS 06] presque tous les habitats terrestres où leur biomasse est considérable. Par exemple, en forêt amazonienne, la biomasse des fourmis est quatre fois supérieure à la biomasse des vertébrés terrestres [PAS 05]. En mars 2008, on recensait 12 307 espèces différentes et il en reste peut-être encore autant à découvrir, surtout en milieu tropical très mal connu. Elles ont inventé l'élevage des pucerons (c'est un vrai bétail qui est mis à l'abri en hiver, transporté lors des déménagements, et défendu contre les prédateurs), la culture de champignons, l'esclavage (il s'agit d'esclavagisme interspécifique : une espèce comme la fourmi amazone *Polyergus* utilise une autre espèce pour subvenir à ses propres besoins), la vie en parasite, la guerre, la division du travail, le nomadisme, des antibiotiques et des poisons. Elles mesurent de moins d'un millimètre à 4 cm. Leurs colonies comportent de quelques individus à plusieurs dizaines de millions (plus de 20 millions chez les fourmis légionnaires nomades d'Afrique tropicale, plus de 300 millions dans les supercolonies de fourmis rousses des bois au Japon ou dans le Jura suisse, avec un million de reines et 45 000 nids sur quelques km^2). Elles nichent dans le sol jusqu'en haut des plus hauts arbres de la canopée. Comment expliquer ce succès ? Justement par leur vie sociale très développée dont on va présenter les principales caractéristiques, mais d'abord comment reconnaître une fourmi ?

Ce que l'on appelle fourmi est en général une ouvrière. Il s'agit d'un insecte, caractérisé par un exosquelette (littéralement « squelette extérieur », c'est-à-dire carapace, aussi appelée cuticule), six pattes, des yeux et deux antennes. Elle a un corps en trois parties : la tête, le thorax et l'abdomen (figure 1.1). Tout cela fait que le faciès fourmi est très caractéristique. Tout le monde reconnaît facilement une fourmi de loin, même si elle n'a que quatre pattes comme dans les dessins animés de Walt Disney. L'anatomie de la fourmi se caractérise classiquement par un tube digestif, un appareil circulatoire, un système nerveux et de nombreuses glandes, qui sont de véritables usines chimiques. Le tube digestif comporte une poche spéciale, le jabot, où est stockée la nourriture liquide. L'ouvrière est stérile. La reine est le seul individu pondeur, il peut d'ailleurs y avoir plusieurs reines au sein d'une même colonie (chez les fourmis envahissantes comme la fourmi d'Argentine sur la Côte d'Azur, il y a des milliers de reines). Elles sont en général beaucoup plus grosses que les ouvrières et ce sont souvent de véritables machines à pondre (deux œufs à la minute, en période de ponte, chez les fourmis légionnaires).

Lorsque l'on observe une fourmilière ou des fourmis dans un nid artificiel en laboratoire, on est surpris par la richesse du répertoire comportemental de ces animaux.

1. Cet engouement s'est également traduit par le succès considérable des livres de B. Werber, livres dont le point de départ est souvent un fait réel, mais très vite l'auteur fait de la science-fiction et attribue aux fourmis des sentiments humains, ce qui, évidemment, n'est pas pertinent pour l'étude scientifique des fourmis, qu'elles soient réelles ou artificielles.

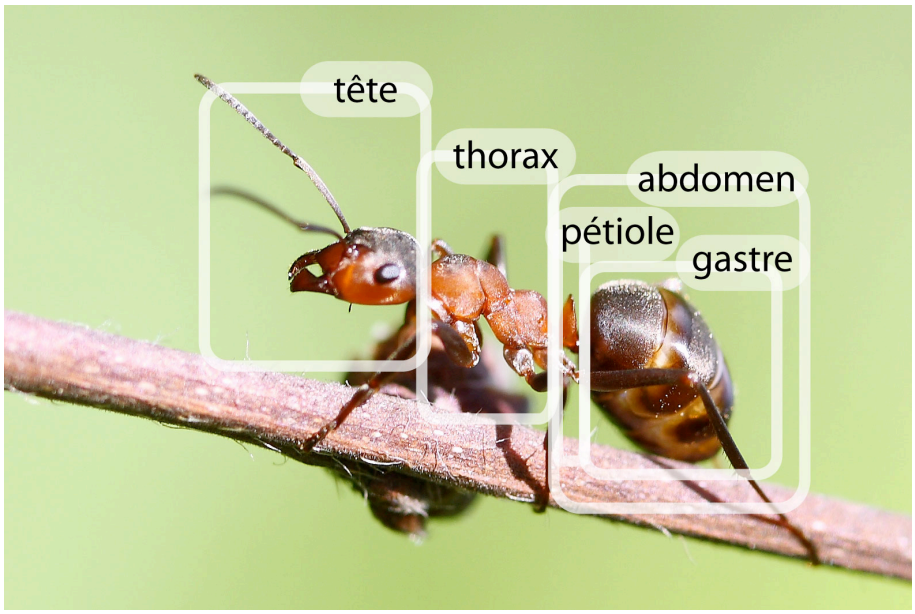


Figure 1.1. *Quelques renseignements anatomiques sur la fourmi (ici, formica polyctena) – photo N. Monmarché*

Par exemple, quand deux fourmis se rencontrent, elles s'inspectent à l'aide de leurs antennes. Elles peuvent ainsi se reconnaître comme familières ou étrangères et avoir le comportement adapté (voir figure 1.2). Les relations interindividuelles sont très développées, elles aiment se lécher réciproquement, comme les singes. La fourmi va aussi passer beaucoup de temps à sa toilette personnelle ; elle se nettoie les antennes et tout le corps avec ses pattes antérieures. Une autre particularité des insectes sociaux est le développement extraordinaire des échanges de nourriture de bouche à bouche. C'est ce que l'on appelle la trophallaxie. C'est un moyen simple de transporter les aliments liquides. En effet, la fourmi récolteuse stocke la nourriture liquide (par exemple le miellat de pucerons) dans son jabot. De retour au nid, elle peut la régurgiter (« la vomir ») à ses congénères restées dans le nid qui vont l'utiliser pour leur propre alimentation et aussi pour élever les larves. C'est pour cette raison que l'on qualifie le jabot d'estomac social.

Dans la colonie, les ouvrières n'ont pas toutes les mêmes occupations. Il existe une véritable répartition des tâches, ou division du travail. Elle est plus ou moins marquée selon les espèces, mais elle existe partout. Schématiquement, les jeunes ouvrières s'occupent du couvain et de la reine, puis progressivement des autres tâches à l'intérieur du nid (construction, nettoyage, défense, transferts de nourriture), et enfin sortent pour récolter la nourriture qu'elles rapportent au nid. En fait ce système

est très souple, il permet de s'adapter aux besoins de la colonie, par exemple si le taux de prédation des fourrageuses est très élevé, les jeunes fourmis vont sortir plus vite. Chez certaines espèces, il existe des castes différenciées anatomiquement, avec des ouvrières de grande taille (soldats) spécialisées dans la défense du nid. Cette organisation est l'une des causes du succès écologique des fourmis (toutes les fourmis envahissantes pratiquent ce recrutement de masse), cela permet par exemple d'exploiter rapidement une source de nourriture ou de mobiliser un grand nombre d'individus pour défendre le nid (voir figure 1.3).

Figure 1.2. *Rencontre de deux fourmis (ici, formica polyctena) – photo N. Monmarché*

Une telle organisation sociale très sophistiquée ne pourrait fonctionner sans des mécanismes assurant la communication entre individus. Les fourmis utilisent trois types de communication : tactile par les antennes (voir figure 1.2), sonore par des stridulations, et chimique par de nombreuses odeurs. La vision est en général peu importante. La communication tactile intervient surtout pour inviter une congénère à suivre la piste vers une source de nourriture : la recruteuse tambourine la tête de l'autre fourmi pour la stimuler, on parle de comportement d'invitation. Lors de la trophallaxie, on assiste à un véritable ballet entre les antennes des deux fourmis : celle qui reçoit sollicite l'autre en lui balayant la tête ; la donneuse répond de la même manière. Elles s'informent sur leur état de satiété.

Figure 1.3. *Défendre la colonie est aussi une tâche collective (ici, formica rufa) – photo N. Monmarché*

C'est chez les insectes qu'ont été découvertes, dans les années 1960, les substances chimiques (phéromones) qui permettent à deux individus de la même espèce de communiquer.

On dispose à l'heure actuelle de chromatographes en phase gazeuse couplés à un spectromètre de masse qui permettent d'analyser et d'identifier les substances constitutives d'une phéromone à des concentrations très faibles, de l'ordre du nanogramme (mais soyons modestes, les antennes des insectes ont une sensibilité colossale, à des concentrations que l'on ne sait pas détecter actuellement).

Ces phéromones sont sécrétées par de très nombreuses glandes situées partout dans le corps qui est une véritable usine chimique : on connaît déjà 39 glandes chez les fourmis (voir figure 1.4) et plusieurs centaines de substances de catégories chimiques très variées. Elles interviennent dans toutes les activités de la colonie : l'agrégation, le marquage du territoire et de l'entrée du nid, la formation de pistes pour exploiter une source de nourriture ou déménager (changer de nid), le recrutement de congénères pour défendre la colonie (alarme et défense), l'appel sexuel, la reconnaissance de la reine, des autres membres de la colonie et même des cadavres.

Une autre particularité importante des sociétés d'insectes est qu'elles sont fermées : les individus étrangers à la colonie sont rejetés et parfois tués. On peut assister par exemple au printemps à de véritables « guerres » entre deux colonies voisines qui

essaient de repousser leurs frontières au détriment de l'autre, avec de nombreux cadavres entre les deux zones. Certaines espèces tropicales comme les fourmis fileuses œcophylles vivent sur l'arbre qu'elles monopolisent complètement et défendent farouchement contre tout intrus.

La fermeture des sociétés d'insectes s'inscrit dans le cadre théorique de la sélection de parentèle, élaboré par W.D. Hamilton [HAM 64a, HAM 64b]. En effet, selon la théorie de l'évolution darwinienne, un animal élève sa propre descendance dans le but de transmettre ses gènes. Or, chez les fourmis, les ouvrières stériles ne se reproduisent pas et élèvent leurs sœurs. On les qualifie d'altruistes. Mais cet altruisme est intéressé, elles transmettent indirectement leurs gènes. Alors, elles ne doivent pas élever des larves avec lesquelles elles n'ont pas de lien de parenté. Il faut que la colonie reste une entité fermée au niveau génétique. Même s'il y a de nombreux cas particuliers et diverses critiques, à l'heure actuelle, la théorie de sélection de parentèle reste un modèle heuristique très intéressant.

Par quel mécanisme les fourmis se reconnaissent-elles quand elles se rencontrent et qu'elles échangent des battements antennaires ? Comme l'on pouvait s'en douter, elles se reconnaissent à l'odeur. L'odeur des fourmis est formée de substances très peu volatiles qui sont portées par la carapace de la fourmi et perçues par les antennes. Il s'agit essentiellement d'hydrocarbures qui sont des molécules simples dérivées de lipides, des chaînes de carbone et d'hydrogène. Chaque colonie a un bouquet « d'odeurs » qui la caractérise, composé souvent d'une cinquantaine, voire même d'une centaine de molécules différentes : c'est l'odeur coloniale. C'est un peu un abus de langage de parler d'odeur car cela se rapproche plutôt de la gustation, mais c'est entré dans le langage des myrmécologues. Ces substances sont fabriquées dans le corps par des cellules spécialisées. Chaque fourmi fabrique ses propres hydrocarbures en fonction de son patrimoine génétique, exactement comme les mammifères sécrètent des odeurs liées au complexe majeur d'histocompatibilité (CMH). Les hydrocarbures sont excrétés sur la cuticule, mais sont aussi stockés dans une glande de la tête, la glande post-pharyngienne, dont le contenu est versé avec les liquides régurgités pendant la trophallaxie. Ainsi l'odeur coloniale est-elle échangée constamment entre individus. Si l'on ajoute les léchages interindividuels très fréquents, on constate un brassage permanent et très important des odeurs individuelles. Cela aboutit à une odeur commune partagée entre tous les individus et caractéristique de la colonie. On parle de *gestalt* [DAH 98].

On peut se demander ensuite comment la fourmi s'identifie à sa colonie pour être capable de discriminer les autres. La jeune fourmi apprend, quand elle sort de son cocon, à reconnaître l'odeur de son nid, donc de ses sœurs et elle va mémoriser cette odeur pour toute sa vie. Cet apprentissage commence d'ailleurs parfois même à l'état larvaire. Bien sûr, il reste une flexibilité puisque l'odeur de la colonie change un peu en fonction de l'alimentation, des matériaux du nid ou de la composition en individus. La fourmi doit en permanence ajuster l'odeur qu'elle porte avec celle des autres (*labels*),

mais aussi l'image mentale qu'elle mémorise dans son cerveau que l'on appelle *template*. En fait, l'évolution a imaginé un processus très économique pour permettre la reconnaissance des congénères : il suffit d'apprendre et mémoriser en toute confiance l'odeur des congénères du nid [LEN 99] ! C'est ainsi que les fourmis amazones *Polyergus* esclavagistes vont faire des razzias de cocons dans les colonies de l'espèce hôte. Quand les jeunes ouvrières éclosent dans la colonie esclavagiste, elles s'imprègnent de l'odeur de leur colonie d'adoption et vont considérer les amazones comme leurs sœurs et travailler à leur service.

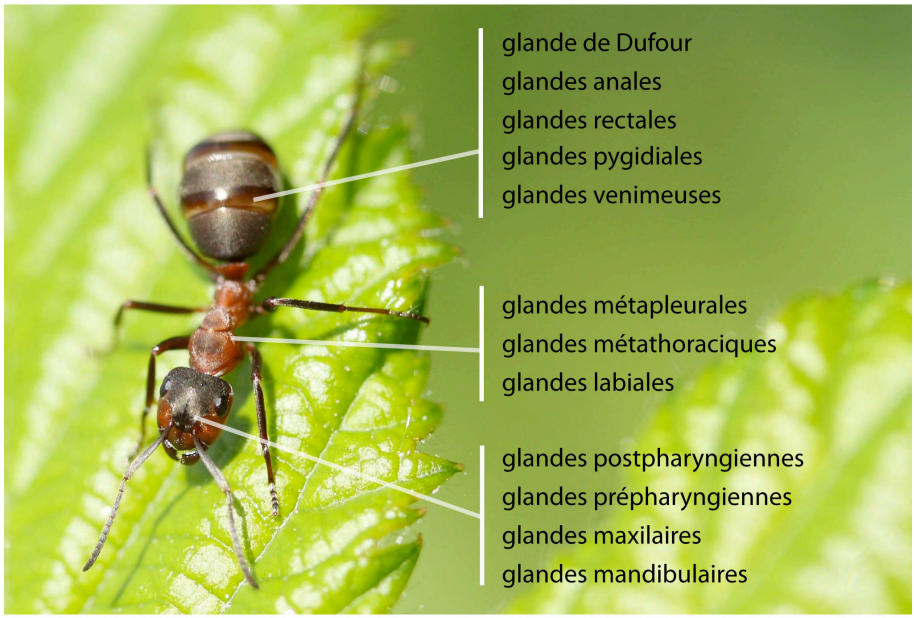


Figure 1.4. *L'importance des glandes chez les fourmis – photo N. Monmarché*

1.2. L'intelligence collective et l'auto-organisation

Lorsque l'on regarde un nid de fourmis, comme on l'a dit plus haut, cela grouille dans tous les sens ! A première vue, les comportements individuels sont fantaisistes et semblent indépendants. C'est une observation courante que de voir une fourmi tirer la charge dans le mauvais sens et il est impossible de prédire exactement ce qu'un individu va faire à un moment donné. Pourtant, ce fouillis apparent est remarquablement efficace : même si certains individus échouent, la tâche est accomplie, la grosse proie rentrée au nid, la brindille insérée correctement dans le dôme et la colonie fonctionne de manière prévisible. Comment expliquer cela alors qu'il n'existe pas d'organe central donnant des ordres ? En effet, la reine ne joue pas ce rôle, il n'y a ni *leader* ni

despote. On peut marquer individuellement des ouvrières avec une pastille numérotée (ou des tâches de couleur), suivre leur activité dans un nid artificiel et les soumettre à diverses tâches. On s'aperçoit que certaines sont hyperactives, d'autres peu actives ou même totalement inactives, ces dernières représentent même la majorité. Tout se passe comme s'il y avait un volant important d'ouvrières inactives en attente de travail. On constate que la fourmi adapte son comportement aux besoins de la colonie, qui peuvent varier, et aussi en fonction de l'environnement. En dotant ainsi la fourmi d'un simple comportement d'adaptation de ses seuils de réponse à différents stimuli de son quotidien, celle-ci se spécialise, c'est-à-dire répond plus rapidement à un stimulus donné et accomplit l'action correspondante plus efficacement, et ceci d'autant plus que la colonie s'agrandit. Plus la colonie est importante, plus les besoins sont importants mais plus les fourmis se spécialisent et accomplissent certaines tâches très efficacement : ceci peut expliquer que, la taille de la colonie augmentant, la proportion de fourmis inactives augmente également. L'ensemble devient ainsi de plus en plus flexible et robuste, s'il faut beaucoup d'individus pour exploiter une source de nourriture importante, les réserves sont disponibles.

A partir des données d'observation, on peut construire des modèles pour essayer de simuler le fonctionnement des sociétés. Ce travail permet d'approfondir nos connaissances fondamentales sur les sociétés de fourmis. Un pas décisif a été franchi avec les modèles récents inspirés de la théorie de l'auto-organisation. L'auto-organisation correspond à un ensemble de processus décrits par I. Prigogine (Prix Nobel de chimie en 1977), au cours desquels des structures émergent au niveau collectif, à partir d'une multitude d'interactions entre individus, sans être codées explicitement au niveau individuel ([NIC 77] et voir par exemple [BON 97, CAM 01]).

Le concept d'auto-organisation permet d'expliquer, du moins de prédire, de nombreux phénomènes macroscopiques en définissant des règles comportementales au niveau microscopique. Dans la perspective scientifique habituelle, ces règles sont sélectionnées en respectant le principe de minimalité (le système le plus réduit/simple qui peut induire ou expliquer le phénomène est plébiscité). Un certain nombre de propriétés générales ont été dégagées de ces systèmes auto-organisés pour qu'il y ait des chances d'observer l'émergence d'une structure stable :

– la présence de deux types de renforcements : positif (*positive feedback*), qui induit l'amplification d'un phénomène, et négatif (*negative feedback*), qui contrebalance l'amplification et assure donc une stabilisation du phénomène ;

– l'amplification des fluctuations : les fluctuations sont la composante aléatoire du phénomène et l'amplification de certaines manifestations repose sur les mécanismes de renforcement ;

– la multitude des interactions : plus les interactions sont nombreuses, plus la stabilité des structures construites est forte.

Le phénomène observé macroscopiquement prend la forme d'une structure présentant une certaine stabilité spatio-temporelle, c'est-à-dire observable dans l'espace et/ou le temps, dont les manifestations chez les fourmis sont, par exemple, la construction de pistes, la construction du nid (creusement de galeries). Ces structures apparaissent dans un milieu plus ou moins homogène (la surface de la terre ou le sous-sol) et plusieurs états stables peuvent coexister (plusieurs chemins vers différentes sources de nourriture). Ces phénomènes auto-organisés présentent aussi parfois des bifurcations dans le comportement observé, notamment lorsque certains paramètres environnementaux sont modifiés : par exemple, l'évaporation des phéromones devenant plus importante à cause des conditions climatiques, les chemins qui ont été construits et qui représentent des structures stables peuvent subitement se disloquer.

La simplicité des entités considérées et participant à l'auto-organisation ne doit cependant pas dissimuler que la fourmi reste à son échelle une merveille de miniaturisation et d'autonomie (notamment lorsque l'on envisage de construire un robot présentant les mêmes capacités, voir chapitre 2 du volume 2 [URB 09]). En effet, les fourmis ne se comportent pas comme de simples molécules [DET 06] et leurs capacités individuelles, notamment cognitives, ne sont pas à négliger pour expliquer des comportements évolués, qu'ils soient collectifs ou individuels.

La suite de cette section donne quelques illustrations de modèles de comportement.

1.2.1. *Les pistes*

Comme on l'a vu, une fourmi qui découvre une source de nourriture importante dépose une trace de phéromone en rentrant au nid. La règle dans ce cas est très simple : les fourmis suivent la piste chimique et l'on observe une croissance exponentielle du nombre de fourmis sur cette source. D'un comportement simple individuel (le dépôt d'une gouttelette de phéromone et l'attirance pour un chemin marqué par de la phéromone), émerge un comportement collectif (le suivi de piste et le recrutement de masse). S'il y a plusieurs chemins possibles pour accéder à la nourriture, le chemin le plus court sera alors choisi collectivement sans aucune décision individuelle et perception individuelle de la distance, tout simplement parce que la piste la plus courte sera renforcée plus vite, comme cela a été observé chez la fourmi d'Argentine [DEN 90, GOS 89]. C'est un processus autocatalytique, les fourmis rentrant au nid choisissent et renforcent toujours davantage la piste la plus marquée (renforcement positif) jusqu'à ce que la source s'épuise ou que l'accès à la nourriture devienne trop embouteillé (renforcement négatif). Ces expériences sont maintenant classiques et sont décrites dans des manuels comme celui de M. Dorigo et T. Stützle [DOR 04] (voir également les chapitres 2 et 3 du présent ouvrage).

Une recherche récente montre aussi que les fourmis sont capables d'estimer le trafic sur une piste : quand la densité devient trop importante, elles choisissent le

chemin le moins embouteillé. Cela a été montré chez la petite fourmi noire des jardins, *Lasius niger*, par A. Dussutour [DUS 04]. Cette fourmi est capable de recrutement de masse et forme des pistes plus ou moins permanentes. Dès que la densité sur la piste atteint un certain seuil, une deuxième piste est formée avant que le trafic ne soit affecté, ce qui garantit une efficacité optimale.

Elles sont aussi capables d'ajuster le trafic à la largeur de la piste. Ce phénomène a été étudié, toujours chez *Lasius niger*, où l'on peut observer sur les pistes un trafic très important dans les deux sens. On a examiné les effets d'un goulot d'étranglement sur la piste. A. Dussutour [DUS 05] a réuni un nid de *Lasius* à une aire de fourragement à l'aide d'un pont. Deux types de pont ont été utilisés : avec ou sans goulot. Les fourmis arrivent à réguler leur trafic : avec les deux types de pont, le volume de trafic et la quantité totale de nourriture rapportée au nid restent identiques. Cela est possible, quand il y a un goulot d'étranglement, par une réorganisation temporelle du flux : quand la largeur de la voie décroît, les fourmis se répartissent en groupes alternant dans les deux sens (aller ou retour) comme s'il y avait un feu rouge virtuel. Cela limite le nombre de rencontres en face à face entre deux fourmis allant en sens contraire et permet la même durée de trajet. On peut imaginer que ce type de problème se pose sans arrêt dans les galeries des nids de fourmis ou de termites.

Ces capacités des fourmis réelles à résoudre des problèmes, comme trouver le meilleur chemin pour arriver à une source de nourriture, à l'aide de phéromones, a inspiré des algorithmes nombreux, par exemple à des chimistes pour élaborer des polymères [MAR 08]. Cependant, les pistes ne sont pas utiles qu'à la recherche de nourriture, par exemple chez les *Magnans* on observe une organisation particulière quand le nid est déplacé : les soldats surveillent et protègent la piste empruntée par les ouvrières (voir figure 1.5). Ainsi, la recherche d'un nouveau nid et le déplacement des colonies est un autre modèle pour comprendre les interactions entre décisions individuelles et collectives. N. Franks et son équipe à Bristol étudient depuis plusieurs années une toute petite fourmi qui vit dans les fissures de rochers (*Temnothorax albipennis*). Si l'on donne le choix entre deux sites possibles, l'un proche mais de mauvaise qualité, l'autre plus éloigné et sur la route du premier mais meilleur, le plus souvent les colonies commencent à émigrer simultanément vers les deux nids. Mais, un peu plus tard, elles redirigent tout le trafic exclusivement vers le meilleur nid. Un modèle a montré que ce choix minimise l'exposition aux risques liés au déménagement à l'air libre [FRA 08].

On peut enfin noter que certains travaux [ALT 05] ont montré qu'en situation de panique, les fourmis pouvaient perdre leur aptitude à distribuer équitablement la charge de trafic et ont tendance à s'engouffrer vers la sortie la plus engorgée (à l'image d'une foule entraînée par une seule sortie).

Figure 1.5. *Colonne de Magnans – photo A. Lenoir*

1.2.2. *Tri d'objets*

Une application particulière du tri d'objets est le transport et l'agrégation des cadavres, qui aboutissent à l'élaboration des fameux « cimetières » de fourmis observés par les anciens naturalistes qui n'hésitaient pas à dessiner les cadavres bien alignés. Les fourmis rejettent les cadavres hors du nid afin de réduire les risques d'infection dans la colonie et les regroupent dans un endroit éloigné. Lorsque l'on disperse au hasard des cadavres dans une arène expérimentale, en quelques heures, les fourmis vont les regrouper en petits tas. Il est possible de modéliser ce comportement en attribuant à chaque fourmi une probabilité d'exécuter une action, tout d'abord de saisir le cadavre rencontré, puis de le déposer dès qu'elle rencontre un autre cadavre. Cependant, il ne faut pas prendre un cadavre déjà déposé sur un tas. Pour résoudre ce problème, on introduit une nouvelle règle : plus le tas est gros, moins la fourmi aura tendance à prendre un cadavre et plus elle le déposera facilement. Le modèle permet de prédire qu'il existe une densité critique de cadavres en deçà de laquelle l'agrégation n'aura pas lieu : les tas n'ont pas le temps de se former [THE 02].

Un autre exemple de tri est le tri du couvain (figure 1.6). En effet, dans le nid, les divers éléments de couvain ne sont pas mélangés en vrac mais répartis dans des

chambres différentes selon leur âge. C'est ainsi que les œufs et petites larves sont entassés au centre du nid, les larves plus grosses dans des chambres périphériques. Les cocons sont régulièrement montés à la surface du nid pour être chauffés au soleil, ce qui accélère leur développement. Dans le modèle correspondant, les fourmis prennent et déposent tout élément de couvain en fonction du nombre d'items présents et de leur catégorie [DEN 91].

Figure 1.6. *Le couvain chez Lasius niger : les cocons de sexués et d'ouvrières sont séparés (ici sur la photo) et se trouvent dans des chambres différentes des larves et œufs – photo N. Monmarché*

Le chapitre 1 du volume 2 [HAM 09] donne des détails sur les modèles construits à partir de ces notions de tri d'objets par les fourmis, et notamment leurs utilisations en classification de données.

1.2.3. *Activités de creusement du nid*

Dans le cas du creusement du nid, les ouvrières dégagent les chambres du nid et rejettent les grains de sable en dehors de l'entrée du nid. C'est ce que l'on peut observer après une pluie quand la terre est meuble et plus facile à creuser. Le dépôt de centaines de charges conduit à la formation d'un cratère régulier. Comment cela est-il possible ? E.J.H. Robinson *et al.* ont étudié ce problème [ROB 07] : des règles simples permettent de le résoudre. La fourmi rejette la particule préférentiellement

au sommet du cratère vers la pente la plus douce depuis l'entrée du nid. D'autres règles simples, comme la mémoire de la direction des sorties précédentes, permettent d'affiner le modèle et un comportement régulier et prévisible émerge (voir figure 1.7).

Figure 1.7. *Cratères creusés après une pluie (Camponotus, Maroc) – photo A.Lenoir*

1.2.4. *Agrégation d'individus*

L'agrégation est une caractéristique de tous les animaux sociaux, mais elle est fondamentale pour les insectes sociaux qui ne peuvent survivre longtemps hors de leur colonie. Toutes les fourmis présentent, à divers degrés, cette attraction mutuelle qui permet l'agrégation [DEP 08].

La distribution spatiale des individus dans la colonie est déterminante pour l'efficacité de celle-ci, il faut être au bon endroit au bon moment. Les individus vont se regrouper en fonction des tâches comme le couvain ou la reine à soigner (voir la revue de [AND 02]). L'agrégation peut s'effectuer en fonction de repères physiques ou physico-chimiques dans le nid (par exemple, taille des chambres, humidité, luminosité, etc.). Des critères chimiques peuvent aussi intervenir. C'est ainsi que l'on a montré chez la blatte (qui n'est pas sociale comme les fourmis, mais grégaire, ce qui peut être considéré comme une forme basique de la socialité) que les individus reconnaissent l'odeur de leurs congénères et s'en servent pour se regrouper dans des abris

[RIV 98]. En l'occurrence, les odeurs sont aussi des hydrocarbures, comme chez les fourmis. Par exemple, chez *Lasius niger*, des traces de marquage colonial sur le sol favorisent l'agrégation [DEP 04a, DEP 04b]. Ce système de reconnaissance est donc apparu tôt dans l'évolution. On a même fabriqué des robots imprégnés de l'odeur des blattes qui arrivent à induire un regroupement des individus, les blattes sont leurrées et considèrent les robots comme des congénères [HAL 07].

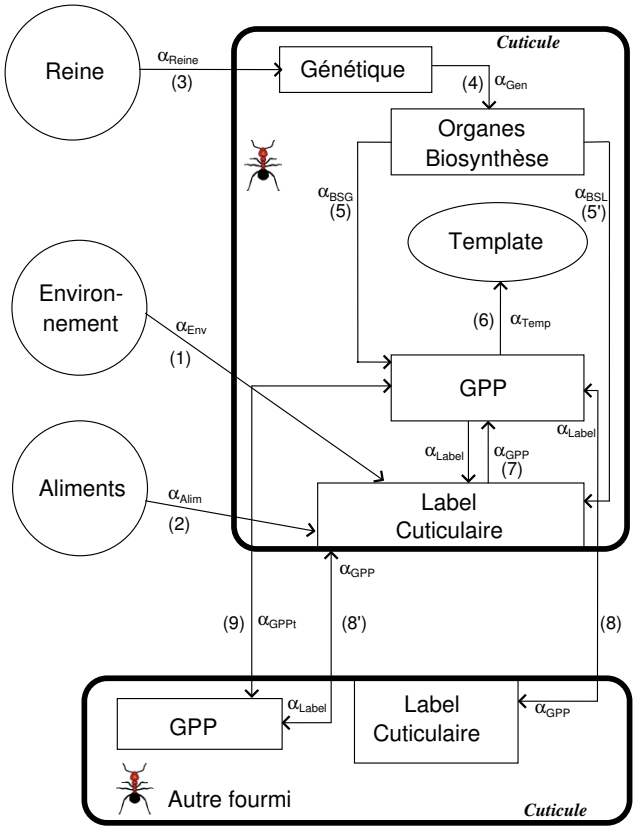


Figure 1.8. *Schéma récapitulatif du modèle d'échange des odeurs*

1.2.5. *Modélisation de l'odeur coloniale*

N. Labroche [LAB 02] a développé un simulateur virtuel de l'odeur coloniale des fourmis (*Virtual Ants Laboratory Simulator* : VALS) dont le schéma est présenté figure 1.8. Il représente une fourmi (encadré en haut à droite) avec les organes qui jouent un rôle dans la fermeture coloniale. Les interactions de la fourmi avec son environnement sont également représentées : les relations sociales avec les autres individus de

la colonie (une ouvrière et la reine), l'environnement physique de la fourmi, comme son alimentation ou l'odeur des matériaux du nid. Il a ainsi été possible de simuler l'évolution de l'odeur de colonies en faisant varier le poids respectif de l'environnement (la diète ou les matériaux du nid) et de la production propre de l'individu (facteurs génétiques). Ce modèle a été testé à partir de données expérimentales obtenues chez la fourmi champignonniste *Acromyrmex subterraneus subterraneus*, où la séparation entre groupes induit une divergence d'odeur, divergence accrue ou diminuée en fonction du régime alimentaire. Dans ce cas, la simulation aboutit à des résultats comparables aux données expérimentales.

1.3. Fourmis artificielles

Dans une troisième étape, la fourmi devient complètement virtuelle et n'a presque plus aucune caractéristique, du moins physique, d'une fourmi. C'est un agent artificiel. On a gardé le nom de fourmi par facilité et parce que la fourmi a inspiré les premiers informaticiens qui se sont intéressés à l'intelligence collective. Toutes les fourmis étant sociales, cela simplifie aussi le propos. Enfin, la fourmi reste plus populaire que le termite et son image – même fausse – de travailleuse infatigable correspond à ce que l'on peut attendre d'elle d'un point de vue informatique !

La programmation informatique moderne permet de faire fonctionner en parallèle divers « agents », par exemple des agents fourmis qui interagissent entre eux et avec leur environnement. Chaque agent répond en fonction d'une ou d'un petit nombre de règles simples, sans qu'il soit nécessaire de faire appel à des capacités cognitives élaborées. Les fourmis modélisées sont alors des agents réactifs et on parle alors d'intelligence en essaim [BON 99, BON 00]. Le terme « intelligence en essaim » a d'abord été utilisé dans le contexte des systèmes robotiques cellulaires (fin des années 1980), mais s'applique parfaitement à une gamme plus large de systèmes artificiels. La manifestation d'une intelligence collective s'observe par l'émergence ou l'apparition de structures temporelles et/ou spatiales issues d'interactions multiples et répétées, directes ou indirectes, entre des individus appartenant à une même colonie ou un même groupe. Cette définition n'est pas sans rappeler les phénomènes d'auto-organisation évoqués dans la section précédente. La notion d'intelligence collective intègre en plus l'adaptation du comportement dans les conditions environnementales données : l'intelligence peut être considérée comme une capacité à s'adapter et à survivre dans son environnement. Et, de ce point de vue, les fourmis sont bien placées car on peut trouver des fourmis à peu près partout sur la surface des terres émergées.

La fourmi artificielle peut prendre la forme d'un processus informatique, qui possède donc ses propres zones de mémoire et ses instructions de fonctionnement. L'implantation informatique des différents modèles précédemment présentés amène à des considérations nous éloignant des contraintes naturelles : discrétisation du temps et de l'espace, synchronisme/asynchronisme des actions. Malgré ces efforts, l'intérêt de la

modélisation et la simulation informatique pour le biologiste est évident : cela permet de tester des modèles facilement (par rapport à l'effort expérimental de manipuler de vraies fourmis), de chercher une explication à des phénomènes émergents ou de tester la capacité de prédiction d'un modèle.

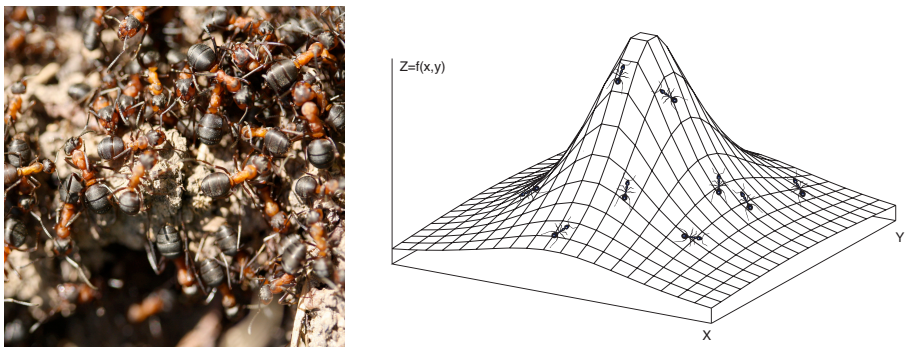


Figure 1.9. *Les fourmis réelles et artificielles partagent-elle le même espace de recherche ?*

Quand il s'agit de résoudre des problématiques « humaines », l'objet de la suite de cet ouvrage, la fourmi artificielle, va perdre encore de ce qui la rattachait à son modèle naturel. Par exemple, si cela suffit de considérer la fourmi comme un point dans un espace de recherche auquel on s'intéresse, on s'abstiendra bien de lui fournir des pattes, un thorax, des antennes, etc. Le lien entre le point et la vraie fourmi ne sera alors peut-être plus qu'une lointaine similitude dans la stratégie de déplacement du point et le modèle de comportement de la fourmi dans son environnement naturel.

Dans les chapitres de ce volume, mais aussi du volume 2, certains des modèles exposés précédemment seront repris et abordés dans le cadre d'une problématique précise : l'utilisation des pistes pour exploiter des sources de nourriture dans les problèmes modélisés par un graphe (chapitres 2 et 3 du volume 1), le tri d'objet et la modélisation de l'odeur coloniale dans les problèmes de classification (chapitre 1 du volume 2 [HAM 09]).

Enfin, nous avons commencé ce chapitre par le constat de la suprématie des fourmis dans les environnements naturels et nous pourrions tenter la même analyse dans le domaine des fourmis artificielles. C'est ainsi que, si l'on observe le nombre d'articles publiés dans des journaux ou conférences scientifiques dans les domaines de l'informatique, la robotique ou des mathématiques appliquées, on constate, même avec un relevé incomplet, une augmentation significative du nombre de publications abordant

la notion de fourmi artificielle (voir la figure 1.10, qui donne l'évolution du nombre de publications scientifiques abordant le thème des fourmis artificielles[2]).

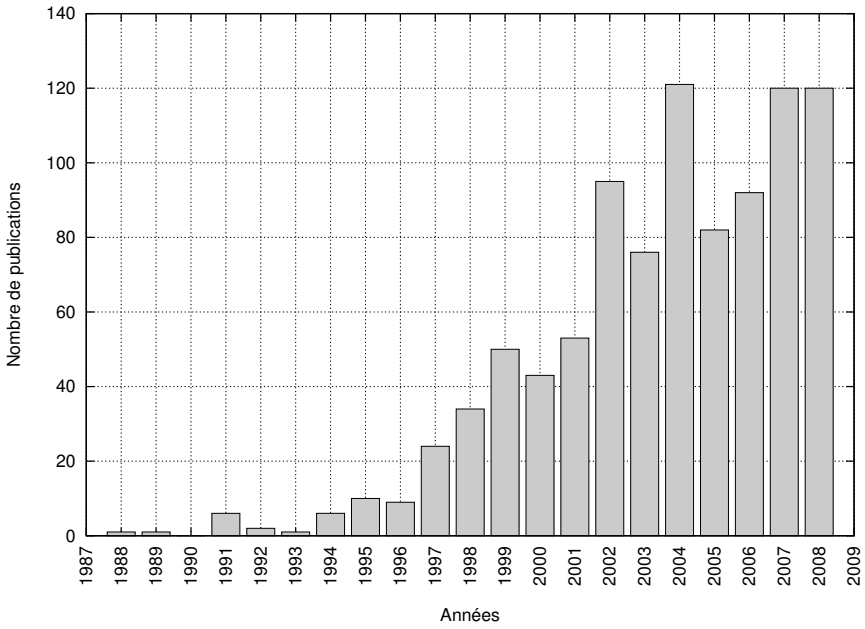


Figure 1.10. *Evolution du nombre de publications repérées qui abordent le thème des fourmis artificielles*

1.4. Bibliographie

[ALT 05] Altshuler E., Ramos O., Núñez Y., Fernández J., Batista-Leyva A. J., Noda C., « Symmetry Breaking in Escaping Ants », *The American Naturalist*, vol. 166, n° 6, p. 643–649, 2005.

[AND 02] Anderson C., Theraulaz G., Deneubourg J.-L., « Self-assemblages in insect societies », *Insectes Sociaux*, vol. 49, p. 99–110, 2002.

[BON 97] Bonabeau E., Theraulaz G., Deneubourg J.-L., Aron S., Camazine S., « Self-organization in social insects », *Trends in Ecology and Evolution*, vol. 12, p. 188–193, 1997.

[BON 99] Bonabeau E., Dorigo M., Theraulaz G., *Swarm Intelligence*, Oxford University Press, New York, 1999.

[BON 00] Bonabeau E., Théraulaz G., « L'intelligence en essaim », *Pour la Science*, p. 66–73, 2000.

2. http://www.hant.li.univ-tours.fr/artantbib/artantbib.php.

[CAM 01] Camazine S., Deneubourg J.-L., Franks N. R., Sneyd J., Theraulaz G., Bonabeau E., *Self-Organization in Biological Systems*, Princeton University Press, Princeton, New Jersey, 2001.

[DAH 98] Dahbi A., Jaisson P., Lenoir A., Hefetz A., « Comment les fourmis partagent leur odeur », *La Recherche*, p. 32–34, 1998.

[DEN 90] Deneubourg J.-L., Aron S., Goss S., Pasteels J. M., « The self-organizing exploratory pattern of the Argentine ant », *Journal of Insect Behavior*, vol. 3, p. 159–168, 1990.

[DEN 91] Deneubourg J.-L., Goss S., Franks N., Sendova Franks A., Detrain C., Chrétien L., « The dynamics of collective sorting : robot-like ants and ant-like robots », J. A. Meyer, E. O. Wilson (dir.), *Simulations of animal behavior : from animals to animals*, p. 356–365, Cambridge University Press. Cambridge, MA, 1991.

[DEP 04a] Depickère S., Fresneau D., Deneubourg J.-L., « A basis for spatial and social patterns in ant species : dynamics and mechanisms of aggregation », *Journal of Insect Behavior*, vol. 17, p. 81–97, 2004.

[DEP 04b] Depickère S., Fresneau D., Detrain C., Deneubourg J.-L., « Marking as a decision factor in the choice of new nesting site in Lasius niger », *Insectes Sociaux*, vol. 51, p. 243–246, 2004.

[DEP 08] Depickère S., Ramirez Avila G.-M., Fresneau D., Deneubourg J.-L., « Polymorphism : a weak influence on worker aggregation level in ants », *Ecological Entomology*, vol. 33, p. 225–231, 2008.

[DET 06] Detrain C., Deneubourg J.-L., « Self-organized structures in a superorganism : do ants "behave" like molecules ? », *Physics of Life Reviews*, vol. 3, p. 162–187, 2006.

[DOR 04] Dorigo M., Stützle T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.

[DUS 04] Dussutour A., Fourcassié V., Helbing D., Deneubourg J.-L., « Optimal traffic organization in ants under crowded conditions », *Nature*, vol. 428, p. 70–73, 2004.

[DUS 05] Dussutour A., Deneubourg J.-L., Fourcassié V., « Temporal organization of bidirectional traffic in the ant *Lasius niger* (L.) », *The Journal of Experimental Biology*, vol. 208, p. 2903–2912, 2005.

[FRA 08] Franks N., Hardcastle K. A., Collins S., Smith F. D., Sullivan K. M. E., Robinson E. J. H., Sendova Franks A., « Can ants choose a far-and-away better nets over an in-the-way poor one ? », *Animal Behaviour*, vol. 75, n° 2, p. 323–334, 2008.

[GOS 89] Goss S., Aron S., Deneubourg J.-L., Pasteels J., « Self-organized shortcuts in the Argentine ant », *Naturwissenschaften*, vol. 76, p. 579–581, 1989.

[HAL 07] Halloy J., Sempo G., Caprari G., Rivault C., Asadpour M., Tâche F., Saïd I., Durier V., Canonge S., Amé J. M., Detrain C., Correll N., Martinolli A., Mondada F., Siegwart R., Deneubourg J.-L., « Social integration of robots into groups of cockroaches to control self-organized choices », *Science*, vol. 318, p. 1155–1158, 2007.

[HAM 64a] Hamilton W. D., « The genetical evolution of social behaviour II », *Journal of Theoretical Biology*, vol. 7, p. 17–52, 1964.

[HAM 64b] Hamilton W., « The genetical evolution of social behaviour I », *Journal of Theoretical Biology*, vol. 7, p. 1-16, 1964.

[HAM 09] Hamdi A., Antoine V., Monmarché N., Alimi A., Slimane M., « Fourmis artificielles et classification automatique », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, nouvelles directions pour une intelligence collective*, vol. 2 de *Traité IC2*, chap. 1, Hermès, Paris, 2009.

[HOL 90] Hölldobler B., Wilson E., *The Ants*, Springer Verlag, Berlin, 1990.

[KEL 06] Keller L., Gordon E., *La vie des fourmis*, Odile Jacob, Paris, 2006.

[LAB 02] Labroche N., Richard F.-J., Monmarché N., Lenoir A., Venturini G., « Modelling of the chemical recognition system of ants », C. Hemelrijk (dir.), *International Workshop on Self-Organization and Evolution of Social Behaviour*, p. 283–292, Monte Verità, Ascona, Suisse, septembre8-13 2002.

[LEN 99] Lenoir A., Fresneau D., Errard C., Hefetz A., « Individuality and colonial identity in ants : the emergence of the social reprensentation concept », C. Detrain, J.-L. Deneubourg, J. M. Pasteels (dir.), *Information Processing in Social Insects*, p. 219–237, Birkhäuser Verlag, Bâle, Suisse, 1999.

[MAR 08] Martins B. V. C., Brunetto G., Sato F., Coluci V. R., Galvão D., « Designing conducting polymers using bioinspired ant algorithms », *Chemical Physics Letters*, vol. 453, n° 4-6, p. 290–295, 2008.

[NIC 77] Nicolis S. C. P. I., *Self-organization in non-equilibrium systems*, Wiley and Sons, New York, 1977.

[PAS 05] Passera L., Aron S., *Les fourmis : comportement, organisation sociale et évolution*, Les Presses scientifiques du CNRC, Ottawa, 2005.

[PAS 06] Passera L., *La véritable histoire des fourmis*, Fayard, Paris, 2006.

[RIV 98] Rivault C., Cloarec A., Sreng L., « Cuticular extracts inducing aggregation in the German cockroach, Blatella germanica », *Animal Behaviour*, vol. 55, p. 177–184, 1998.

[ROB 07] Robinson E. J. H., Holcombe M., Ratnieks L. W. F., « The organization of soil disposal by ants », *Animal Behaviour*, vol. 75, p. 1389–1399, 2007.

[THE 02] Theraulaz G., Bonabeau E., Nicolis S. C., Solé R. V., Fourcassié V., Blanco S., Fournier R., Joly J., Fernandez P., Grimal A., Dalle P., Deneubourg J.-L., « Spatial patterns in ant colonies », *Proceedings of the National Academy of Sciences, USA 99*, p. 9645–9649, 2002.

[URB 09] Urbano P., Monmarché N., Gaucher P., « Robotique collective et mobile inspirée par les insectes », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielle, nouvelles directions pour une intelligence collective*, vol. 2 de *Traité IC2*, chap. 2, Hermès, Paris, 2009.

Chapitre 2

Principes généraux de résolution de problèmes combinatoires par colonie de fourmis

2.1. Du carbone au silicium

L'idée d'utiliser les fourmis comme modèle pour la mise au point de méthodes de résolution de problèmes combinatoires n'est pas le fruit d'un processus de génération spontanée, elle s'inscrit dans un mouvement de fond dont les premières manifestations apparaissent au cours des années quarante. Ce mouvement s'est probablement nourri des découvertes et théories des *sciences du vivant* tour à tour victimes et actrices de multiples remises en question, causées à la fois par le poids des dogmes religieux qui lui ont longtemps interdit l'énoncé d'idées trop audacieuses et par la découverte de multiples évidences de l'existence de mécanismes intelligibles permettant d'approcher la compréhension du vivant. Au fil du temps, de nombreux domaines ont trouvé dans le vivant une source inépuisable d'inspiration. Ainsi en est-il de l'informatique. Des automates cellulaires qui ne cessent de nous étonner par leur richesse et leur simplicité à la robotique mobile, en passant par la programmation génétique qui permet à des entités immatérielles de présenter certains caractères du vivant, diverses formes mixant allègrement informatique/mathématique et biologie ont été imaginées par de fertiles esprits auxquels cet ouvrage rend indirectement hommage. Sans qu'il soit possible de tous les nommer, citons pour mémoire J. Von Neumann, J. Conway, M. Minksy, A. Turing, H. Maturana, F. Varela, D. Hofstadter, A. Lindenmayer, R. Dawkins, S. Wolfram ou C. Langton, tous ont contribué à rendre l'édifice plus fort et plus stable. Pourquoi ? Quelle fût la motivation à l'origine de leur travail ? Il semblerait que la réponse soit simplement de comprendre

Chapitre rédigé par Antoine Dutot, Frédéric Guinand, Damien Olivier et Yoann Pigné.

le vivant et dans une certaine mesure d'essayer de reproduire certains phénomènes qui nous apparaissent encore aujourd'hui, sinon miraculeux, époustouflants. Sans doute les premières esquisses étaient-elles des tentatives d'approches vers la compréhension des mécanismes à l'œuvre dans le vivant. Une approche que l'on pourrait qualifier de « modéliser pour comprendre ». Puis, peu à peu, les progrès constants et rapides de l'informatique, de l'automatique et de la robotique, ont fait tomber quelques frontières mentales et permis à de nouveaux conquérants de regarder au-delà de l'horizon en imaginant une *terra incognita* peuplée d'entités animées d'une vie non plus carbonée, mais basée sur le silicium. Si le premier sillon « modéliser pour comprendre » était creusé, le second n'allait pas tarder à apparaître : « modéliser pour reproduire ». Ces deux facettes se confondent avec les motivations qui leur sont afférentes. Des premiers travaux de J.L. Deneubourg qui relèvent essentiellement du « comprendre » aux premières tentatives de mimétisme des comportements myrmécoles[1] marquant l'avènement du « reproduire » peu d'années se sont écoulées, mais depuis le mouvement de fond n'a cessé de s'amplifier pour en arriver à des notions qui relèvent du « comprendre le modèle ». Dans le cadre de la résolution de problèmes d'optimisation combinatoires, le processus général, modéliser pour comprendre le système, reproduire le système par un modèle fonctionnel, comprendre le modèle, a produit le long du chemin de nombreux modèles analytiques ou individus-centrés, tantôt conduits par des informations globales, tantôt contraints par des règles locales. Ce chapitre tente de remettre en perspective quelques principes déduits de l'observation du comportement des colonies de fourmis dans le contexte particulier de l'optimisation.

Pourquoi les fourmis ? Selon les estimations de C.S. Moreau *et al.* [MOR 06] basées sur une étude phylogénétique, les premières espèces de fourmis seraient apparues sur Terre il y a environ 150 millions d'années[2]. Cet âge canonique fait de la famille des *Formicidae* l'une des plus anciennes que la Terre héberge. Les fourmis conservées dans l'ambre montrent une structure anatomique identique aux fourmis contemporaines. Ainsi, depuis plusieurs millions d'années, les fourmis, sans modification anatomique substantielle, ont su s'adapter aux changements, parfois dramatiques, de leur environnement et prospérer au point d'être présentes dans la majorité des écosystèmes contemporains. La question de savoir comment de telles caractéristiques ont pu être développées et conservées reste entière, cependant, l'analyse du comportement des fourmis et l'observation des effets de leurs actions mettent en évidence un ensemble de propriétés remarquables. Les fourmis possèdent en tant que société la capacité de s'auto-organiser, de répartir les tâches de travail, de se reproduire, de se différencier, de s'adapter aux changements locaux et globaux de l'environnement et d'évoluer pour occuper de nouveaux biotopes. Si ces propriétés représentent le Graal

1. Relatif aux fourmis.

2. La fourchette déterminée par les auteurs est comprise entre 132 et 173 millions d'années.

des concepteurs de systèmes informatiques dits autonomes et intelligents, pour le sujet qui nous occupe, une autre caractéristique retient notre attention ; la capacité des fourmis à construire et à maintenir des structures dans leur environnement.

Dans ce chapitre, nous considérons la structure comme l'élément central dans le processus de résolution de problèmes d'optimisation combinatoire par colonies de fourmis. L'approche proposée fait de la structure l'interface entre le problème et la méthode de résolution. Le lien étroit entre problème d'optimisation et structure est mis en évidence dans la section 2.2. En nous appuyant sur un ensemble d'exemples tirés de la littérature et des différents chapitres de cet ouvrage, nous montrons que la solution à un problème donné peut s'exprimer comme une structure dans l'espace de recherche. A l'autre extrémité de la chaîne, les systèmes à base de fourmis artificielles ont la capacité, à la manière des sociétés de fourmis, de construire, de mettre en évidence et de maintenir des structures. Ce point fait l'objet de la section 2.3. Cependant, ces structures ne sont pas optimisées pour un problème particulier. La section 2.4 fait le point sur les éléments additionnels et les différentes techniques qui permettent d'atteindre cet objectif.

2.2. Du problème à la structure

Les graphes constituent un formalisme dont l'expressivité est idéale pour la modélisation de nombreux problèmes d'optimisation. Une solution au problème d'origine, dans ce formalisme, est le plus souvent une structure, c'est-à-dire un ensemble de sommets et/ou d'arêtes correspondant à un objet clairement identifié dans le domaine des graphes. Il peut s'agir d'un parcours, d'un ou plusieurs chemins, d'une partition des sommets, d'une clique, d'un arbre couvrant, etc. Dans les paragraphes qui suivent, nous illustrons cela par quelques exemples tirés de la littérature et des contributions compilées dans cet ouvrage. Cette première partie discute le passage des problèmes aux modèles de graphes et identifie les structures associées aux solutions. Dans un objectif de clarté, la description des problèmes originaux a été souvent simplifiée, c'est pourquoi nous vous invitons à une lecture complémentaire des chapitres correspondant aux exemples extraits des contributions présentées dans ce livre.

2.2.1. *Parcours*

Dans un graphe, un parcours est une suite alternée de sommets et d'arêtes $P = (s_0, a_0, s_1, a_1, s_2, a_2, \ldots, s_p, a_p, s_{p+1})$, telle que $a_i = (s_i, s_{i+1})$. Dans un parcours, un sommet et/ou une arête peut apparaître à plusieurs reprises au contraire d'un chemin qui ne peut contenir qu'une seule occurrence d'un sommet ou d'une arête.

Pour illustrer notre propos, le premier problème que nous considérons est un problème d'optimisation qui se pose dans l'industrie automobile. Dans une usine de production de voitures, chaque véhicule est caractérisé par un ensemble d'options dont il

sera pourvu à l'issue de l'assemblage. Chaque option est montée par un poste dédié qui possède une certaine capacité de traitement exprimée par le *ratio* entre le nombre de véhicules qu'il peut traiter et le nombre de véhicules qui défilent sur la ligne de production. Sur la base d'un ensemble de commandes, différentes classes (basées sur des ensembles d'options) de véhicules sont définies. Le problème consiste à ordonnancer les véhicules en fonction de leur classe afin de ne pas dépasser les capacités des postes. Lorsqu'une sous-séquence excède la capacité d'un poste ceci génère un conflit. L'objectif est d'en minimiser le nombre. Une solution est une séquence de classes puisque les véhicules qui appartiennent à une même classe sont équivalents du point de vue des postes. La présentation du problème d'origine, avec en particulier la description des contraintes qui régissent les conditions de production, ainsi que les réponses qui peuvent y être apportées, font l'objet d'un chapitre de cet ouvrage [GAG 09]. Les auteurs proposent de modéliser ce problème à l'aide d'un *graphe complet* dont les sommets sont les classes des véhicules et dont les arêtes matérialisent le fait que deux véhicules appartenant aux classes correspondant aux extrémités de l'arête se suivent sur la ligne de production. Les boucles sur les sommets indiquent que deux véhicules d'une même classe peuvent se suivre sur la ligne de production. Une solution est alors *un parcours* qui traverse chaque sommet autant de fois qu'il existe de véhicules appartenant à la classe associée au sommet.

Dans un domaine bien différent, celui de la bioinformatique, de nombreux problèmes présentent des similitudes au niveau de la modélisation [SET 97]. A titre d'exemple, certaines instances du problème de la reconstruction d'une séquence nucléotidique[3] à partir de données issues de la technique du séquençage par hybridation peuvent se modéliser par un graphe [LYS 88], dont chaque sommet représente un mot de longueur fixée et un arc a_{ij} entre deux sommets m_i et m_j est valué par la longueur maximum du suffixe de m_i qui est aussi un préfixe de m_j (on parle de recouvrement). Dans ce graphe, dans la mesure où certains mots peuvent apparaître plusieurs fois dans la séquence, une solution au problème de la reconstruction est un *parcours* [BLA 02]. Une méthode de résolution possible qui met en œuvre un système à base de fourmis artificielles est présentée dans cet ouvrage [BAL 09].

2.2.2. *Chemins simples et multiples*

2.2.2.1. *Plus court chemin*

Un problème d'optimisation du coût de la production d'énergie est présenté dans [SAN 09]. Les auteurs décrivent un système composé de k machines dont l'objectif est de satisfaire une demande d'énergie qui varie sur un intervalle de temps subdivisé en N étapes. Le démarrage et l'extinction des machines ont un coût et chacune d'elles est

3. Suites de symboles formées à partir des célèbres quatre bases *Adenine (A), Cytosine (C), Guanine (G)* et *Thymine (T)*.

caractérisée par un coût de production. Une solution au problème est un ensemble de N vecteurs d'états, un par étape de temps, dont les composantes, à valeur dans $\{0, 1\}$, représentent l'état de fonctionnement des machines. Les auteurs proposent une modélisation de ce problème sous la forme d'une *grille orientée* de dimension $2^k \times N$. Les sommets représentent les vecteurs d'états, à chacun desquels est associé un coût de production, et les arcs entre les sommets de deux colonnes de cette grille représentent le passage d'un état de fonctionnement du système à un autre état de fonctionnement à l'étape suivante. A chaque arc est associé le coût cumulé des démarrages et extinctions engendrés par ce passage. En outre, l'existence des arcs est conditionné par un ensemble de contraintes détaillées dans le chapitre correspondant. En termes de graphes, une solution à ce problème est un *chemin* constitué de N sommets. La longueur d'un chemin est calculée en cumulant les coûts de production associés aux sommets traversés et les coûts de démarrage et d'extinction associés aux arcs appartenant au chemin. Sur la base de ce graphe, la meilleure solution qui peut être obtenue correspond au *plus court chemin* entre un sommet appartenant à la première colonne et un sommet appartenant à la dernière colonne.

2.2.2.2. *Chemin hamiltonien*

Dans l'énoncé du problème de l'optimisation de la production de barres d'aluminium [GRA 09], une usine répond à un ensemble de commandes de pièces caractérisées par leur type d'alliage, leur forme, et une date de livraison. L'objectif est d'optimiser la production de barres permettant de produire ces pièces avec un triple objectif de minimisation. En particulier, la production de pièces correspondant à deux commandes successives s'accompagne d'un ensemble de réglages qui ont un coût qui, de manière globale, doit être minimisé. Une solution est un ordonnancement des différentes commandes. Les auteurs proposent une modélisation du problème sous la forme d'un *graphe complet* dont les sommets correspondent aux commandes et dont les arêtes sont valuées par les coûts induits par l'enchaînement de deux commandes associées aux extrémités. Une solution est alors un *chemin hamiltonien* dans ce graphe.

Bien qu'il ne soit pas mentionné, on retrouve implicitement un *graphe complet* dans le problème de la construction de claviers virtuels présenté dans [SEP 09]. Ce problème consiste à ordonner les différentes touches d'un clavier virtuel de manière à minimiser une certaine fonction de coût. Dans le graphe complet qui modélise le problème, les sommets sont associés aux touches du clavier et les arêtes matérialisent leur voisinage. L'ordre des touches du clavier virtuel correspond à un *chemin hamiltonien* dans ce graphe.

D'une manière générale, le *chemin hamiltonien* représente la structure solution des problèmes d'optimisation dont l'objet est de construire une permutation de n éléments pour former une liste ordonnée constituée de ces n éléments. Dans les cas les plus courants, le graphe considéré peut être complet, mais certaines contraintes propres aux problèmes pratiques sous-jacents peuvent rendre certains chemins hamiltoniens non valides.

2.2.2.3. *Chemins multiples*

Le besoin de mettre en évidence des chemins multiples dans un graphe peut répondre à deux motivations différentes. Il peut s'agir soit de chemins indépendants les uns des autres, chaque chemin représentant une alternative possible pour un problème donné, soit de chemins concurrents entre eux, chaque chemin représentant une partie de la solution.

A titre d'exemple, dans le domaine des réseaux de communications, il peut être important de disposer de plusieurs chemins de communication sans élément commun, ils sont alors dits deux-à-deux arêtes distincts. Cette contrainte peut être imposée pour des exigences de sécurité, une panne survenant sur l'un des liens de communication dégrade le débit du transfert mais ne l'interrompt pas. La mise en évidence de k chemins deux-à-deux arêtes distincts entre une source et une destination permet de garantir une certaine forme de robustesse, puisqu'un graphe k-arête connexe permet la perte de $k-1$ liens sans que la connexité du graphe n'en soit affectée. Dans ce cas de figure, les chemins ne sont pas indépendants les uns des autres, ils sont en concurrence du point de vue des arêtes.

Considérons maintenant le cas d'une recherche d'itinéraire qui soit à la fois court en temps, court en distance et de faible coût. Il s'agit d'un problème multi-objectif. Chaque objectif peut être optimisé indépendamment des autres. Dans ce cas, une solution à chacun des objectifs est un chemin. Dans le cas idéal, un chemin unique répond aux trois objectifs simultanément, mais dans la plupart des cas, il existe un ensemble de solutions appelé front de Pareto dont chaque point correspond à un chemin qui peut partager des éléments communs (sommets et ou arêtes) avec les autres chemins.

Dans le processus de traitement de textes écrits en langue naturelle, il est un problème connu sous le nom de « désambiguïsation » qui consiste à déterminer pour chaque terme ayant plusieurs sens possibles, le meilleur en fonction du contexte général du texte. Ce problème est décrit en détail dans [GUI 09]. Le graphe utilisé pour modéliser le problème est un *arbre* déduit de la structure syntaxique du texte. Dans ce graphe les sommets sont associés aux sens des mots, aux mots eux-mêmes, ainsi qu'aux éléments syntaxiques. En cours de traitement, de nouvelles arêtes sont ajoutées entre les sens des mots qui s'accordent, et une solution est généralement un *chemin* liant les sommets qui leur sont associés. Lorsque les phrases sont ambiguës, comme cela peut être le cas pour des textes humoristiques, une solution n'est plus un unique chemin, mais un ensemble de chemins qui constituent autant d'interprétations possibles du texte.

2.2.3. *Partitions*

L'étude de l'évolution d'un écosystème requiert souvent une modélisation des éléments qui le constituent et des relations entre ces éléments. L'étude *in silico* recourt le

plus souvent à la simulation. Constituée de très nombreuses tâches de calcul communiquant abondamment entre elles, cette application, pour être exécutée efficacement, nécessite à la fois une bonne répartition de la charge entre les ressources de calcul et une allocation des tâches aux ressources qui essaye de tenir compte au mieux des volumes de données échangés et de la fréquence de leurs communications. Ce problème, par essence dynamique, est traité dans [BER 07]. En fonction de leurs caractéristiques des ensembles de tâches nécessitent des volumes variables de communications, certaines communiquant intensivement tandis que d'autres n'échangent que très peu d'informations. L'objectif de la répartition dynamique de charge consiste à équilibrer les calculs entre les différentes ressources, tout en minimisant les volumes de communication entre processeurs. Selon l'hypothèse dite de « localité », les communications entre deux tâches allouées à un même processeur peuvent être négligées, ainsi, le problème s'exprime comme un compromis entre répartir les tâches pour équilibrer la charge et regrouper les tâches pour minimiser les volumes de communication. Les auteurs modélisent ce problème par un graphe dont les sommets représentent les tâches de calcul et les arêtes représentent les communications. Une solution à un instant donné est constituée par des ensembles de sommets qui constituent une partition de l'ensemble des tâches. Ce type de structures se retrouve également dans les problèmes de partitionnement de graphes [KUN 94] ou de détection de communautés [DUT 09].

2.2.4. *Conclusion*

Il est bien évident que la grande diversité des problèmes d'optimisation ne peut se satisfaire des quelques structures présentées dans les paragraphes précédents pour représenter la variété des solutions. Conscients de laisser de côté d'importantes structures, nous nous sommes pourtant attachés à en décrire quelques-unes parmi les plus souvent rencontrées. Le problème de configuration en programmation par contraintes présenté dans [ALB 09] illustre cette variété qui rend vaine toute tentative d'exhaustivité. Dans ce texte, les auteurs décrivent un modèle qui correspond à un graphe virtuellement infini dans lequel chaque sommet est associé à un composant et dans lequel un lien entre deux sommets est une relation entre les composants correspondants. Une solution correspond à un ensemble de composants et de relations entre composants. Il s'agit d'un *sous-graphe* du graphe qui modélise l'espace de recherche. La solution est nommée par les auteurs *graphe de structure*.

Dans un autre registre, un *couplage* ou *un stable* peut représenter une solution dans le graphe qui modélise le problème. Ces structures se retrouvent dans le problème d'optimisation de la capacité d'un réseau ferroviaire décrit dans [GAN 09] ou dans le problème classique de la coloration de graphes dont on pourra trouver une description détaillée dans [HER 09], deux des chapitres de cet ouvrage.

La conclusion qui s'impose est que pour une majorité de problèmes d'optimisation qui sont modélisés par des graphes, les solutions correspondent à des structures très souvent identifiables au sein de ces graphes.

Dans le processus de résolution d'un problème d'optimisation combinatoire par algorithmes de fourmis artificielles, la modélisation du problème à l'aide d'un graphe et l'expression des solutions sous la forme de structures constituent une partie importante du chemin qui mène du problème à de bonnes solutions. Une autre partie du chemin est parcourue lorsque l'on dispose d'un système à base de fourmis artificielles capables de construire ou de mettre en évidence au sein du graphe des structures du même type que les structures qui sont des solutions du problème d'origine. Cette étape fait l'objet de la partie suivante. Différents systèmes à base de fourmis artificielles qui permettent de générer des structures de différents types sont décrits et analysés.

2.3. Du système à la structure

Si dans la nature il est possible d'observer des structures complexes construites par les fourmis, le processus à la base de cette capacité n'est pas toujours simple à reproduire pour les fourmis artificielles. Dans cette partie, nous essayons de décrire précisément comment les systèmes à base de fourmis artificielles sont capables de générer des structures dans les graphes, indépendamment de tout problème d'optimisation sous-jacent. Il s'agit de l'une des propriétés les plus remarquables de ces approches. Bien que le domaine de l'optimisation par colonies de fourmis artificielles ait bientôt une vingtaine d'années, de nombreux articles récents de la littérature scientifique consacrent encore un court paragraphe pour expliquer le fonctionnement des colonies de fourmis artificielles. Le principe général et les principaux éléments constitutifs de ces systèmes sont décrits à grands traits et leurs effets sont présentés comme des évidences. De notre point de vue quelques explications sont malgré tout nécessaires pour mieux en saisir le fonctionnement.

Un système classique à base de fourmis artificielles est généralement composé : d'un environnement, dans notre cas il s'agira d'un graphe, de fourmis, d'un mécanisme de mémorisation et d'un mécanisme d'oubli. Dans la majorité des cas, le fonctionnement du système repose dans son ensemble sur le déplacement des fourmis dans l'environnement et sur le dépôt d'informations dans cet environnement par ces mêmes fourmis. Le plus souvent, les informations déposées dans l'environnement sont nommées *phéromones* (une explication plus détaillée est proposée dans [LEN 09]). La persistance de cette information constitue le mécanisme de mémorisation. Cependant, les phéromones permettent également aux fourmis de communiquer de manière indirecte. Ce mécanisme de communication indirect par lecture des informations contenues dans l'environnement est appelé *stigmergie*. Le point-clé est que les fourmis sont attirées par les régions de l'environnement les plus riches en phéromones. Ce mécanisme constitue une forme de rétroaction positive puisque statistiquement le nombre

de fourmis dans une région augmente avec le niveau de phéromone qui augmente avec le nombre de fourmis. Afin de limiter les effets de cette rétroaction positive, le mécanisme de mémorisation s'accompagne généralement d'un mécanisme d'oubli qui joue le rôle inverse, ce qui permet au système d'évoluer plus lentement. Dans l'idée, le principe est proche de la descente de gradient. Ce mécanisme d'oubli est mis en œuvre par *l'évaporation* des phéromones. Ainsi, les éléments, les règles, les mécanismes qui permettent de *biaiser* le déplacement dans l'environnement, constituent la clé de voûte des stratégies qui permettent aux fourmis de construire ou de mettre en évidence des structures dans les graphes.

Les objectifs de cette partie sont d'analyser le fonctionnement des systèmes de fourmis artificielles, par des exemples simples, en limitant leur complexité et la multiplication des paramètres de réglages, et de montrer qu'ils permettent de produire des structures variées sans nécessiter un effort de conception insurmontable.

Dans un premier temps, nous nous attachons à la mise en évidence d'un simple chemin entre deux sommets dans un graphe, indépendamment de sa qualité en termes de longueur. Dans la suite de cette partie, nous montrons comment en modifiant qualitativement le système, il est possible de produire des chemins multiples indépendants ou concurrents, ainsi que des partitions de l'ensemble des sommets des graphes parcourus par les fourmis. Au préalable, une analyse des processus attachés aux algorithmes de fourmis s'avère intéressante pour mieux comprendre les phénomènes qui sont en action. En effet, un algorithme de fourmis est avant tout un algorithme qui repose sur la notion de *marche aléatoire*, une marche que les caractéristiques du système permettent de biaiser pour mettre en évidence les structures qui nous intéressent.

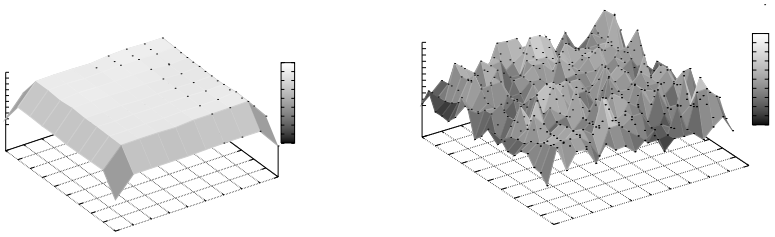
Dans la suite de cette partie, nous considérons un graphe $G = (S, A)$ d'ordre n ($|S| = n$) et possédant m arêtes ($|A| = m$). Le principe à la base du fonctionnement du système testé est le suivant. Le système est à temps discret, le temps est divisé en étapes. Au cours d'une étape toutes les fourmis se déplacent, aucune ne reste en place. Un déplacement consiste à changer de sommet en traversant une arête. A l'issue du déplacement de toutes les fourmis, les phéromones sont mises à jour sur l'ensemble des arêtes et/ou des sommets selon le système testé, puis l'évaporation est appliquée. L'itération suivante peut alors commencer.

2.3.1. *Fourmis et marches aléatoires*

Les systèmes à base de fourmis artificielles reposent sur le mouvement des fourmis. Ce sont elles qui génèrent les structures, par les effets combinés de leurs mouvements et des dépôts de phéromones. Considérons un système dont les fourmis se déplacent dans un graphe. Si le mouvement des fourmis n'est soumis à aucune influence particulière, leur déplacement est une *marche aléatoire* (*random walk* dans la terminologie anglo-saxonne). Dans ces conditions, la probabilité pour une fourmi

de traverser une arête est la même pour toutes les arêtes [LOV 93], ainsi si les fourmis déposent des phéromones sur les arêtes et n'en tiennent pas compte pour effectuer leurs déplacements, toutes les arêtes auront des quantités de phéromones comparables. En conséquence, le premier des objectifs dans la conception d'un système à base de fourmis artificielles est de *biaiser* la marche aléatoire pour briser cette équiprobabilité au profit des arêtes et des sommets qui, considérés collectivement, possèdent des propriétés particulières.

Notons toutefois qu'il n'en va pas exactement de même avec les sommets puisque la probabilité de présence d'une fourmi, animée d'une marche aléatoire, sur un sommet s est proportionnelle à $\frac{\text{degré}(s)}{m}$ (où m est, rappelons-le, le nombre d'arêtes du graphe). En conséquence, dans les mêmes conditions, si les fourmis déposaient des phéromones sur les sommets, les quantités de phéromones seraient différentes pour des sommets de degrés différents. Les figures 2.1a et 2.1b illustrent ce phénomène.



(a) *Grille 10x10*

(b) *Grille irrégulière 20x20 (voisinage de Moore incomplet)*

Figure 2.1. *Lorsque les quantités de phéromones ne sont pas prises en compte par les fourmis pour leurs déplacements, les sommets sont visités un nombre de fois proportionnel à leur degré. Ceci est parfaitement illustré lorsque le graphe considéré est une grille. Sur la figure de gauche, 25 fourmis se déplacent dans une grille de taille 10 et le nombre d'itérations a été fixé à 100 000. Sur la figure de droite, 100 fourmis se déplacent dans une grille avec un voisinage de Moore (huit voisins) parfois incomplet (et donc irrégulier). La hauteur des « pics » observés est proportionnelle au degré du sommet correspondant. Le nombre d'itérations a été fixé à 10 000.*

2.3.2. *Génération d'un chemin*

Considérons un graphe et deux sommets particuliers, un sommet source (le nid de la colonie), et un sommet destination. Le résultat que l'on souhaite obtenir de la colonie est la mise en évidence d'un chemin entre ces deux sommets. Nous considérons la tâche réalisée lorsque le niveau de phéromone des arêtes le long de ce chemin est supérieur au niveau de phéromone des autres arêtes du graphe.

Le système de fourmis fonctionne étape par étape. Au cours d'une étape :

1) chaque fourmi choisit l'un des voisins du sommet v sur lequel elle est positionnée pour se déplacer. La probabilité de choix du sommet v', voisin de v, dépend du niveau de phéromone de l'arête (v, v'). Cette probabilité se calcule très simplement comme le rapport entre la quantité de phéromone de l'arête (v, v') et la somme des quantités de phéromones des arêtes incidentes à v :

$$P(v') = \frac{\text{phéromone}(v, v')}{\sum_{w \in \text{voisins}(v)} \text{phéromone}(v, w)} \tag{2.1}$$

2) si la fourmi a atteint la destination, elle retourne vers le sommet source ;

3) lorsque toutes les fourmis se sont déplacées : chaque arête reçoit une quantité de phéromone proportionnelle au nombre de fourmis qui l'ont traversée, puis ;

4) le niveau de phéromone de chaque arête est diminué d'une fraction correspondant à *l'évaporation*.

Les deux dernières phases peuvent être permutées sans conséquence notable.

Si le graphe de départ est libre de toute trace de phéromone, au cours de la première étape, chaque fourmi se déplace depuis le sommet source s vers un sommet voisin choisi aléatoirement. A l'issue du déplacement des fourmis, les arêtes adjacentes à s reçoivent chacune une quantité de phéromone proche du *ratio* entre le nombre de fourmis et le degré du sommet source. Durant la seconde étape, le taux de phéromone étant non nul seulement pour les arêtes dont l'une des extrémités est le sommet source s, toutes les fourmis retournent vers s. La figure 2.2 illustre ce phénomène. Les faibles fluctuations qui peuvent exister entre les niveaux de phéromone des différentes arêtes adjacentes à s entraînent après plusieurs itérations le choix d'une seule et unique arête par l'ensemble des fourmis qui effectuent des allers-retours entre ses deux extrémités. Ce phénomène de choix arbitraire entre plusieurs possibilités *a priori* équiprobables a été mis en évidence par l'expérience du *binary bridge* conçue par J.L. Deneubourg et ces collègues pour des fourmis réelles [DEN 90].

Pour résoudre ce problème, la solution la plus souvent mise en œuvre consiste à déposer sur chaque arête du graphe, à l'initialisation, une quantité non nulle de phéromone et à imposer après chaque étape que chaque arête dispose d'une quantité minimale de phéromone. Une remarque s'impose à ce propos. Si la quantité initiale de phéromone est largement supérieure au dépôt effectué par l'ensemble des fourmis alors le processus dans un premier temps sera proche d'une marche aléatoire. Du point de vue des méthodes d'optimisation, il s'agit d'un comportement principalement exploratoire. Au contraire, si la valeur est très faible au regard du dépôt effectué par les fourmis et sans ajout de règles comportementales supplémentaires, alors le système se comportera d'une manière similaire à ce qu'il fait lorsque le graphe est libre de toute trace de phéromone.

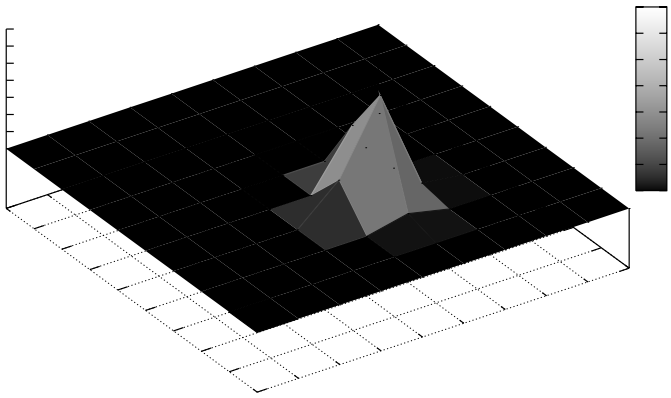


Figure 2.2. *Lorsque les quantités de phéromone sont prises en compte par les fourmis pour leurs déplacements et lorsque le niveau initial de phéromone est nul, seuls sont visités les sommets voisins du sommet source. Cette figure est l'image du nombre de passages de fourmis en chaque point du graphe. La zone noire correspond aux sommets qui n'ont reçu aucune visite de fourmi. Cette zone représente tous les sommets à l'exception du sommet source et de ses voisins immédiats. Le « pic » correspond à la position de la source. Sur cet exemple, 25 fourmis se déplacent dans une grille de taille 10 et le nombre d'itérations a été fixé à 1 000.*

Il existe, au moins, deux politiques de retour pour les fourmis qui ont atteint la destination d. La première politique consiste à faire le parcours à l'envers et à effectuer un dépôt de phéromone sur chacune des arêtes traversées. La seconde politique consiste à éliminer du parcours les cycles pour ne conserver qu'un chemin, c'est-à-dire un parcours dont chaque sommet et chaque arête n'est traversé qu'une seule fois. Arrêtons-nous un instant sur ces deux politiques.

La première politique n'apporte pas de réponse évidente au problème de la construction d'un chemin. En effet, de simples simulations montrent que le nombre de passages par la destination diminue fortement lorsque cette politique est mise en œuvre. Le même phénomène se produit également avec la seconde politique même s'il est un peu atténué. Comment cela s'explique t-il ? Chacune de ces deux politiques consiste à renvoyer une fourmi vers la source dès lors que celle-ci a atteint la destination, en conséquence, le voisinage de la destination est plus pauvre en fourmis que d'autres régions du graphe dont les sommets peuvent être visités plusieurs fois sur

de courtes périodes. Ainsi, si chaque fourmi dépose des phéromones à chaque passage sur une arête, les arêtes incidentes à la destination auront, après quelque temps, les quantités de phéromone les plus faibles. Les effets de ce phénomène peuvent être visualisés sur la figure 2.3.

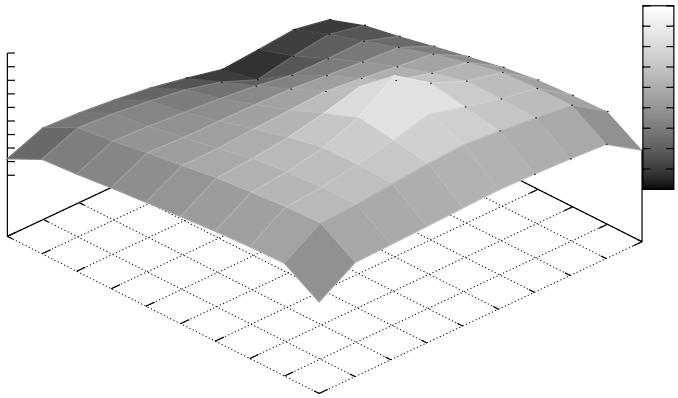


Figure 2.3. *Sur cette figure, les fourmis parcourent le graphe et déposent des phéromones à l'aller comme au retour. Un « pic » apparaît en clair et un « creux » en sombre, ces deux positions particulières correspondent respectivement à la source et à la destination. Cette différence s'explique par les règles de comportement des fourmis qui peuvent visiter le sommet source fréquemment alors que la visite du sommet destination les renvoit sans condition vers le sommet source.*

La solution qui s'impose est d'autoriser les fourmis à ne déposer des phéromones que sur le chemin du retour et seulement selon la seconde politique de retour puisque la première politique en permettant les cycles sur tous les sommets excepté la destination introduit un biais qui va à l'encontre de l'objectif initial. Cette simple modification dans le mode de fonctionnement du dépôt de phéromone suffit à faire, enfin, apparaître un chemin comme l'illustre la figure 2.4.

Il est important de noter que *l'évaporation* joue un rôle majeur dans le processus. Le mouvement seul des fourmis ne permet pas de construire une structure. Les phéromones jouent le rôle d'une mémoire globale *à court terme*. Si l'évaporation est absente du processus, tous les chemins seront conservés en mémoire ce qui permet à beaucoup de chemins d'être représentés, mais le rapport signal/bruit est très faible. Au contraire,

si l'évaporation est trop forte, des bouts de chemins éloignés de la source peuvent ne plus être mémorisés. Une évaporation trop forte nivelle le niveau de phéromone de l'ensemble des arêtes au niveau minimum ce qui ne permet plus de distinguer les chemins les uns des autres, contraignant les fourmis a effectuer des choix aléatoires.

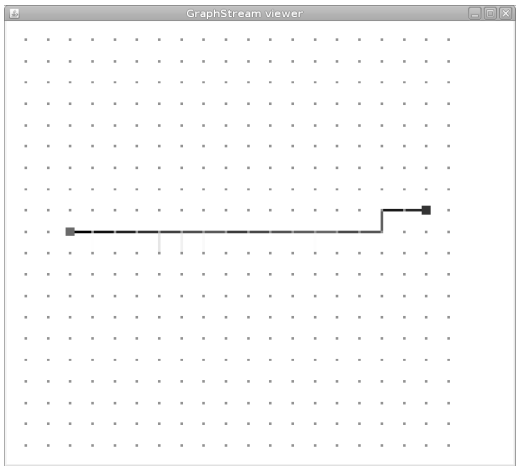


Figure 2.4. *Lorsque les phéromones ne sont déposées par les fourmis que sur le chemin du retour, correspondant au parcours de l'aller débarrassé des cycles, une structure de type chemin apparaît. Le système considéré est constitué d'une grille de 400 sommets dans laquelle 100 fourmis se déplacent. La quantité initiale de phéromone est de 0,3 pour chaque arête. Chaque fourmi dépose une quantité de 0,04 sur chaque arête sur le chemin du retour. Le taux d'évaporation est de 5 % à chaque itération. Le niveau minimum imposé pour chaque arête est de 0,0001 et le chemin se stabilise après environ 15 000 itérations.*

Si chemins et parcours représentent une classe importante de structures solutions, ils ne permettent pas de mettre en évidence plusieurs solutions pour un problème donné. Dans la suite de cette partie, nous nous intéressons à la formation d'autres structures, en commençant par les structures composées de plusieurs chemins.

2.3.3. *Génération de chemins multiples*

A plusieurs reprises dans les exemples décrits dans la première partie de ce chapitre, nous avons rencontré des problèmes pour lesquels une solution s'exprimait sous la forme d'un ensemble de chemins. Nous présentons un système à base de fourmis artificielles capable, sur la seule base des phéromones, de générer plusieurs chemins. Dans le cadre de ce chapitre, nous distinguons deux cas.

Le premier se rapporte aux ensembles de chemins qui constituent des ensembles de solutions. Par exemple, pour le problème du plus court chemin, il peut exister plusieurs solutions qui peuvent partager certaines arêtes, ou encore dans le problème de la désambiguïsation de la langue naturelle [GUI 09], plusieurs chemins d'interprétations sont possibles pour un même texte, chemins qui peuvent avoir des éléments en commun.

Il existe également des problèmes pour lesquels une solution est composée d'un ensemble de chemins pour lesquels aucune arête ne peut être partagée. Le problème de la vérification de la k-arête connexité d'un graphe rentre dans cette catégorie. Ce type de problèmes constitue notre second cas d'étude.

L'expérience du *binary bridge* montre qu'une colonie de fourmis ne permet généralement pas de mettre en évidence deux chemins dans un graphe. En effet, le principe de base repose sur deux processus antagonistes. Le premier est matérialisé par le dépôt de phéromone. L'arête sur laquelle sont déposées des phéromones augmente ses chances d'attirer de nouvelles fourmis qui à leur tour déposeront des phéromones. Cette rétroaction positive se traduit par deux effets, conserver en mémoire une trace des arêtes les plus fréquentées et assurer le recrutement de nouvelles fourmis vers ces arêtes. Le second processus est exprimé par le principe d'évaporation. L'évaporation a pour effet immédiat d'effacer une fraction des phéromones sur chacune des arêtes. Cette action va à l'encontre du processus précédent puisqu'elle nivelle les niveaux de phéromone ce qui a pour conséquence, d'une part d'introduire une forme d'oubli des arêtes les plus fréquentées, et d'autre part de limiter l'effet de recrutement des fourmis. On parle de rétroaction négative. Lorsque plusieurs choix équivalents s'offrent aux fourmis, l'action simultanée de ces deux processus produit un équilibre instable entre ces différents choix. Il suffit généralement de faibles fluctuations pour que l'une des alternatives devienne prépondérante, ce qu'ont mis en lumière J.L. Deneubourg et ses collègues [DEN 90].

Partant de l'hypothèse qu'il est difficile de mettre en évidence plusieurs chemins à l'aide d'une seule colonie, l'idée d'utiliser plusieurs colonies s'impose naturellement.

Lorsque les chemins qui doivent être produits sont indépendants, nous nous trouvons dans une situation qui constitue une application directe de l'étude présentée au paragraphe 2.3.2. L'ajout de nouvelles colonies dans le processus de construction de structure ne s'accompagne d'aucune difficulté particulière. La figure 2.5 offre une illustration des résultats qui peuvent être attendus de ce type de systèmes à base de plusieurs colonies de fourmis artificielles. Là où une seule colonie tend vers une unique solution, un ensemble de colonies peut générer plusieurs structures de qualités comparables. Cette approche peut être mise en œuvre avec intérêt pour un problème d'optimisation multi-objectif. Dans ce cas, à chaque objectif est associée une colonie.

Le second cas concerne les problèmes pour lesquels une solution est composée d'un ensemble de chemins qui ne sont plus indépendants, en particulier de chemins qui ne doivent partager aucune arête. C'est le cas par exemple de la vérification de la *k*-arête connexité d'un graphe. Dans ce cas, chaque colonie doit construire un chemin, tout en évitant d'emprunter les arêtes choisies par les autres colonies et tout en empêchant ces autres colonies d'utiliser les arêtes qu'elle a choisies. L'utilisation de phéromones qui se distinguent en fonction des colonies auxquelles appartiennent les fourmis qui les déposent est une solution souvent envisagée. Une couleur est souvent associée à chaque colonie et les phéromones déposées sont dites colorées. En outre, cet artifice offre un moyen simple pour l'observation du phénomène d'émergence de structures.

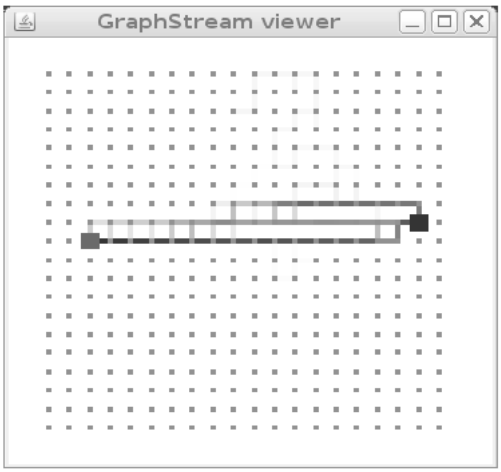


Figure 2.5. *Cette image est le résultat obtenu à partir de trois colonies conçues pour construire des chemins. Chacune d'elles, indépendamment des autres, a mis en évidence un chemin entre le sommet source et le sommet destination. On peut remarquer que certaines arêtes sont partagées. C'est ce type de fonctionnement qui a été mis en œuvre pour le traitement du problème de la désambiguïsation présenté dans [GUI 09]. Paramètres : la valeur du taux d'évaporation a été fixée à 2 %, le nombre de fourmis est égal au nombre de colonies multiplié par le quart du nombre de sommets. La quantité initiale de phéromone a été fixée à 0,3 la valeur du dépôt est de 0,04 et la valeur minimum sur chaque arête a été fixée à 0,0001.*

Le dépôt de phéromone induit cette fois deux types de rétroaction. Une rétroaction positive habituelle qui favorise le recrutement des fourmis de la même couleur, et une rétroaction négative nouvelle qui repousse les fourmis de couleurs différentes. Ce phénomène se traduit par une ségrégation des fourmis en fonction de leur couleur. Les fourmis d'une couleur donnée vont se cantonner à une région particulière du graphe au sein de laquelle elles construisent un chemin. Du point de vue du déplacement, cela revient à biaiser la marche aléatoire, non plus en fonction des seules quantités de

phéromone présentes sur les arêtes, mais en fonction de la proportion de phéromone de la colonie à laquelle appartient la fourmi par rapport aux quantités de phéromone déposées par les fourmis d'autres colonies. Dans la formule du choix cela se traduit par la distinction des différentes phéromones.

Pour la mise en œuvre de ce mécanisme, plusieurs possibilités peuvent être envisagées. Dans le système que nous avons conçu et testé, la fourmi privilégie l'arête dont la proportion de phéromone de sa couleur est la plus importante parmi les voisins du sommet sur lequel elle est positionnée. On note C l'ensemble des couleurs. Pour une fourmi de couleur c positionnée sur le sommet v, la probabilité de choix du sommet v' voisin de v vaut :

$$P(v') = \frac{\frac{\text{phéromone de couleur } c(v,v')}{\sum_{c' \in C} \text{phéromone de couleur } c'(v,v')}}{\sum_{w \in \text{voisins}(v)} \left(\frac{\text{phéromone de couleur } c(v,w)}{\sum_{c' \in C} \text{phéromone de couleur } c'(v,w)} \right)} \tag{2.2}$$

S'il est permis d'espérer voir apparaître les structures que nous attendons, les résultats des tests sont assez mitigés. Une rapide analyse de la formule de choix permet de réaliser que la force d'attraction n'est pas assez forte par rapport aux comportements d'évitement des fourmis. La figure 2.6 illustre le problème de la convergence vers trois chemins.

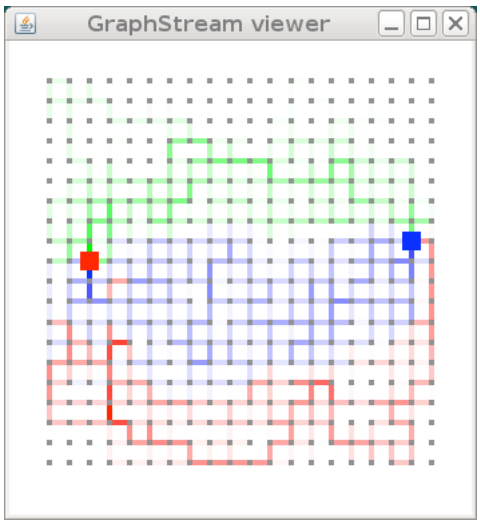
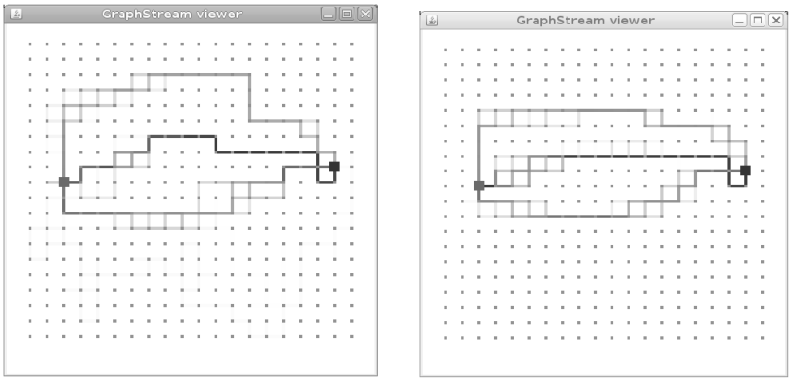


Figure 2.6. *Sur cette image, le fonctionnement général du système composé de trois colonies en compétition apparaît assez clairement. Les colonies occupent chacune une région du graphe, sans qu'il leur soit possible de dégager chacune, rapidement, un chemin non conflictuel depuis la source vers la destination. La formule de calcul des probabilités utilisée est celle de l'équation (2.2).*

Pour améliorer cette situation, nous avons, dans la formule du calcul de la probabilité de choix, amplifié les différences entre les différentes valeurs de probabilité, afin de renforcer le phénomène de rétroaction positive. La nouvelle formule reste très proche de la précédente :

$$P(v') = \frac{\frac{\text{phéromone de couleur } c(v,v')}{\sum_{c'(\neq c)\in C} \text{phéromone de couleur } c'(v,v')}}{\sum_{w\in \text{voisins}(v)} \left(\frac{\text{phéromone de couleur } c(v,w)}{\sum_{c'(\neq c)\in C} \text{phéromone de couleur } c'(v,w)} \right)} \tag{2.3}$$

L'utilisation de cette formule de calcul pour les probabilités de choix donne de bien meilleurs résultats dont on peut avoir un aperçu sur la figure 2.7.



(a) *Après 17 000 itérations* (b) *Après 100 000 itérations*

Figure 2.7. *Avec l'utilisation de la fonction de calcul de la probabilité de choix (2.3), les trois chemins non conflictuels se dégagent déjà nettement au bout de 17 000 itérations. Après 100 000 itérations, les trois chemins non conflictuels ont légèrement évolué. En effet, les chemins ont une tendance naturelle à diminuer et à tendre vers un plus court chemin. Dans le cas qui nous occupe, il n'existe que deux plus courts chemins non conflictuels, mais le système parvient malgré tout à maintenir trois chemins deux-à-deux arêtes disjointes simultanément.*

La génération de chemins par des colonies de fourmis est un phénomène qui semble assez naturel dans la mesure où chaque fourmi effectue un parcours et donc, construit une solution. Mais d'autres structures, dont on souhaite la mise en évidence par le système de fourmis artificielles, sont de nature différente de ce point de vue. En effet, les capacités individuelles d'une fourmi ne lui permettent pas, dans la majorité des cas, de mettre en évidence un ou plusieurs ensembles de sommets. C'est le cas de toutes les structures qui ne sont pas constituées de chemins ou de parcours. Dans la suite de cette partie, nous étudions la manière dont il est possible de concevoir un système de fourmis artificielles pour qu'il produise un partitionnement de l'ensemble des sommets d'un graphe.

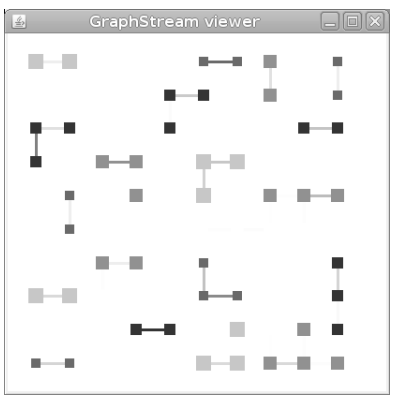
2.3.4. *Génération de partitions*

La mise en évidence d'ensembles de sommets par des colonies de fourmis artificielles ne relève pas seulement des mécanismes qui sont à l'œuvre pour la mise en évidence de chemins. En effet, une fourmi construit naturellement un parcours, voire un chemin dans un graphe, son déplacement seul le lui permet. En revanche, une fourmi n'a pas la capacité de mettre en évidence une partition de l'ensemble des sommets d'un graphe, ni même des groupes de sommets. La stratégie de conception d'un système le permettant ne peut donc pas s'appuyer uniquement sur les parcours construits par les fourmis, mais doit considérer la globalité de la colonie et trouver les règles permettant de la guider vers cet objectif. Pour différencier les différents groupes mis en évidence, dans la suite de cette partie, nous aurons recours aux couleurs, comme pour les chemins multiples au paragraphe 2.3.3. Les éléments qui régissent les systèmes à base de fourmis présentés aux paragraphes 2.3.2 et 2.3.3 sont les phéromones qui induisent une rétroaction positive et l'évaporation qui induit une rétroaction négative sur le marquage des chemins. Le principe des phéromones est celui d'une mémoire tandis que celui de l'évaporation s'apparente à une fonction d'oubli qui entraîne la désertion d'une partie du graphe par les fourmis au profit des régions les plus marquées.

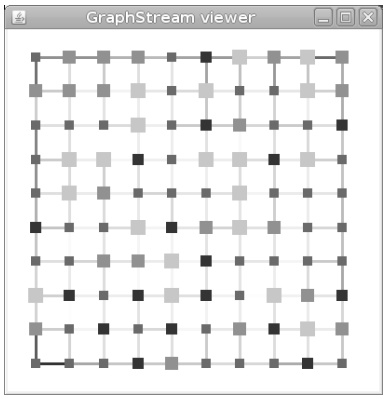
Dans le cas de la recherche de partitions, aucune région ne doit être désertée des fourmis. Il est, en effet, nécessaire que les sommets soient visités de manière régulière pour que leur couleur soit déterminée puis entretenue. La fonction d'oubli de l'évaporation ne peut être conservée que pour permettre l'oubli des couleurs non entretenues au sein de certains sommets. Le dépôt de phéromone joue le même rôle que précédemment, une présence majoritaire de phéromones d'une couleur donnée sur un sommet attire davantage les fourmis de cette couleur vers ce sommet, et, mécaniquement fait « fuir » les fourmis d'autres couleurs. Cependant, ce simple mécanisme ne suffit pas à entretenir le marquage de l'ensemble des sommets du graphe car la rétroaction positive engendrée par les phéromones n'est contrecarrée par aucune force permettant de maintenir une présence des fourmis sur des sommets dont le niveau de phéromone (toutes couleurs confondues) est faible. Les figures 2.8a et 2.9a illustrent cela. Il apparaît clairement que le nombre de passages de fourmis est très hétérogène en fonction des régions du graphe. Ce biais est entièrement aléatoire puisque les fourmis sont initialement positionnées sur l'ensemble des sommets du graphe, de manière équilibrée en nombre et en couleurs, et que le graphe est une grille, c'est-à-dire ne présente aucune irrégularité particulière en dehors des côtés.

Pour mettre en évidence une partition des sommets, il est donc nécessaire d'une part de permettre aux phéromones de jouer leur rôle discriminatoire entre les couleurs, mais il est également nécessaire de garantir un relatif équilibrage en nombre des fourmis sur les sommets. En d'autres termes, il s'agit d'introduire un biais dans la répartition des fourmis dans le système de telle sorte que l'équilibre en nombre pour chaque sommet soit conservé mais que cet équilibre pour les couleurs soit brisé. Des solutions existent. Nous en présentons une assez simple qui consiste à ajouter au système

un principe d'attirance des fourmis vers les sommets les moins occupés. Ce principe est renforcé lorsque le nombre de fourmis présentes sur un sommet est important proportionnellement aux sommets voisins, il constitue donc une rétroaction négative qui va jouer contre la rétroaction positive induite par les phéromones. Avant de décrire les formules de probabilités de choix de déplacements par les fourmis, étudions les forces en présence. Si nous faisons jouer les seules phéromones, sans contrepartie d'équilibrage en nombre de fourmis par sommet, le résultat est l'apparition dans le graphe de groupes de fourmis de même couleur autour d'une ou deux arêtes, et cela en plusieurs endroits dans le graphe comme l'image de la figure 2.8a. Au contraire, si seul l'équilibrage en nombre de fourmis par sommet est actif, alors la répartition obtenue est proche de celle que l'on obtient avec une simple marche aléatoire, comme l'atteste la similitude entre les figures 2.1a et 2.9b. En revanche, aucun groupe de fourmis et *a fortiori* de sommets de même couleur n'est mis en évidence. La figure 2.8b en fournit une illustration.



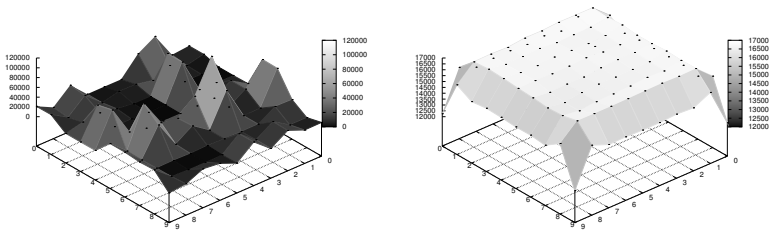
(a) *Visualisation des régions riches en phéromones, lorsque le système est conduit par la seule action des phéromones*



(b) *Visualisation des régions riches en phéromones, lorsque le système est conduit par l'équilibrage en nombre*

Figure 2.8. *Sur la figure de gauche, on peut constater que la seule action de phéromones entraîne une désertification de certaines régions du graphe. Les fourmis sont attirées puis prisonnières des régions riches en phéromones. Sur la figure de droite, au contraire, il semble que les fourmis occupent la totalité du graphe. Le graphe considéré est une grille 10x10, le nombre de colonies a été fixé à quatre et le nombre d'itérations à 2 000. La taille de sommets permet, en complément des niveaux de gris, de distinguer les différentes colonies.*

Au niveau du choix des déplacements, la formule qui permet de calculer les probabilités de choix respectives des différents voisins d'un sommet donné v doit faire intervenir à la fois les phéromones et l'équilibrage en nombre de fourmis. Nous avons choisi de combiner ces deux critères sous la forme d'une somme pondérée. Pour la



(a) *Nombre de passages de fourmis par sommet lorsque le système est conduit par la seule action des phéromones*

(b) *Nombre de passages de fourmis par sommet lorsque le système est conduit par la seule action d'une répartition équilibrée en nombre de fourmis*

Figure 2.9. *Comparaison du nombre de passages de fourmis sur chaque sommet. Sur la figure de gauche, les fourmis ne sont manifestement pas équitablement réparties entre les différents sommets du graphe ce que laisse apparaître un nombre de passages très hétérogène en fonction du sommet considéré. Au contraire, la figure de droite montre un nombre de passages des fourmis par chaque graphe qui correspond à la distribution observée pour une marche aléatoire. Le graphe considéré est une grille 10x10, le nombre de colonies est de quatre et le nombre d'itérations a été fixé à 2 000.*

prise en compte des phéromones, on utilise la formule (2.3) :

$$P_{\text{ph}}(v') = \frac{\frac{\text{phéromone de couleur } c(v,v')}{\sum_{c'(\neq c)\in C} \text{phéromone de couleur } c'(v,v')}}{\sum_{w\in \text{voisins}(v)} \left(\frac{\text{phéromone de couleur } c(v,w)}{\sum_{c'(\neq c)\in C} \text{phéromone de couleur } c'(v,w)} \right)} \tag{2.4}$$

et pour augmenter l'attractivité des sommets qui ne sont que faiblement occupés, nous proposons une seconde valeur inversement proportionnelle au nombre de fourmis présentes sur un sommet v' relativement au nombre total de fourmis situées sur l'ensemble des sommets voisins de v.

$$P_{\text{rep}}(v') = \frac{\frac{1}{\text{nombre de fourmis de } v'}}{\sum_{w\in \text{voisins(v)}} \left(\frac{1}{\text{nombre de fourmis de } w} \right)} \tag{2.5}$$

dont la combinaison donne :

$$P(v') = \frac{P_{\text{ph}}(v') + P_{\text{rep}}(v')}{2} \tag{2.6}$$

On peut remarquer que ces deux termes, $P_{\text{ph}}(v')$ et $P_{\text{rep}}(v')$, expriment pour le premier l'idée d'exploitation et pour le second l'idée d'exploration, deux notions chères

au domaine des métaheuristiques. Comme pour le cas de la majorité des métaheuristiques, il est possible de pondérer l'importance de l'un ou de l'autre de ces termes en leur adjoignant un coefficient multiplicateur.

Le système de fourmis artificielles ainsi structuré permet de mettre en évidence, par la colonisation (en l'occurrence la coloration), des groupes de sommets voisins par l'action conjointe des phéromones et d'une répartition équilibrée[4] en nombre de fourmis sur les sommets. Les figures 2.10 et 2.11 illustrent cela.

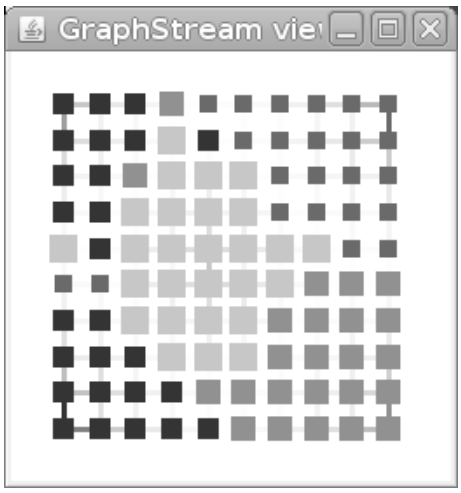


Figure 2.10. *Sur cette figure, quatre colonies mettent en évidence des groupes de sommets voisins par l'action combinée des phéromones et d'un équilibrage en nombre de fourmis présentes sur les sommets. On peut constater qu'aucune région n'est désertée et que des groupes de sommets ont été mis en évidence. Le graphe est une grille 10x10, le nombre de colonies est de quatre, et le nombre d'itérations a été fixé à 50 000. Tous les autres paramètres ont les mêmes valeurs que pour les figures précédentes. La taille des sommets permet de distinguer, en complément des niveaux de gris, les différents groupes.*

2.3.5. *Conclusion*

Tout au long de cette seconde partie, nous nous sommes attachés à montrer que des systèmes à base de fourmis artificielles relativement simples permettent de produire des structures assez complexes, y compris des structures qui ne se déduisent pas de parcours, mode naturel de déplacement des fourmis dans un graphe. Cette simplicité dans le système reste cependant toute relative pour le concepteur qui doit déterminer

4. Nommée parfois « pression démographique ».

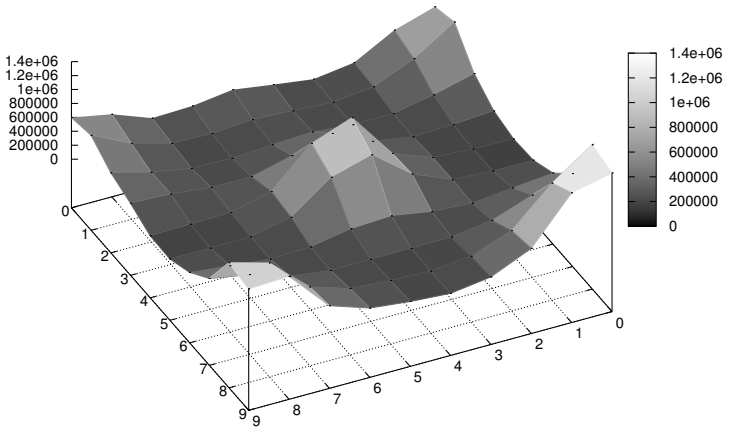


Figure 2.11. *Sur cette figure, quatre colonies mettent en évidence des groupes de sommets voisins par l'action combinée des phéromones et d'un équilibrage en nombre de fourmis présentes sur les sommets. On peut constater que le nombre de passages des fourmis sur les sommets, bien qu'il ne soit pas équivalent partout reste à un niveau important, y compris pour les sommets qui se trouvent à la frontière entre groupes monochromes. Le graphe est une grille 10x10, le nombre de colonies est de quatre, et le nombre d'itérations a été fixé à 50 000. Tous les autres paramètres ont les mêmes valeurs que pour les figures précédentes.*

puis introduire la ou les règles adaptées pour que le système dans son ensemble parvienne à mettre en évidence les structures ciblées. Ainsi, pour réussir à partitionner un graphe, il est nécessaire d'introduire un nouveau mécanisme, dit de « pression démographique », initialement absent des systèmes classiques. Il existe un grand nombre d'autres structures qui correspondent à des solutions de problèmes qui se posent dans les graphes. En particulier, le chemin hamiltonien revient de manière récurrente. Très souvent le graphe à la base de l'étude est un graphe complet et là encore, un mécanisme additionnel est nécessaire pour que le système construise les structures attendues. Les fourmis sont pourvues d'une mémoire qu'elles utilisent comme une liste tabou leur permettant de construire des permutations de l'ensemble des sommets et de déposer des phéromones en conséquence sur les arêtes.

A ce stade, le problème d'optimisation d'origine a été modélisé par un graphe, les solutions de ce problème ont été identifiées comme des structures dans le graphe et nous disposons d'un système à base de fourmis artificielles qui permet de générer dans le graphe des structures qui correspondent à des solutions au problème d'origine. Malheureusement, du point de vue de l'optimisation, ces structures ne sont souvent pas d'une qualité suffisante. La dernière partie fait le point sur quelques techniques

utilisées communément pour améliorer les solutions produites par les algorithmes de fourmis.

2.4. Optimiser : guider la formation des structures

Dans la partie précédente, nous avons montré que différentes caractéristiques de colonies de fourmis permettent de construire ou de mettre en évidence différentes structures qui constituent des solutions à certains problèmes d'optimisation. L'ensemble des tests permet également de constater que cette simple formulation n'est pas suffisante pour obtenir de bons résultats. En clair, si les caractéristiques des colonies permettent de produire des structures dans les graphes, c'est-à-dire de biaiser suffisamment la marche aléatoire des fourmis pour qu'elles mettent en évidence certaines régions du graphe, ce biais n'est souvent pas suffisant pour l'obtention de solutions de très bonne qualité. Pour un grand nombre de problèmes d'optimisation, il est nécessaire d'avoir recours à des éléments de choix supplémentaires qui permettent d'introduire davantage de sémantique du problème d'origine dans le système à base de fourmis artificielles, mais il est également possible de mettre en œuvre, de façon complémentaire, des mécanismes d'amélioration soit de la vitesse d'obtention du résultat, soit de la qualité du résultat.

2.4.1. *Visibilité*

L'un de ces éléments de choix, sans doute le plus important est nommé dans la littérature la *visibilité*, un terme qui est introduit dans la formule de choix de déplacement des fourmis [BON 99].

Cette « visibilité » est une information locale et représente une « désirabilité heuristique » (*heuristic desirability*) de choisir le sommet u lorsque la fourmi est sur le sommet v voisin de u. Ce terme apparaît parfois sous le nom de « désirabilité » et parfois sous le nom de « attractivité ». Assez souvent, dans la littérature, les trois termes « visibilité », « désirabilité » et « attractivité » sont utilisés de manière interchangeable. La visibilité peut représenter par exemple un poids sur les arêtes dont la valeur oriente conjointement aux niveaux de phéromone les fourmis vers le choix de leur prochaine destination. Son importance dans l'obtention de résultats de bonne qualité ou tout simplement pertinents est cruciale. Au pragraphe 2.3.2, nous avons décrit un système à base de fourmis artificielles capable de mettre en évidence un chemin dans un graphe lorsqu'un sommet source et un sommet destination sont choisis. Le système converge de lui-même vers un chemin particulier qui est l'un des plus courts chemins du graphe en nombre d'arêtes séparant les deux sommets. Pour l'ensemble des problèmes pour lesquels un chemin entre source et destination constitue une solution mais dont les arêtes sont valuées par une ou plusieurs valeurs, les chemins construits par le système ne sont plus pertinents. La marche aléatoire est biaisée seulement par la structure du

graphe, les informations ajoutées sur les arêtes et/ou les sommets doivent être introduites dans la formule de probabilité de choix du sommet voisin. Ces informations sont à la base de la construction de la visibilité.

Une ou plusieurs visibilités peuvent être introduites dans la formule de choix, et comme le soulignent M. Gravel et C. Gagné dans la conclusion du chapitre 7, « les métaheuristiques [...] doivent [...] incorporer les éléments spécifiques du problème traité [...] le ou les termes de visibilité sont très importants pour guider le processus de construction de solutions ».

2.4.2. *Autres mécanismes*

Le nombre de techniques d'amélioration des solutions produites par les algorithmes de fourmis est virtuellement infini. Pour une majorité d'entre elles, elles gravitent autour de la recherche de l'équilibre entre l'exploration de l'espace de recherche et l'exploitation des solutions trouvées. Ce principe se retrouve sous différentes formes dans la plupart des métaheuristiques comme la recherche tabou dont la longueur de la liste détermine cet équilibre.

La résolution d'un problème d'optimisation combinatoire consiste à trouver, parmi l'ensemble des solutions réalisables, une meilleure solution. Mais que signifie *meilleure* ? Dans la majorité des cas considérés, la formulation du problème s'accompagne d'une *fonction objectif* qui permet de mesurer numériquement la qualité de la solution proposée. Parmi les techniques les plus utilisées pour améliorer les solutions et la vitesse de convergence, l'une d'elles consiste à ne permettre le dépôt de phéromone qu'aux fourmis ayant produit les meilleures solutions. Il s'agit de l'*élitisme*. Bien que séduisant, ce mécanisme ne peut pas être appliqué à tous les problèmes. En particulier, ceux pour lesquels les fourmis ne construisent pas individuellement des solutions ne peuvent pas bénéficier de cette technique. D'autre part, l'élitisme suppose pour chaque fourmi l'évaluation de la qualité de la solution qu'elle produit par rapport à un étalon, établi par une autre fourmi de la colonie. Or si les données ou les caractéristiques du problème changent au cours du temps l'étalon perd sa validité.

Une autre technique de plus en plus souvent mise en œuvre est le couplage d'algorithmes de fourmis avec des heuristiques de recherche locale. Cette « composition » permet d'atteindre d'excellents résultats. Généralement, l'algorithme de fourmis est utilisé pour explorer l'espace de recherche et l'exploitation des régions d'intérêt est confiée à des méthodes plus efficaces de recherche locale. Là encore, tous les problèmes ne sont pas adaptés à la mise en œuvre d'un tel couplage, en particulier ceux pour lesquels le résultat est une structure globale sur le graphe.

2.4.3. *Voisinage de structure*

Il est nécessaire de souligner un autre point crucial pour l'obtention de bonnes solutions par les algorithmes de fourmis artificielles. Ce point est illustré par la discussion présentée au chapitre [SIA 09]. Dans ce chapitre les auteurs présentent un tour d'horizon des méthodes utilisées pour l'optimisation en variables continues. Il s'agit de problèmes d'optimisation combinatoire pour lesquels certaines variables de décision sont à valeur continue. Parmi l'ensemble des solutions proposées qui sont décrites, notre attention se porte sur les méthodes qui expriment les nombres réels sous la forme de chaînes binaires comme cela est illustré dans le chapitre. Le problème consiste à déterminer la meilleure valeur réelle possible pour chaque variable de décision, étant entendu que chaque variable est associée à un tel graphe. Une solution pour le problème est constituée par la valeur réelle de chaque variable de décision, c'est-à-dire par le chemin dans le graphe associé à chaque variable, il s'agit donc d'un ensemble de chemins. On notera cependant que la notion de chemin dans ce cas particulier pose un problème. En effet, la sémantique associée au chemin n'est pas très claire puisque le chemin représente une suite de bits et qu'un changement local dans le chemin (choix d'une autre arête) ne correspond pas du tout à une variation locale dans l'espace de recherche, en particulier si l'arête modifiée correspond à un bit de poids fort. Ainsi, contrairement à d'autres modélisations qui considèrent le chemin comme une structure représentative d'un voisinage dans l'espace de recherche, ce n'est pas le cas pour le présent problème. Autrement dit, dans la modélisation proposée, un chemin doit être considéré comme un élément atomique, ainsi, le dépôt de phéromone n'apporte rien à la découverte de la solution dans la mesure où la modification d'une arête associée à un bit b fait perdre leur signification à l'ensemble des arêtes associées à des bits de poids plus faible que b. Ainsi, il nous semble essentiel, pour garantir l'efficacité d'une approche à base de fourmis artificielles, qu'un *voisinage structurel dans le graphe corresponde à un voisinage dans l'espace des solutions*.

2.5. Conclusion

La colonie de fourmis fait partie d'un vaste ensemble de systèmes dont chaque représentant est composé de nombreuses entités autonomes interagissant entre elles et avec leur environnement et dont le comportement et l'évolution permettent l'émergence de structures ou de fonctions qui ne peuvent être déduites des caractéristiques et des comportements individuels des entités. Les éléments de cet ensemble sont souvent qualifiés de *systèmes complexes*.

Dans ce chapitre, nous avons proposé une approche de la résolution de problèmes d'optimisation combinatoire basée sur ce principe. Nous considérons que résoudre un problème à l'aide d'un algorithme de fourmis est une tâche qui se décompose en plusieurs parties : la modélisation du problème d'origine sous la forme d'un graphe, l'identification des structures qui au sein de ce graphe constituent des solutions au

problème d'origine, la conception d'un système à base de fourmis artificielles qui construise ou qui mette en évidence le même type de structures et enfin l'adaptation de ce système au problème d'origine par l'introduction et la prise en compte d'éléments spécifiques à ce problème. La visibilité constitue dans les algorithmes de fourmis l'un des éléments les plus courants pour l'introduction de ces informations, mais la modification de la forme des phéromones peut être une autre façon d'y parvenir.

Cette approche de la résolution des problèmes d'optimisation combinatoire en termes de recherche de structures introduit une étape supplémentaire qui peut induire une dégradation de la vitesse à laquelle les solutions sont trouvées, voire une dégradation de la qualité des solutions déterminées. En contrepartie, cette approche possède une qualité absente des approches traditionnelles de résolution de problèmes d'optimisation, elle met implicitement en œuvre un processus de maintien de structures. Ainsi, les capacités d'adaptation des systèmes à base de fourmis artificielles permettent aux structures construites d'évoluer en fonction des fluctuations qui affectent les éléments constitutifs du problème pendant la résolution de ce problème. De la même manière, moyennant un faible effort de modélisation, ces systèmes sont capables de traiter des problèmes caractérisés par des données incertaines ou changeantes. A ce titre cette approche plus que toute autre mérite que l'on s'y intéresse.

2.6. Annexe : *GraphStream*

Les codes ayant permis la conception et le test des différents systèmes à base de fourmis artificielles décrits dans ce chapitre ont été développés à l'aide de GraphStream[5] une librairie de graphes dynamiques. Ces codes, ainsi que les scripts à partir desquels ont été produites les images sont disponibles sur simple demande aux auteurs.

2.7. Bibliographie

[ALB 09] Albert P., Henocque L., Kleiner M., « Optimisation par colonie de fourmis pour la configuration en programmation par contraintes », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, des bases de l'optimisation aux applications industrielles*, vol. 1 de *Traité IC2*, chap. 5, Hermès, Paris, 2009.

[BAL 09] Balev S., Gaci O., Pigné Y., « Fourmis artificielles et bioinformatique (repliement de protéine, alignement multiple et sequençage par hybridation) », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, nouvelles directions pour une intelligence collective*, vol. 2 de *Traité IC2*, chap. 9, Hermès, Paris, 2009.

[BER 07] Bertelle C., Dutot A., Guinand F., Olivier D., « Organization Detection for Dynamic Load Balancing in Individual-based Simulations », *Multiagent and Grid Systems :*

5. http://graphstream.sourceforge.net.

special issue on Nature-Inspired Systems for Parallel, Asynchronous and Decentralized Environments, vol. 3, n° 1, p. 141–163, IOS Press, 2007.

[BLA 02] Błażewicz J., Formanowicz P., Guinand F., Kasprzak M., « Heuristic managing errors for dna-sequencing », *Bioinformatics*, vol. 18, p. 652–660, 2002.

[BON 99] Bonabeau E., Dorigo M., Theraulaz G., *Swarm Intelligence : from natural to artificial systems*, Oxford University Press, New York, 1999.

[DEN 90] Deneubourg J., Aron S., Goss S., Pasteels J., « The Self-organizing Exploratory Pattern of the Argentine Ant », *Journal of Insect Behavior*, vol. 3, p. 159–168, 1990.

[DUT 09] Dutot A., Olivier D., « La detection d'organisations dans les systèmes complexes par colonies de fourmis », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, nouvelles directions pour une intelligence collective*, vol. 2 de *Traité IC2*, chap. 5, Hermès, Paris, 2009.

[GAG 09] Gagné C., Gravel M., « OCF pour l'ordonnancement d'une chaîne d'assemblage automobile », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, des bases de l'optimisation aux applications industrielles*, vol. 1 de *Traité IC2*, chap. 8, Hermès, Paris, 2009.

[GAN 09] Gandibleux X., Jorge J., Delorme X., Rodriguez J., « Algorithme de fourmis pour mesurer et optimiser la capacité d'un réseau ferroviaire », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, des bases de l'optimisation aux applications industrielles*, vol. 1 de *Traité IC2*, chap. 9, Hermès, Paris, 2009.

[GRA 09] Gravel M., Gagné C., « Optimisation par colonie de fourmis pour la fabrication de barres d'aluminium », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, des bases de l'optimisation aux applications industrielles*, vol. 1 de *Traité IC2*, chap. 7, Hermès, Paris, 2009.

[GUI 09] Guinand F., Lafourcade M., « Fourmis artificielles et traitement de la langue naturelle », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, nouvelles directions pour une intelligence collective*, vol. 2 de *Traité IC2*, chap. 8, Hermès, Paris, 2009.

[HER 09] Hertz A., Zufferey N., « La coloration des sommets d'un graphe par colonies de fourmis », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, des bases de l'optimisation aux applications industrielles*, vol. 1 de *Traité IC2*, chap. 11, Hermès, Paris, 2009.

[KUN 94] Kuntz P., Snyers D., « Emergent colonization and graph partitioning », *Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, p. 494–500, Brighton, Royaume-Uni, 1994.

[LEN 09] Lenoir A., Monmarché N., « Des fourmis réelles aux fourmis artificielles », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, des bases de l'optimisation aux applications industrielles*, vol. 1 de *Traité IC2*, chap. 1, Hermès, Paris, 2009.

[LOV 93] Lovàsz L., « Random walks on graphs : a survey », *Combinatorics, special issue : Paul Erdös is Eighty*, vol. 2, p. 1–46, 1993.

[LYS 88] Lysov, Yu P., Florentiev V., Khorlin A., Khrapko K., Shik V., Mirzabekov A., « Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides »,

Doklady Akademii Nauk SSSR, vol. 303, p. 1508–1511, 1988.

[MOR 06] Moreau C., Bell C., Vila R., Archibald S., Pierce N., « Phylogeny of the Ants : Diversification in the Age of Angiosperms », *Science*, vol. 312, n° 5770, p. 101–104, avril 2006.

[SAN 09] Sandou G., Font S., Tebbani S., Mondon C., Hiret A., « Colonies de fourmis pour le problème d'affectation d'unités », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, des bases de l'optimisation aux applications industrielles*, vol. 1 de *Traité IC2*, chap. 6, Hermès, Paris, 2009.

[SEP 09] Sepchat A., Colas S., Clair R., Monmarché N., Gaucher P., Slimane M., « Les fourmis artificielles et le handicap : trois exemple de compensation de déficits d'autonomie et de jeux pour déficients visuels », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, nouvelles directions pour une intelligence collective*, vol. 2 de *Traité IC2*, chap. 6, Hermès, Paris, 2009.

[SET 97] Setubal J., Meidanis J., *Introduction to Computational Molecular Biology*, PWS Publishing Company, Boston, 1997.

[SIA 09] Siarry P., Dréo J., Monmarché N., « Les fourmis artificielles pour l'optimisation en variables continues », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, des bases de l'optimisation aux applications industrielles*, vol. 1 de *Traité IC2*, chap. 4, Hermès, Paris, 2009.

Chapitre 3

Tour d'horizon des problèmes combinatoires traités par les fourmis artificielles

Ce chapitre s'intéresse dans un premier temps aux différentes contributions scientifiques ayant utilisé la métaphore des fourmis pour résoudre des problèmes. Nous constatons qu'une filiation existe entre certains algorithmes qui héritent de leurs aînés en proposant des améliorations s'appliquant parfois mieux aux problèmes considérés.

Les différentes contributions ont vocation à résoudre des problèmes variés que l'on peut classer en différentes familles de problèmes. Les différents types de problèmes sont présentés dans la seconde partie du chapitre.

Dans le cadre du traitement de problèmes réels, des contraintes plus délicates et moins formalisées doivent être prises en compte telles que les environnements dynamiques et décentralisés, les incertitudes, les erreurs dans les données, etc. D'autre part, pour certains de ces problèmes, la formulation d'une fonction objectif peut s'avérer délicate, voire impossible. De plus, les objectifs peuvent être multiples, on parlera alors d'optimisation multi-objectif. Ces contraintes définissant des problèmes moins souvent abordés dans la littérature, dont les contours sont moins bien définis, sont traitées dans la troisième partie de ce chapitre.

3.1. Les principales approches à base de fourmis

Des centaines de contributions utilisant la métaphore des fourmis ont été proposées ces deux dernières décennies (voir dans le chapitre 1 la figure 1.10). Certaines

Chapitre rédigé par Antoine Dutot et Yoann Pigné.

propositions sont restées proches du modèle naturel, d'autres s'en sont éloignées, afin d'améliorer les résultats des problèmes traités. La méthode originale apparue pour la première fois en 1991 dans la thèse de M. Dorigo [DOR 92] consiste à proposer une métaheuristique pour l'optimisation combinatoire. Cette approche, plus tard formalisée sous le nom de ACO (*Ant Colony Optimization*) [DOR 97a], mêle l'idée de collaboration entre entités (les fourmis) explorant un espace de recherche et l'évaluation des solutions basée sur une fonction objectif. A partir de l'algorithme original, une multitude de variantes furent proposées et beaucoup de problèmes furent traités avec succès. Cette section revient sur quelques-unes des contributions fondatrices de l'approche ACO ; puis elle se termine sur d'autres propositions originales plus proches du modèle naturel.

3.1.1. *La famille des ACO*

La méthode proposée dans les travaux de M. Dorigo *et al.* [DOR 91, DOR 92, DOR 96] sous le nom de *Ant System* est une métaheuristique conçue pour l'optimisation de problèmes combinatoires. L'idée d'une population d'agents qui évoluent dans un environnement et coopèrent par le biais de communications asynchrones est clairement inspirée des colonies d'insectes sociaux comme les fourmis. La méthode hérite aussi des mécanismes classiques de recherche opérationnelle et d'intelligence artificielle tels que l'apprentissage par renforcement ou la construction gloutonne d'une solution guidée par une fonction objectif. *Ant System* fut la première des contributions du genre. Toutes les méthodes utilisant ce paradigme sont maintenant appelées des *ACO*. Chaque nouvelle contribution a apporté des améliorations au modèle original ou des spécialisations pour des problèmes particuliers.

3.1.1.1. *L'algorithme* Ant System

Ant System (AS) a été initialement conçu pour le problème du voyageur de commerce mais peut facilement être adapté à d'autres problèmes combinatoires. Le problème consiste à déterminer l'itinéraire le plus court d'un voyageur de commerce lui permettant de traverser chacune des n villes de sa tournée une fois et une seule. Ce problème est généralement modélisé par un graphe complet dont l'ensemble N des sommets représente les villes et l'ensemble E des arêtes les routes entre ces villes. Chaque arête est valuée par la distance d_{ij} qui sépare les villes i et j.

En théorie des graphes, ce problème correspond à la recherche d'un cycle hamiltonien de poids minimum qui est connue pour être NP-difficile au sens fort.

Dans la méthode *Ant System*, le graphe est parcouru par des fourmis artificielles qui se déplacent de manière à construire des cycles hamiltoniens. Chaque fourmi possède un comportement simple et une vision localisée. Le fonctionnement d'une colonie peut être schématisé ainsi :

– de manière itérative, chaque fourmi choisit son prochain sommet. Son choix dépend de la distance entre son sommet courant et le suivant, et de l'attractivité des liens qui est fonction de la quantité de phéromone associée au lien ;

– les villes déjà visitées sont supprimées du voisinage des fourmis, de manière à ce qu'elles visitent une seule fois chaque ville ;

– quand les fourmis ont terminé leur cycle, elles modifient l'attractivité des liens qu'elles ont visités en déposant des phéromones.

La méthode est itérative, une fourmi choisit au temps t quelle sera la ville visitée au temps $t + 1$. Ces temps sont des itérations. A chaque itération, toutes les fourmis du système se déplacent d'une ville à une autre. Au bout de n itérations (n est le nombre de villes) les fourmis ont construit un cycle et sont revenues à la ville de départ parce que le graphe est complet. Afin de produire des solutions réalisables (des cycles hamiltoniens), chaque fourmi k maintient une liste (notée $tabu_k$) des villes qu'elle a déjà visitées et les retire de son voisinage lors du choix de la prochaine ville.

L'attractivité des liens repose sur le dépôt de phéromones. Soit $\tau_{ij}(t)$ la quantité de phéromone sur le lien entre les villes i et j à l'instant t. La mise à jour de l'attractivité des liens est effectuée grâce à la formule suivante :

$$\tau_{ij}(t + n) = \rho.\tau_{ij}(t) + \Delta\tau_{ij} \tag{3.1}$$

ρ est un paramètre qui règle la conservation des phéromones d'un cycle au suivant. $(1 - \rho)$ correspond donc au facteur d'évaporation des phéromones sur les pistes.

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{3.2}$$

$\Delta\tau_{ij}^{k}$ est la quantité de phéromone que dépose la fourmi k sur l'arête (i, j) pendant le cycle courant. Le paramètre m est le nombre de fourmis dans le système.

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \text{si la fourmi } k \text{ est passée par } (i, j) \text{ pendant le cycle} \\ 0 & \text{sinon} \end{cases} \tag{3.3}$$

avec L_k la longueur du cycle effectué par la fourmi k et Q une constante.

Chaque fourmi choisit de manière probabiliste la prochaine ville à chaque itération en fonction de l'attractivité des liens et de la distance entre la ville courante et les villes voisines. Pour permettre aux fourmis de maximiser la qualité de leurs solutions, la distance est prise en compte dans la *visibilité* : $\eta_{ij} = 1/d_{ij}$:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in voisins_k} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}} & \text{si } j \in voisins_k \\ 0 & \text{sinon} \end{cases} \tag{3.4}$$

avec $voisins_k$ l'ensemble des villes, privé de celles déjà visitées par la fourmi k. α et β sont deux paramètres qui permettent de modifier l'importance relative des pistes de phéromone par rapport à la visibilité. Cette formule représente donc la probabilité à l'itération t que la fourmi k choisisse j comme prochaine destination à $(t + 1)$.

Une fois un cycle effectué, le système dispose de solutions au problème donné. Ces solutions peuvent être améliorées. D'autres cycles peuvent être effectués et tirer profit des cycles précédents grâce au marquage des pistes. L'algorithme général de *Ant System* pour la résolution du problème du voyageur de commerce peut alors être décrit :

1) initialisation :
   - $t = 0$,
   - $\tau_{ij}(t) = c$;

2) pour k allant de 1 à m :
   - positionner la fourmi k sur un sommet source S,
   - $tabu_k = \{S\}$;

3) pour t allant de 1 à $n - 1$ et k allant de 1 à m :
   - choix du prochain sommet S' en fonction de la formule (3.4),
   - déplacement de la fourmi k vers le sommet S',
   - ajout de S' à la liste tabou : $tabu_k = tabu_k \cup \{S'\}$;

4) à la fin du cycle, mise à jour des pistes de phéromone avec la formule (3.1) ;

5) si une condition d'arrêt ou un certain nombre de cycles ne sont pas atteints, alors retour à l'étape 2.

Les auteurs ont appliqué *Ant System* [DOR 96] à des instances répertoriées du problème du voyageur de commerce pour lesquelles des solutions sont connues. Ils ont ainsi pu comparer les résultats produits par leur algorithme avec des résultats optimaux, ainsi qu'avec des résultats obtenus par d'autres heuristiques [DOR 97b]. L'algorithme a ainsi montré son efficacité et la qualité des solutions produites.

Des optimisations du modèle ont été mises en œuvre et des variantes du modèle original ont été proposées. La suite de cette section présente les différentes améliorations apportées au modèle dans des contributions ne s'attaquant pas obligatoirement au même problème.

3.1.1.2. *Renforcement des meilleures solutions*

Dans le but d'accélérer la convergence de la méthode, les auteurs de *Ant System* proposent dans les mêmes contributions une modification du comportement du système : *Elitist Ant*. A la fin de chaque cycle, pendant l'étape de renforcement des pistes, ils proposent d'ajouter un second renforcement sur les liens de la meilleure solution jamais réalisée depuis le début de la recherche (*global best solution* S_{gb}). La formule de renforcement à la fin de chaque cycle est alors :

$$\tau_{ij}(t+n) = \rho.\tau_{ij}(t) + \Delta\tau_{ij} + \Delta\tau^*_{ij} \tag{3.5}$$

Seul le troisième terme diffère de la formule de renforcement classique (formule (3.1)).

$$\Delta\tau^*_{ij} = \begin{cases} \sigma\frac{Q}{L_{gb}} & \text{si } (i,j) \text{ appartient à } S_{gb} \\ 0 & \text{sinon} \end{cases} \tag{3.6}$$

avec L_{gb} la longueur de la meilleure solution globale S_{gb} et σ un paramètre supérieur à 1, qui permet d'ajuster l'importance du renforcement. L'idée est de rendre encore plus attractifs les liens participant aux meilleures solutions afin qu'ils soient utilisés dans la construction de nouvelles solutions.

Dans le prolongement de cette idée, l'algorithme *Rank-Based Ant System* [BUL 99] propose de classer les fourmis, après chaque cycle, en fonction de la qualité de leur solution, pour sélectionner les individus qui vont participer au renforcement. Ainsi la liste ordonnée des ω meilleures fourmis va être construite, le renforcement va être proportionnel à leur classement (meilleur est le classement d'une fourmi, plus important sera le renforcement associé à sa solution). Pour ne pas interférer avec le renforcement élitiste précédent, les auteurs notent que ω ne doit pas être supérieur à σ, qui est le facteur multiplicatif appliqué à la meilleure solution globale et rien ne devrait être supérieur à cette solution. ω est fixé à $\sigma - 1$. La formule de renforcement dans cet algorithme ressemble à la formule de renforcement de *Elitist Ant* (formule (3.5)), à la différence du second terme $\Delta\tau_{ij}$, qui vaut maintenant :

$$\begin{aligned} \Delta\tau_{ij} &= \sum_{\mu=1}^{\sigma-1} \Delta\tau^{\mu}_{ij} \\ \Delta\tau^{\mu}_{ij} &= \begin{cases} (\sigma-\mu)\frac{Q}{L_{\mu}} & \text{si } (i,j) \text{ appartient au cycle de la } \mu^{ème} \text{ fourmi,} \\ 0 & \text{sinon} \end{cases} \end{aligned} \tag{3.7}$$

Dans [GAM 95], les auteurs proposent un algorithme proche de *Ant System*, mais se focalisant sur la partie apprentissage par renforcement. *Ant-Q* propose de n'appliquer la formule de renforcement des pistes que sur les liens utilisés lors du cycle courant, ainsi les liens non utilisés ne sont pas mis à jour et seules les solutions générées sont renforcées. De plus, comme dans *Elitist Ant*, *Ant-Q* propose un second renforcement basé sur les meilleures solutions déjà produites. Deux variantes concernant le

choix des pistes à renforcer sont proposées : la version *global-best*, qui correspond à *Elitist Ant*, en sélectionnant la meilleure solution depuis le début de la recherche (S_{gb}) et la version *iteration-best*, où la solution utilisée pour le second renforcement est la meilleure du cycle qui vient d'être effectué (*iteration best solution* S_{ib} ou solution locale).

L'algorithme *MAX-MIN Ant System* [STU 00], inspiré de *Ant System*, propose entre autres d'utiliser la formule de renforcement (3.1) en modifiant $\Delta\tau_{ij}$ de manière à ce que seule la meilleure solution (globale ou locale) soit renforcée :

$$\Delta\tau_{ij} = \begin{cases} \frac{1}{L_{best}} & \text{si } (i, j) \text{ appartient à la solution } S_{best}, \\ 0 & \text{sinon} \end{cases} \tag{3.8}$$

où L_{best} est la longueur de la meilleure solution globale (S_{gb}) ou locale (S_{ib}). Ce mécanisme utilisé seul tend à ne renforcer que les bonnes solutions et à faire disparaître l'attractivité des liens plus rarement visités. Ceci permet de faire converger rapidement vers une solution, mais provoque une mauvaise exploration de l'espace de recherche. *MAX-MIN Ant System* règle ce problème en utilisant un mécanisme de bornes inférieures et supérieures, pour les phéromones empêchant la disparition de l'attractivité de certaines arêtes, ou la surreprésentation d'autres.

3.1.1.3. *Le paramètre d'évaporation des pistes*

Plusieurs contributions s'intéressent au mécanisme d'évaporation des pistes, censé éviter la convergence prématurée de la méthode, tout en assurant une bonne exploitation de l'espace de recherche. En effet, l'évaporation des pistes lisse les différences d'attraction entre les liens, augmente la probabilité de visite de certains liens peu utilisés, améliore l'exploration de l'espace de recherche et tend à éviter la convergence prématurée de l'algorithme vers les premières solutions trouvées. L'effet secondaire indésirable est qu'une trop forte évaporation annule l'effet attractif des phéromones, fait perdre la mémoire collective des bonnes solutions trouvées auparavant et réduit le processus de recherche des fourmis à un parcours aléatoire de l'environnement ou à une recherche uniquement guidée par l'heuristique locale (la visibilité). L'évaporation dans les algorithmes ACO est ajustée grâce au paramètre ρ de la formule (3.1).

Dans ANTS [MAN 99a], les auteurs n'utilisent pas le paramètre d'évaporation ρ et proposent un autre mécanisme pour modifier l'attractivité des pistes. Le mécanisme n'entre en jeu qu'après un certain nombre d'étapes de calcul, qui permettent de construire plusieurs (k) solutions et d'obtenir une estimation de la longueur moyenne d'une solution $L_{\overline{S}}$. Après cette phase d'initialisation, les pistes des nouvelles solutions générées sont renforcées, proportionnellement à leur écart par rapport à la solution moyenne $\overline{S}$. Après avoir été comparée à la solution moyenne, chaque nouvelle solution est utilisée pour mettre à jour $L_{\overline{S}}$, qui est la longueur moyenne des k dernières solutions produites. Cette valeur évolue donc au cours du traitement. Seules les arêtes

appartenant à des solutions générées pendant le cycle courant sont renforcées. Ainsi, pour chaque solution s, le renforcement est :

$$\begin{aligned} \tau_{ij}(t+n) &= \tau_{ij}(t) + \Delta\tau_{ij} \\ \Delta\tau_{ij} &= \tau_0.\left(1 - \frac{L_s - LB}{L_{\overline{S}} - LB}\right) \end{aligned} \tag{3.9}$$

où L_s est la solution courante et LB une borne inférieure (*Lower Bound*) connue pour le problème considéré. Ce mécanisme permet de modifier les quantités de phéromone par une valeur comprise entre τ_0 et $-\tau_0$.

Dans *Best-Worst Ant System* [COR 00], les auteurs s'intéressent aussi à la sensibilité du paramètre ρ. Ici, le mécanisme d'évaporation n'est pas supprimé, mais il est restreint, de manière à minimiser son impact. Dans ce modèle, seules les arêtes de la meilleure solution locale (S_{ib}) sont renforcées, et seules les arêtes de la plus mauvaise solution ne faisant pas partie de S_{ib} subissent l'évaporation $\tau_{ij} = \rho.\tau_{ij}$.

3.1.1.4. *Renforcements global et local des pistes de phéromone*

La solution apportée par l'algorithme *Ant Colony System* (ACS) pour éviter le problème de mauvaise exploration, tout en gardant les améliorations de convergence du renforcement élitiste, est de définir deux types de renforcements, un local et un global [DOR 97b, DOR 97c]. Le renforcement global est celui classiquement défini par *Ant System* et repris par les autres méthodes, pour favoriser les liens appartenant à de bonnes solutions déjà trouvées. Il est élitiste et propose qu'à chaque cycle, seule la fourmi ayant produit la meilleure solution renforce sa piste de manière inversement proportionnelle à la longueur de celle-ci. C'est donc la meilleure solution locale (pour l'itération courante) qui est privilégiée (S_{ib}), de sorte que les liens de cette solution soient renforcés de manière proportionnelle : $\Delta\tau_{ij} = 1/L_{ib}$. Ainsi, plus les solutions sont bonnes, plus elles sont renforcées. Le renforcement local, quant à lui, a pour vocation de diminuer l'attractivité des liens très visités et donc de redonner de l'attractivité aux liens peu visités. Il consiste en un renforcement systématique et constant des liens traversés par toutes les fourmis. Ce renforcement est dit local car les fourmis renforcent les liens à l'instant où elles les traversent. De plus, ce renforcement est constant (et fait tendre la valeur vers τ_0, valeur initiale des phéromones), il n'est proportionnel à aucune solution et ne peut pas l'être car, au moment de ce renforcement, la fourmi n'a pas encore construit de solution.

3.1.1.5. *Exploration* versus *exploitation*

Toujours dans le souci d'améliorer l'exploration de l'espace de recherche, tout en tirant profit des bonnes solutions générées, les auteurs de *Ant-Q* [GAM 95] s'intéressent à d'autres manières de mettre en œuvre la procédure de choix du prochain sommet pour chaque fourmi. La formule classique (formule (3.4)) est avant tout basée

sur une exploration aléatoire influencée par les solutions déjà trouvées et par l'heuristique locale. La proposition ici est de mieux exploiter les solutions déjà produites. Ainsi, les auteurs proposent la formule suivante, pour déterminer le sommet s que la fourmi k située sur le sommet r au temps t visitera au temps $t + 1$:

$$s = \begin{cases} \arg \max\limits_{j \in voisins_k} \{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}\} & \text{si } p \leq p_0 \\ S & \text{sinon} \end{cases} \tag{3.10}$$

avec p une valeur choisie aléatoirement de manière uniforme dans [0, 1] et p_0 un paramètre ($0 \leq p_0 \leq 1$). S est une variable aléatoire ayant pour but de déterminer un sommet suivant s. Cette formule permet, grâce au paramètre p_0, d'équilibrer l'exploitation des résultats précédemment trouvés et l'exploration aléatoire de l'environnement. La formule $\arg\max_{j \in voisins_k} \{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}\}$ détermine en effet le sommet le plus attractif, alors que S est relatif à l'exploration aléatoire de l'environnement. S et p_0 doivent être définis, les auteurs proposent trois versions :

1) S est une variable aléatoire uniforme dans [0, 1], dans ce cas, le choix de s est dit *pseudo-aléatoire* et partagé entre une exploitation des meilleurs résultats et une exploration aléatoire pure ;

2) S correspond à la formule (3.4) de *Ant System* et le choix de s est partagé entre l'exploitation des meilleurs résultats et une exploration aléatoire biaisée par les résultats existants. Ce mécanisme est dit *pseudo-aléatoire proportionnel* ;

3) S correspond à la formule (3.4) et $p_0 = 0$. Cela signifie que S est systématiquement choisie pour déterminer le prochain sommet s. Cette formule correspond au mécanisme de sélection de *Ant System*, on le dit *aléatoire proportionnel*.

Les tests effectués par les auteurs montrent que le modèle *pseudo-aléatoire proportionnel* est supérieur aux deux autres pour certaines valeurs de p_0. C'est ce modèle de sélection qui est repris dans l'algorithme *Ant Colony System* [DOR 97b, DOR 97c].

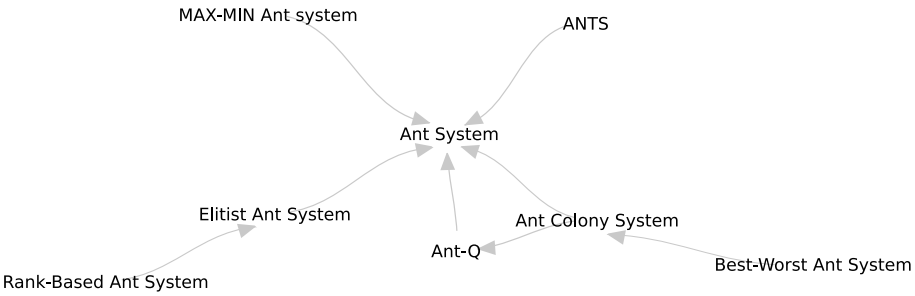


Figure 3.1. *Graphe d'héritage entre les ACO. Un lien orienté de B vers A signifie que B hérite de l'algorithme A pour construire sa propre contribution*

Pour aider à la compréhension des différents modèles présentés ici, la figure 3.1 montre un graphe d'héritage qui illustre les relations entre méthodes.

Ainsi, même si les fourmis n'améliorent pas nécessairement les méthodes plus anciennes, ces techniques montrent que l'on peut se rapprocher des meilleures performances.

3.1.2. Ant-Based Control, *fidèle à la métaphore fourmi*

Ant-Based Control (ABC) [SCH 97] est un algorithme très proche de la métaphore fourmi qui exploite l'idée d'un système composé d'entités autonomes, au fonctionnement totalement décentralisé. Cet algorithme est appliqué à un problème naturellement distribué de répartition de charge dans un réseau de télé . Ce réseau est composé de routeurs censés assurer la communication de bout en bout entre des paires de clients. Chaque routeur possède une capacité (un nombre maximum de communications simultanées). Le but est non seulement de trouver des routes relativement courtes, mais aussi d'équilibrer la charge et d'éviter les congestions.

L'idée générale est que des agents (fourmis) sont dispersés dans ce réseau, qu'ils s'y déplacent et qu'ils mettent collectivement à jour les tables de routage dans les routeurs du réseau. Ces tables servent à router les appels téléphoniques. Les appels modifient la charge du réseau et influencent indirectement le comportement des fourmis. Il faut noter que, dans ce problème, il n'est pas possible d'évaluer globalement les solutions produites et de les partager dans tout le réseau. Du fait de la nature distribuée du réseau, les décisions de routage et les modifications de tables de routage se font localement, sans évaluation centralisée.

3.1.2.1. *Tables de phéromone*

Dans les problèmes de routage classiques, chaque nœud possède une table de routage qui permet de déterminer, pour chaque sommet du réseau, quelle est la route à prendre. Cette route est déterminée la plupart du temps de proche en proche : la table de routage d'un sommet A indique lequel des voisins de A doit être traversé pour atteindre un sommet destination.

Dans ABC, les tables de routage sont remplacées par des tables de phéromone. Celles-ci n'indiquent plus, pour chaque nœud, vers quel voisin aller, mais la probabilité pour chaque voisin d'être le suivant. Les tables de phéromone ne contiennent donc pas explicitement des valeurs numériques quantitatives, comme dans les *ACO*, mais l'effet produit reste identique puisque les phéromones des *ACO* sont exprimées en probabilités, lors de la procédure de choix des fourmis.

Ainsi, chaque sommet possède, pour chaque nœud du réseau, une table de probabilités avec autant d'entrées que le sommet a de voisins. La somme des probabilités

des voisins d'un sommet, pour une destination donnée, est un. Les fourmis choisissent leur prochain nœud à l'aide d'un tirage aléatoire proportionnel aux probabilités de la table de routage : les fourmis tirent un nombre aléatoire, selon la distribution de probabilité donnée par la répartition des phéromones. Ce nombre indique ensuite le noeud voisin à emprunter.

Les fourmis sont lancées dans le réseau, indépendamment des appels téléphoniques, afin d'entretenir les tables de phéromone.

Les appels téléphoniques sont routés en utilisant les tables de phéromone et en sélectionnant systématiquement les voisins avec les meilleures probabilités.

3.1.2.2. *Renforcement des tables de routage*

Les fourmis, en se déplaçant, mettent à jour les tables de phéromone. Une fourmi choisissant d'emprunter le voisin v du sommet s, pour se rendre à la destination d, va mettre à jour la table de phéromone de d dans s de la manière suivante :

$$\begin{aligned} p_d^s(v) &= \frac{p_d^s(v) + \Delta p}{1 + \Delta p} \\ p_d^s(u) &= \frac{p_d^s(u)}{1 + \Delta p} \qquad \forall u \in voisins(s) \end{aligned} \tag{3.11}$$

avec $voisins(s)$ l'ensemble des voisins de s et Δp le renforcement des probabilités à définir. Cette formule assure que la somme des probabilités des voisins de s pour la destination d vaut un.

3.1.2.3. *Plus court chemin et équilibrage de charge*

Pour permettre la construction de plus courts chemins, il faut que les tables de phéromone guident les fourmis de préférence vers les plus courtes pistes. Pour faire transparaître cette idée de longueur des chemins, le dépôt des fourmis est proportionnel au trajet déjà effectué par la fourmi, de sorte que celles ayant parcouru un long chemin influencent moins les tables de routage que les fourmis ayant parcouru de plus courts chemins. Si a est l'âge, en nombre de sauts, d'une fourmi, les auteurs proposent, pour les phéromones, la valeur de renforcement suivante : $\Delta p = \frac{0.08}{a} + 0,005$. Ce mécanisme favorise donc les chemins les plus courts.

L'équilibrage de la charge est mis en œuvre en ralentissant les fourmis sur les sommets congestionnés (dont la bande passante arrive à saturation). Le fait de bloquer une fourmi sur un sommet congestionné empêche le dépôt des phéromones, n'améliore donc pas l'attractivité de ce nœud et fait aussi diminuer l'affluence des communications *via* celui-ci. Expérimentalement, le délai en nombre d'étapes pendant lequel les fourmis sont bloquées sur les sommets congestionnés est fixé à $\lfloor 80.e^{-0.075.s} \rfloor$, avec s la capacité restante sur le sommet. La capacité, elle, représente le nombre d'appels simultanés possibles sur un sommet donné.

3.2. Les différentes classes de problèmes abordés

3.2.1. *Les problèmes d'affectation*

Les problèmes d'affectation consistent en la recherche de couples dans un graphe biparti de manière à optimiser (maximiser ou minimiser) la somme des poids des arêtes sélectionnées dans le graphe. Une illustration de ce problème est l'affectation d'un ensemble de tâches à un ensemble d'employés dans une entreprise. Chaque employé peut effectuer une seule tâche à la fois et possède des compétences différentes vis-à-vis des différentes tâches. Le but est de maximiser le nombre de tâches effectivement réalisées.

On peut aussi formaliser le problème avec deux ensembles A et B et une matrice M de coûts entre les éléments de A et B. Le problème consiste alors à trouver la fonction $f : a \rightarrow b$ qui maximise ou minimise la somme :

$$\sum_{a \in A} M[a, f(a)] \tag{3.12}$$

Plusieurs problèmes connus d'affectation ont été traités à l'aide d'algorithmes de fourmis. Parmi ceux-ci, le problème d'affectation quadratique ou le problème de coloration de graphe.

3.2.1.1. *Affectation quadratique*

Le problème d'affectation quadratique considère un ensemble U de n unités et un ensemble S de n sites. Il existe des flux entre les différentes unités et une distance entre les sites. La fonction $f : U \times U \rightarrow \mathbb{R}$ définit le flux entre deux unités de U. La fonction $d : S \times S \rightarrow \mathbb{R}$ donne la distance entre les sites. Soit $\psi : U \rightarrow S$ la fonction qui, à chaque unité, associe un site. Le problème est de définir quel site associer à quelle unité (quelle fonction ψ), de façon à minimiser la somme des produits des distances et des flux :

$$\sum_{u \in U} \sum_{v \in U} f(u, v).d(\psi(u), \psi(v)) \tag{3.13}$$

Les algorithmes à base de fourmis, utilisés pour le problème d'affectation quadratique, le modélisent sous forme d'un graphe $G = (V, E)$ où V est l'ensemble des sommets représentant l'affectation d'une entité à un site et E est l'ensemble des liens. Le graphe est complet.

La famille des *ACO* est largement utilisée pour résoudre ce problème. L'idée générale de tous les *ACO* tient en quelques principes. Le sens de l'affectation est fixé par avance : les unités sont affectées aux sites (ou le contraire). Dans la suite, on suppose que les unités sont affectées aux sites. Ensuite, l'ordre dans lequel les unités vont

être traitées est défini. Cet ordre peut être aléatoire, fixé par avance ou non. Puis, les fourmis construisent des chemins dans le graphe G. Chaque chemin représente une affectation valide (une solution réalisable) dans l'espace de recherche. Enfin, les solutions générées sont améliorées grâce à des heuristiques de recherche locale. Ces solutions sont finalement marquées de pistes de phéromone et permettent d'influencer la construction des futures solutions générées. Le processus est itératif autour de ce mécanisme.

Parmi les différents *ACO* proposés, on retrouve une adaptation de *Ant System*, nommée AS-QAP [MAN 99b], qui est la première adaptation du modèle au problème. Les algorithmes ANTS [MAN 99a] et *MAX-MIN Ant System* [STU 00] déjà présentés traitent également ce problème.

Hybrid Ant Sytem (HAS-QAP) [GAM 99a] est une proposition inspirée des *ACO* mais fonctionnant différemment. Les pistes de phéromone ne sont pas utilisées pour fabriquer de nouvelles solutions mais pour améliorer des solutions déjà existantes. A chaque fourmi est associée une solution complète. Le but est d'améliorer cette solution, en effectuant des permutations de sites entre les unités. A chaque cycle, sur chaque solution, deux unités sont choisies en fonction d'une formule aléatoire proportionnelle aux phéromones présentes. Les sites de ces deux unités sont échangés. La solution ainsi générée est souvent moins bonne que la solution d'origine mais elle est ensuite optimisée, en utilisant une heuristique locale. Ce mécanisme vise à faire sortir la solution d'un optimum local, grâce à l'échange de sites dans un premier temps, puis à l'optimiser, grâce à l'heuristique locale, dans un second temps.

Dans [TAL 01], les auteurs montrent l'efficacité de leur algorithme inspiré des ACO par rapport à d'autres métaheuristiques pour le QAP. Cet algorithme surpasse même les résultats de HAS-QAP.

3.2.1.2. *Coloration de graphe*

Ce problème consiste à affecter à chaque nœud d'un réseau une couleur, de sorte que deux voisins n'aient pas la même couleur et que le nombre total de couleurs utilisées dans le graphe soit minimal. Ce nombre ne peut être inférieur à la taille de la clique maximale (plus grand sous-graphe complet) du graphe. Ce nombre minimal de couleurs est appelé nombre chromatique du graphe, il est apparenté à la notion de stable. Un stable est un sous-ensemble d'un graphe dont les sommets ne sont pas adjacents. La coloration d'un graphe est donc une partition de l'ensemble des sommets en stables. La recherche d'une coloration minimale consiste à minimiser le nombre de stables.

Dans ANTCOL [COS 97], les auteurs considèrent que les nœuds du réseau à colorer sont des unités et les couleurs sont des ressources. La qualité d'une solution

dépend à la fois du choix des couleurs pour les nœuds mais aussi de l'ordre de sélection de ceux-ci. La construction d'une solution se fait donc en deux temps, d'abord la sélection du prochain sommet à colorer puis la sélection de la couleur.

Les auteurs observent qu'une procédure de choix de la séquence de nœuds à colorer, s'adaptant à la coloration partiellement réalisée, donne de meilleurs résultats qu'une liste fixée à l'avance. Ainsi, ils utilisent deux heuristiques de choix. La première sélectionne de préférence les nœuds avec le degré de saturation le plus élevé. Le degré de saturation d'un sommet est le nombre de couleurs différentes utilisées dans le voisinage de ce sommet. La seconde heuristique tente de maximiser la taille des stables. Quand un sommet est sélectionné et que sa couleur est attribuée, la méthode sélectionne d'autres nœuds de manière à maximiser la taille du stable courant.

Le choix des couleurs est conditionné par les solutions déjà proposées. Ainsi, la probabilité de choix d'une couleur donnée (d'un stable) pour un sommet est inversement proportionnelle au nombre chromatique des solutions passées où le sommet en question était affecté de cette couleur.

ANTCOL est un algorithme qui calcule une coloration exacte, les solutions produites respectent les contraintes de coloration (pas de voisins de couleur identique). D'autres auteurs se sont intéressés à une version moins rigide du problème de coloration. Ainsi, les auteurs de [SHA 01] proposent un algorithme dont le but n'est pas de construire des solutions, mais d'améliorer itérativement une solution non valide, jusqu'à ce qu'elle le devienne.

Nous renvoyons le lecteur au chapitre 11 pour des explications plus détaillées.

3.2.2. *Les problèmes d'ordonnancement*

Les problèmes d'ordonnancement consistent à organiser l'exécution d'un ensemble de tâches sur un ensemble de ressources. Les tâches ont des durées et des contraintes de précédence entre elles. Le but dans ces problèmes est souvent de minimiser la date de fin d'exécution de l'ensemble des tâches.

Le problème d'atelier à cheminements multiples (*job-shop problem*) se pose lorsque plusieurs pièces ont besoin d'être usinées sur plusieurs machines dans un ordre donné. Le but est de trouver une séquence qui minimise la durée totale de production. Dans [ZWA 99], les auteurs appliquent l'algorithme *Ant System* en proposant une modélisation du problème à l'aide d'un graphe. Si n est le nombre de tâches et m le nombre de machines, la matrice $T(n \times m)$ représente, pour chaque tâche, son ordre d'utilisation des machines. Les cases de la matrice correspondent aux nœuds du graphe. Les nœuds sont reliés pour former un graphe complet. Les fourmis construisent

ensuite des chemins dans ce graphe qui correspondent à des séquences. Une liste tabou des nœuds déjà visités, ainsi qu'une liste des nœuds disponibles, sont maintenues, pour garantir la production de solutions réalisables.

Le problème de retard pondéré (*weighted tardiness*) dans le contexte d'ateliers à cheminement unique concerne la définition de l'ordre d'exécution de tâches sur une seule machine, de sorte que la somme des durées pondérées des tâches soit minimisée. Dans [BES 00], les auteurs s'attaquent à ce problème en utilisant *Ant Colony System*.

Beaucoup d'autres problèmes d'ordonnancement ont été traités avec des algorithmes de fourmis, comme par exemple dans le cas d'un problème bicritère, dans un atelier de type *Flowshop* [TK 02] où les fourmis sont utilisées pour optimiser un critère (la somme des retards), pendant qu'un algorithme déterministe (Johnson) est utilisé pour ne conserver que les solutions optimales pour le critère $C_{\max}$ (temps total de réalisation).

3.2.3. *Les problèmes de partitionnement*

Les problèmes d'ensembles en général désignent des problèmes qui consistent à sélectionner un ou des sous-ensembles à partir d'un ensemble de départ, de manière à optimiser une fonction d'évaluation. Le choix d'un élément de l'ensemble a une conséquence sur les autres, il y a donc des relations de dépendance entre les éléments.

3.2.3.1. *Le problème du sac à dos*

Ce problème d'ensembles consiste à sélectionner des objets possédant un poids p ainsi qu'une valeur v. Le but est de maximiser la valeur totale du sous-ensemble sélectionné, tout en considérant une contrainte sur le poids total à ne pas dépasser. Ce problème est similaire au remplissage d'un sac à dos par des objets. Ceux-ci ont un poids et une valeur et le poids total qu'il est possible de mettre dans le sac à dos est limité. Les objets o susceptibles d'être sélectionnés sont numérotés de 1 à n. Une sélection d'objets peut être codée sous la forme d'un vecteur binaire. Ainsi l'objet o_i vaut 1 s'il est sélectionné et 0 sinon. La valeur totale d'une sélection, qui est à maximiser, vaut $V = \sum_{i=1}^{n} o_i.v_i$. Cette valeur est contrainte par le poids de la sélection $P = \sum_{i=1}^{n} o_i.p_i$ qui ne doit pas dépasser la capacité du sac à dos P_c.

Le problème peut être généralisé à plusieurs contraintes. Si l'on considère m contraintes par objet, le poids d'un objet o_i devient $P_i = \sum_{j=1}^{m} p_i^j$. Le problème peut donc être formulé ainsi :

$$\begin{aligned} &\text{maximisation de } \sum_{i=1}^{n} o_i.v_i \\ &\text{sous la contrainte de } \sum_{i=1}^{n}\sum_{j=1}^{m} o_i.p_i^j \end{aligned} \tag{3.14}$$

La première contribution concernant les fourmis pour le problème du sac à dos est celle de G. Bilchev et Y. Parmee [BIL 96]. Celle de G. Leguizamon et Z. Michalewicz [LEG 99] mime celle de *Ant System* avec le problème du voyageur de commerce. La principale différence avec le problème du voyageur de commerce est que, dans celui-ci, le nombre d'éléments (de villes) à sélectionner est constant, il faut sélectionner toutes les villes et seul l'ordre de sélection compte. Dans le problème du sac à dos, le nombre d'éléments est variable et l'ordre de sélection n'importe pas. Dans *Ant System*, pour le problème du voyageur de commerce, la notion de cycle est importante. Après la sélection de tous les éléments, l'évaluation des solutions, ainsi que le renforcement des pistes de phéromone, est effectué. Pour coller au comportement de *Ant System*, les auteurs proposent une notion de cycle avec un nombre d'itérations fixé en paramètre. Après ce nombre d'itérations, chaque fourmi va évaluer sa solution. Une solution est un remplissage itératif de sac à dos. Le choix des objets dépend des phéromones déposées sur les objets, ainsi que de l'heuristique de recherche locale qui donne la désirabilité d'un objet, en fonction du profit qu'il génère et des coûts qu'il engendre. La principale innovation consiste à effectuer le dépôt de phéromone sur les sommets du graphe qui représentent les objets, plutôt que sur les liens, comme cela est fait avec le problème du voyageur de commerce.

3.2.3.2. *Le problème d'ensemble indépendant maximal*

Le problème d'ensemble indépendant maximal consiste, dans un graphe $G = (V, E)$, à trouver le sous-ensemble maximal de V, tel que deux sommets de cet ensemble ne soient pas reliés par une arête dans le graphe. Soit $V^* \subset V$, $\forall u, v \in V^*, (u, v) \notin E$ et $|V^*|$ est maximal. Les mêmes auteurs que précédemment pour le problème de sac à dos proposent la même adaptation de *Ant System* [LEG 01]. Pour ce nouveau problème, seule l'heuristique de recherche locale est différente.

3.2.3.3. *Clique maximale dans un graphe*

Une clique dans un graphe est un sous-ensemble de sommets, tous connectés les uns aux autres. Une clique est donc un sous-graphe complet du graphe étudié. Le problème de la clique maximale consiste à trouver le plus grand sous-graphe complet dans un graphe. Le cardinal de ce sous-graphe correspond d'ailleurs au nombre chromatique vu plus haut.

Dans [FEN 03], S. Fenet et C. Solnon proposent *Ant-Clique*, un algorithme dérivé de *MAX-MIN Ant System*, qui construit itérativement des cliques. Chaque fourmi de la colonie construit une clique, qui est évaluée, et la plus grande clique construite est renforcée. L'originalité de cette approche est qu'aucune heuristique locale n'est utilisée pour construire les cliques, seules les pistes de phéromone influencent les fourmis pour la sélection des sommets. De ce fait, l'initialisation des pistes de phéromone est cruciale. Les auteurs proposent deux méthodes d'initialisation, d'abord ils reprennent l'une des méthodes de *MAX-MIN Ant System*, qui consiste à initialiser les arêtes avec

la quantité maximale de phéromone, pour assurer une diffusion maximale de l'exploration. L'autre solution consiste à initialiser les arêtes en utilisant les cliques obtenues lors d'un premier échantillonnage de l'espace de recherche.

3.3. Les problèmes multi-objectifs, la dynamique et l'incertitude

Les grandes familles de problèmes présentés dans la section précédente montrent des algorithmes s'appliquant à des problèmes généraux, aux contraintes clairement définies. Cette section s'intéresse à des problèmes concrets souvent soumis à plusieurs contraintes qui ne peuvent pas toujours être formulées simplement.

Prenons l'exemple du routage de véhicules présenté plus loin. Ce problème peut être résolu par un processus construit de manière centralisée qui cherche à optimiser une solution selon une contrainte (minimiser les retards par exemple) tout en connaissant les différentes contraintes (durées de trajets, nature des demandes). Dans la réalité, un tel service de routage rencontre d'autres problèmes. En effet, il est souvent pertinent de prendre en compte plusieurs objectifs à optimiser (répartition de charge entre les véhicules, minimisation du temps de tournée, minimisation du nombre de véhicules). Ensuite, l'environnement dans lequel évoluent les véhicules est naturellement distribué, et des contraintes locales (embouteillage), imprévisibles et invisibles de manière globale, peuvent localement remettre en question la solution et forcer une décision locale (au niveau du véhicule). Enfin, des demandes aléatoires ou mal définies peuvent remettre en cause des fonctions d'optimisation statiques.

Cet exemple recèle de nombreuses contraintes non triviales, difficiles à modéliser et à traiter. La notion de contraintes multiples nous entraîne vers l'optimisation multi-objectif. Le caractère aléatoire de certaines demandes introduit la notion d'incertitude dans le modèle. Enfin, l'environnement changeant localement de manière imprévisible, est distribué et dynamique. Dans ce contexte, l'optimisation globale à l'aide d'une fonction objectif semble difficile à mettre en œuvre. Ces contraintes sont étudiées dans la suite.

3.3.1. *Les problèmes multi-objectifs*

Les problèmes multi-objectifs [ANG 09] caractérisent des problèmes où plusieurs contraintes doivent être prises en compte pour construire une solution. L'existence de plusieurs contraintes implique que toutes les solutions ne peuvent pas être comparées sur la base d'une fonction objectif simple. En effet, il est possible qu'une solution soit plus efficace qu'une autre pour certains objectifs, mais pas pour d'autres. On définit alors une relation d'ordre partiel entre les solutions d'un tel problème. Cette relation définit la notion de dominance. Une solution en domine une autre si elle améliore au moins un objectif, sans pour autant détériorer les autres.

Deux approches principales existent. Une méthode générale consiste à reformuler le problème multi-objectif pour le transformer en problème mono-objectif. Une fonction qui pondère les différents objectifs est alors créée et est utilisée pour évaluer les solutions produites. L'autre approche consiste à optimiser tous les objectifs indépendamment, en construisant un ensemble de solutions répondant aux contraintes de dominance, c'est un ensemble ou front de *Pareto*.

3.3.1.1. *Le problème de routage de véhicules*

La première contribution est le résultat du travail de L.M. Gambardella, E. Taillard et G. Agazzi [GAM 99b] pour le problème de routage de véhicules avec fenêtres de temps. Dans ce problème, un dépôt contient des véhicules, ainsi que des ressources qui doivent être distribuées à des clients à l'aide des véhicules. Le problème est de définir des tournées pour les véhicules qui optimisent plusieurs critères et respectent certaines contraintes, comme la capacité des véhicules ou leur nombre. Parmi les critères, il y a la minimisation du nombre de véhicules (ou du nombre de tournées), la minimisation du temps total de trajet ou encore la minimisation des délais de livraison.

Dans [GAM 99b], les auteurs proposent un algorithme de fourmis (nommé MACS-VRPTW) qui optimise deux critères, le nombre de véhicules et le temps total de trajet. La construction est itérative et deux colonies de fourmis différentes sont utilisées, une pour chaque critère. Les auteurs utilisent les mêmes mécanismes que *Ant System* dans chaque colonie. Les deux colonies utilisent donc des matrices de phéromone différentes mais partagent une même meilleure solution globale pour les mises à jour de pistes de phéromone. De manière itérative, les solutions sont générées et améliorées. Pour la colonie chargée d'optimiser le nombre de véhicules, chaque itération fait diminuer le nombre maximum de véhicules utilisés. La seconde colonie tente d'améliorer les chemins trouvés par la précédente colonie en diminuant le temps total de trajet. Cette méthode n'est néanmoins pas réellement une approche multi-objectif car, dans ce mécanisme, c'est la première optimisation (le nombre de véhicules) qui guide la recherche. C'est d'abord ce critère qui est optimisé, puis l'autre, en fonction des solutions du précédent.

Dans [BAR 03], les auteurs proposent une amélioration de *MACS-VRPTW*, qui cherche à optimiser trois critères de manière indépendante. Ces critères sont la minimisation du nombre de véhicules ($F1$), du temps total de trajet ($F2$) et des dates de livraison ($F3$). Une seule colonie est utilisée et les critères $F2$ et $F3$ sont considérés dans la formule de choix du prochain sommet. Le nombre de véhicules est modélisé en multipliant artificiellement le nombre de sommets correspondant aux dépôts. Le mécanisme est réellement multi-objectif, même si le critère $F1$ est fixé dès l'initialisation et ne peut être modifié qu'en réinitialisant la colonie et en retirant au graphe un sommet correspondant à un véhicule dans le problème.

3.3.1.2. *Le problème de sélection de portefeuille financier*

Le problème de portefeuille financier consiste à sélectionner un sous-ensemble de projets pour constituer un portefeuille d'investissement. Plusieurs contraintes doivent être respectées et plusieurs objectifs sont considérés. Dans [DOE 01, DOE 04], les auteurs proposent d'utiliser une adaptation de *Ant Colony System* pour construire des solutions sous forme de fronts de *Pareto*, avec K critères pour ce problème.

La modélisation du problème diffère du *TSP* pour *Ant Colony System* car, dans le problème de portefeuille, le nombre de projets sélectionnés n'est pas connu *a priori* alors que, dans le *TSP*, le nombre de villes à visiter est constant. Pour pouvoir adapter l'algorithme de fourmis au problème, un choix aléatoire du nombre de projets à sélectionner Ξ est fait à chaque cycle pour chaque fourmi. De même, l'importance relative χ accordée aux K critères est définie pour chacun d'eux. L'algorithme est relativement similaire à *Ant Colony System*, si ce n'est l'utilisation de K matrices de phéromone (une pour chaque critère), ainsi que la distribution de probabilité du choix du prochain sommet, qui change selon la formule (3.15) :

$$P(\Psi) = \begin{cases} \dfrac{\left[\sum_{k=1}^{K} \chi \cdot k \cdot (\sum_{j \in \Psi} \pi_{ij}^{k})\right]^{\alpha} \cdot [\eta_i(\Psi)]^{\beta}}{\sum_{h \in \Omega} \left(\left[\sum_{k=1}^{K} \chi \cdot k \cdot (\sum_{j \in \Psi} \pi_{ij}^{k})\right]^{\alpha} \cdot [\eta_i(\Psi)]^{\beta}\right)} & \text{si } i \in \Omega \\ 0 & \text{sinon} \end{cases} \tag{3.15}$$

avec Ψ la solution en cours de production, α et β des paramètres qui relativisent l'importance des pistes de phéromone par rapport à la visibilité, et Ω l'ensemble des sommets disponibles (n'ayant pas encore été sélectionnés).

3.3.2. *Incertitude et dynamique*

Les problèmes considérés jusqu'à présent dans ce chapitre sont caractérisés par des espaces de recherche où les informations nécessaires à la construction d'une solution sont bien identifiées et ne sont pas modifiées pendant cette construction.

On parle de problème dynamique lorsque l'une des données du problème change en cours de traitement. Ces données peuvent être les valeurs associées aux arêtes du graphe exploré par les fourmis, comme par exemple la modification de la distance entre deux villes dans le problème du voyageur de commerce ou l'importance d'un sommet.

Mais sur les graphes représentant l'espace de recherche, la dynamique peut aussi se faire sur la présence ou l'absence des sommets et arêtes eux-mêmes, c'est-à-dire

des changements de topologie du graphe. Dans un problème de transport, si une route est barrée pour travaux, l'arête qui la représente est impraticable et peut donc disparaître jusqu'à la fin des travaux. On peut bien sûr considérer ce genre de problèmes comme opérant sur des graphes complets contenant tous les sommets et arcs pouvant apparaître, sur lesquels on fait varier des valeurs qui sont placées à zéro, par exemple, pour exprimer le fait que le sommet ou l'arc ne sont pas exploitables.

Dans de nombreux cas, la dynamique sur un graphe peut être vue comme le découpage en une séquence de graphes statiques décrivant chaque changement dans le graphe. L'algorithme s'appliquant à un graphe statique peut alors être exécuté sur chacune des instances de la famille de graphes. Cependant, avec une telle approche, il ne faut pas perdre de vue le fait qu'optimiser les solutions sur chacun des graphes de la séquence ne revient pas à optimiser la solution complète sur la séquence, comme le rappellent les auteurs dans [MON 03]. Bien entendu, on ne connaît souvent pas l'évolution future du système. Une approche « gloutonne » d'optimisation des séquences de problèmes statiques est souvent retenue.

Si les données varient après l'obtention d'une première solution, on pourra éventuellement repartir du calcul précédent, pour retrouver une nouvelle solution. On parlera alors de « réoptimisation ». C'est souvent l'approche utilisée avec les séquences de problèmes statiques.

Si les données du problème changent durant le calcul même de la solution, ou si l'on ne désire pas appliquer des méthodes statiques sur des séquences de graphes eux-mêmes statiques, il faudra faire en sorte que l'algorithme soit suffisamment robuste pour s'y adapter, en réutilisant les traitements déjà effectués. On parlera d'algorithme adaptatif.

Bien entendu, la capacité de l'algorithme à repartir après un changement, ou à s'exécuter malgré les changements continuels, sera fonction du nombre et de l'importance de ces changements.

Les algorithmes de fourmis sont généralement bien indiqués pour ce genre de situation. En effet, la solution est en grande partie stockée dans l'environnement de calcul : le graphe. Il est alors possible de réutiliser ce qui n'a pas changé, en d'autres termes de conserver tout ou partie de la matrice de phéromone, et recalculer à partir de cette solution partielle une nouvelle solution, et ceci pour la réoptimisation ou pour un algorithme complètement adaptatif.

Ceci est rendu plus aisé dans certains cas, par le fait que les phéromones s'évaporent, et qu'une solution qui n'est plus entretenue par les fourmis finit par disparaître, mécanisme particulièrement adapté à la dynamique.

Plusieurs variantes de l'algorithme *Ant System* (AS) ont été présentées pour s'adapter aux problèmes dynamiques, le plus souvent testées sur le *Dynamic TSP* ou D-TSP

ou sur des problèmes de routage dans les réseaux de télécommunication ou de routage de véhicules qui sont des cas pratiques où l'on rencontre facilement de la dynamique.

Nous commençons dans les trois prochaines sections par décrire les classes de problèmes présentant des caractéristiques dynamiques, puis dans les sections suivantes les diverses solutions proposées.

3.3.2.1. *Le Dynamic TSP*

Le *Dynamic TSP* (*Dynamic Traveling Salesman Problem*), ou problème dynamique du voyageur de commerce, est une modification du problème traditionnel où les temps de parcours peuvent évoluer pendant l'exécution de l'algorithme de recherche d'une solution. Certaines versions du problème prennent également en compte le fait qu'une ville soit ajoutée ou retirée au problème, ou encore que certaines routes disparaissent ou apparaissent. En termes de graphes, il faut distinguer le cas où seules des modifications de temps de parcours sont prises en compte et le cas où des villes et des routes apparaissent et disparaissent. Dans le premier cas, la dynamique du problème réside dans des changements de valeurs des poids des arêtes du graphe. Dans le second cas, la dynamique est structurelle, le graphe qui modélise le problème voit son nombre d'arêtes et de sommets changer au cours de l'exécution. La différence entre ces deux cas est importante car, dans le premier, toutes les solutions générées sont comparables entre elles alors que dans le second cas, une solution générée dans une configuration structurelle du graphe ne peut être comparée avec une solution construite dans une autre configuration.

3.3.2.2. *Le Dynamic VRP*

On parle de D-VRP, ou *Dynamic Vehicle Routing Problem*, lorsque les ordres de livraison ou de collecte peuvent arriver alors que les véhicules sont déjà en route.

On considère deux applications : la collecte, où les véhicules partent à vide et doivent collecter des biens (et donc peuvent facilement modifier leur chemin) et les systèmes de distribution, où les véhicules ont un chargement et ne peuvent modifier leur chemin de distribution que de manière limitée.

Ici, la dynamique ne change pas le graphe représentant le problème (le circuit de routes qui peuvent être empruntées par les fourmis) mais la façon dont les solutions peuvent être remises en cause par de nouvelles contraintes de passage auprès des clients.

3.3.2.3. *Routage dynamique dans les réseaux de télécommunication*

Le routage s'intéresse à l'acheminement dans les délais les plus brefs d'informations à travers un réseau de communication .

L'algorithme le plus connu dans ce domaine est *AntNet* [DIC 98], mais nous avons aussi présenté précédemment la méthode ABC, dans le cas des réseaux commutés [SCH 97].

Ici, la dynamique se situe au niveau des pannes ou dysfonctionnements éventuels, ainsi que sur les changements de charge des liens de communication. Lorsqu'un lien ou un routeur du réseau tombe en panne, devient défaillant, le graphe est modifié. De plus, implicitement dans ces problèmes, la charge de chaque lien peut varier en fonction du trafic, et nous avons évoqué plus haut quelques techniques d'équilibrage de la charge.

L'approche ABC utilise la nature décentralisée des individus pour disperser des fourmis dans un véritable réseau de communication décentralisé. L'une des caractéristiques qui est rarement utilisée est la capacité qu'ont les colonies de fourmis à s'adapter aux changements de leur environnement. En effet, même après la construction d'un chemin, les fourmis sont capables, dans une certaine mesure, d'améliorer ou de modifier ce chemin si les conditions dans l'environnement l'imposent. Il est donc possible d'utiliser des algorithmes de fourmis pour des problèmes modélisés par des environnements qui évoluent, c'est-à-dire dont les informations changent pendant l'exécution de l'algorithme.

3.3.2.4. *Approches autour du D-TSP : changements dans la matrice de phéromone*

Dans [EYC 02], les auteurs s'intéressent à la première version du problème où seuls les temps de parcours sont modifiés. Ils traitent le problème des embouteillages qui se produisent sur les routes trop fréquentées, en augmentant les temps de trajets. Leur proposition reprend l'algorithme *Ant Colony System*, en ajoutant un mécanisme de lissage des pistes de phéromone, nommé *shaking*. Ce mécanisme est appliqué, de manière localisée, autour des routes encombrées et traite les routes avoisinantes. L'idée est de diminuer l'attraction des routes encombrées, ainsi que des routes qui y mènent, sans pour autant annuler l'effet des phéromones.

De la même façon, l'approche proposée par [GUN 01] traite de la dynamique sur l'ajout ou le retrait de nœuds villes dans le graphe et consiste aussi à modifier la matrice de phéromone lorsqu'un changement se produit dans le graphe du problème. Ce changement dans les phéromones peut être total, c'est la stratégie de secousse (*shake*), ou local à l'emplacement dans le graphe où la modification s'est produite. Pour cette seconde technique, deux approches sont proposées.

Les auteurs soulignent un problème majeur de cette approche, qui consiste à modifier suffisamment les valeurs de phéromone pour que l'algorithme reparte et trouve la ou les nouvelles solutions. Ceci est fait tout en conservant suffisamment des anciennes valeurs, pour rendre cette recherche plus rapide que le redémarrage complet de l'algorithme.

L'algorithme proposé suit l'approche générale des ACO. Cependant, dès qu'un changement est opéré sur le graphe, les phéromones associées aux arcs sont modifiées. Cette façon de procéder a déjà été proposée par [GAM 99a], en remettant toutes les phéromones à leur valeur par défaut, ou par [STU 00], en incrémentant toutes les phéromones proportionnellement à leur différence avec la plus haute valeur.

Ici, les différentes stratégies se basent sur la redistribution, pour chaque nœud i, d'une valeur de réinitialisation (*reset-value*) $\gamma_i \in [0, 1]$, qui est distribuée sur les arcs connectés à ce nœud, ainsi :

$$\tau_{ij} = (1 - \gamma_i)\tau_{ij} + \gamma_i \frac{1}{n-1}$$

Les nœuds nouvellement insérés, lieux de la modification, ont par défaut une valeur $\gamma_i = 1$.

La première stratégie propose de changer tous les arcs, en fixant γ_i à une constante dans l'intervalle $[0, 1]$, et ainsi de permettre à l'algorithme de repartir, tout en conservant en partie les anciennes solutions. C'est une procédure de secousse du graphe.

Les deux autres stratégies proposent d'effectuer le changement de manière localisée, autour de la partie modifiée du graphe.

La première, dite η-stratégie, spécifie γ_i pour chaque ville/nœud i proportionnellement à sa distance du point de changement dans le graphe, en utilisant comme distance d^{η}_{ij} les valeurs de η :

$$d^{\eta}_{ij} = 1 - \frac{\eta_{avg}}{\gamma_E \cdot \eta_{ij}}$$

avec $\eta_{avg} = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{k \neq i} \eta_{ik}$, le paramètre $\gamma_E \in [0, \inf[$ permettant de dimensionner le « cône » de distance.

La seconde, dite τ-stratégie, fait de même, mais utilise comme distance les heuristiques τ. La distance d^{τ}_{ij} est le maximum sur tous les chemins P_{ik} entre les nœuds i et k du produit des phéromones sur les arcs de P_{ik}. De plus, les valeurs de phéromone sont alignées sur $[0, 1]$, en utilisant la valeur maximum des phéromones $\tau_{\max}$:

$$d^{\tau}_{ij} = \max_{P_{ik}} \prod_{(x,y) \in P_{ik}} \frac{\tau_{xy}}{\tau_{\max}}$$

3.3.2.5. *Approches autour du D-TSP : changements dans la matrice de phéromone et fourmi élitiste*

Les auteurs précédents ont ensuite ajouté dans [GUN 01], au processus de modification des phéromones décrit à la section précédente, l'utilisation d'une « fourmi

élitiste », qui tente de conserver la meilleure solution, même en la présence de changements dans le graphe qui pourraient l'invalider.

Pour ce faire, au lieu d'invalider une fourmi élue, car le chemin qu'elle utilise n'est plus utilisable, en raison d'un changement dans le graphe, des techniques de mise à jour de son chemin sont utilisées :

– lorsque des nœuds participant au chemin de la fourmi ont été retirés, on reconnecte les successeurs et prédécesseurs dans le chemin ;

– lorsque des nœuds participant au chemin de la fourmi ont été ajoutés, on les insère dans ce chemin à la position qui génère la moindre augmentation en longueur du chemin.

On évite ainsi la perte d'une des meilleures solutions, en la conservant en partie avec une solution proche reconstruite. Ceci permet de converger à nouveau plus vite vers une solution optimale.

3.3.2.6. *Approches autour du D-TSP : approche par population de solutions*

Cette technique, des mêmes auteurs, propose de mélanger une approche orientée algorithmes génétiques avec l'approche de fourmis dans [GUN 02]. Pour cela, une population de bonnes solutions est maintenue.

A l'issue de chaque étape, au lieu de conserver les phéromones, on déduit à nouveau la matrice de phéromone, à partir de cette population de bonnes solutions.

En cela, cette approche reprend le comportement de la fourmi élitiste proposé ci-dessus. Il s'agit maintenant de modifier la population de bonnes solutions, dans le cas où un changement survient dans le problème.

L'approche change des ACO habituels à l'étape de l'évaporation. Dans l'algorithme ACO standard, à chaque étape, les fourmis parcourent le graphe itérativement, jusqu'à atteindre la destination. Ensuite, les phéromones sont déposées en fonction des solutions trouvées. Ici, la matrice de phéromone est mise à jour différemment, et l'étape d'évaporation n'a pas lieu. Ainsi :

– lorsqu'une nouvelle solution apparaît, on ajoute de la phéromone sur le chemin :

$$\forall i \in [1, n] : \tau_{i\pi(i)} \rightarrow \tau_{i\pi(i)} + \Delta$$

– lorsqu'elle disparaît, on enlève de la phéromone du chemin :

$$\forall i \in [1, n] : \tau_{i\sigma(i)} \rightarrow \tau_{i\sigma(i)} - \Delta$$

Comme décrit plus haut, lorsqu'un changement se produit dans le graphe, il est nécessaire de modifier la population de solutions. Dans l'algorithme proposé ici, on

procède au retrait des nœuds villes qui ont disparu, puis on insère les nouveaux nœuds de manière gloutonne aux positions qui accroissent le moins la taille des chemins.

Cet algorithme ne garde pas qu'une fourmi élue, mais une population de solutions. Cette population peut être gérée suivant différentes stratégies.

Une première étape vise à créer une population de solutions initiales pendant une période donnée. Les meilleures solutions pendant k étapes sont insérées dans la population (avec k la taille souhaitée de la population).

Ensuite plusieurs stratégies sont possibles :

– basée sur l'âge : la plus vieille solution quitte la population et la meilleure à l'itération courante entre ;

– basée sur la qualité : on ajoute la meilleure solution de l'itération courante. La pire solution de la population est enlevée. Mais cela peut produire des problèmes avec une convergence parfois trop rapide (optimum local) ;

– basée sur la qualité probabiliste : le retrait est probabiliste ;

– mélange des stratégies probabiliste et par âge.

3.3.2.7. *Approches autour du routage dans les réseaux*

AntNet [DIC 98], ainsi que *AntHocNet* [DIC 05] par G. Di Caro et ses collègues, sont apparentées aux *ACO*, elles sont néanmoins très proches de ABC de par leur fonctionnement distribué et leur adaptation à la dynamique. *AntNet* est un algorithme de routage pour les réseaux infrastructurés tels qu'Internet. *AntHocNet*, quant à lui, est un protocole de routage hybride pour les réseaux sans fil et mobiles *ad hoc*.

3.3.2.8. *Approches autour du D-VRP*

[MON 03] propose aussi un algorithme prenant en compte la dynamique pour le problème du routage de véhicules dynamique. Ici, comme on l'a vu précédemment, la dynamique correspond à la mise au point d'un routage pour une série de véhicules, puis à la prise en compte de nouveaux ordres de routage pendant le déplacement des véhicules. Le principe repose sur l'utilisation de l'algorithme ACS pour résoudre des VRP statiques et un mécanisme de transfert des bonnes solutions d'un VRP résolu à un autre, représentant la modification du problème. L'idée est de réutiliser les calculs d'un VRP pour initialiser les calculs d'un autre VRP sur un problème qui a changé faiblement, et dont on peut donc espérer qu'il possède des solutions proches.

Le système décrit est composé de trois parties principales :

– un système d'optimisation par ACS, chargé de traiter des VRP statiques. Nous avons déjà parlé de ce système dans la partie sur l'optimisation multicritère ;

– un gestionnaire d'événements, chargé de gérer l'arrivée d'ordres et leur distribution, pour ne pas surcharger les ACS statiques ;

– une procédure de passage des matrices de phéromone d'un VRP statique à un autre, chargée de dériver les résultats du premier si un changement survient.

Le gestionnaire d'événements accepte les ordres entre le début de la journée et une heure donnée. Tous les ordres arrivant plus tard sont transférés au jour suivant. Lorsque de tels ordres restent au début d'une journée, une étape spéciale d'optimisation est menée sur ces seuls ordres, puis les nouveaux ordres sont traités.

Le gestionnaire fonctionne par laps de temps, ce qui évite d'interrompre l'ACS en milieu de calcul. A la fin de chaque laps de temps, les meilleures solutions sont utilisées et les routes sont envoyées aux véhicules.

L'algorithme ACS utilisé pour optimiser chacun des VRP réutilise en partie l'algorithme multi-objectif décrit dans [GAM 99b].

Enfin, la procédure de réutilisation de la matrice de phéromone se réalise comme suit. L'idée est qu'entre deux instances statiques du problème, il existe probablement des similitudes importantes, et qu'il est préférable de redémarrer avec la solution au problème précédent, que de partir de zéro. L'idée est tirée, à nouveau, de [GUN 01]. Sur chaque arc, dans le nouveau problème, on conserve une partie de l'information phéromonale de l'arc correspondant dans l'ancien problème (s'il y en a un) :

$$\tau_{ij} = (1 - \gamma_r)\tau_{ij}^{old} + \gamma_r \tau_0$$

où τ_{ij}^{old} est la valeur de phéromone de l'ancien problème. Le paramètre γ_r est utilisé pour gérer cette redistribution des phéromones. Pour les nouveaux arcs, la valeur τ_0 initiale est utilisée.

3.3.2.9. *Une autre approche*

Dans [CAR 06], les auteurs présentent un algorithme, nommé AntCO^2, pour la distribution dynamique et adaptative d'applications réparties sur des grilles ou des grappes de machines hétérogènes. Cette approche est une application des techniques développées dans le chapitre 5 du volume 2 [DUT 09].

Les applications réparties sont considérées comme constituées d'entités éventuellement migrables, chacune communiquant avec d'autres. L'idée principale est qu'une distribution de ces entités sur les diverses machines à disposition devrait essayer de répartir le plus équitablement possible la charge des entités (et ceci, en prenant en compte l'hétérogénéité de ces machines), et en même temps essayer de limiter au maximum la charge réseau induite par les communications entre entités.

Pour ce faire, il faut placer les entités qui communiquent beaucoup entre elles ensemble sur une même machine, de façon à ne pas utiliser ou surcharger les réseaux

entre machines, mais en même temps éviter de surcharger les machines par l'exécution de trop nombreuses entités.

Ces deux critères de distribution sont contradictoires, et un compromis doit être déterminé.

L'approche est dynamique, dans le sens où des entités peuvent apparaître ou disparaître à tout moment. De plus, les communications entre entités elles aussi varient, en intensité et en nombre au fil du temps, si bien qu'un groupe d'entités en très forte interaction à un moment donné peut très bien ne plus interagir à un autre moment, ce qui peut éventuellement modifier la distribution.

Le problème est modélisé par un graphe dynamique, dans lequel les entités sont représentées par les nœuds, et les communications par des arcs valués par l'intensité de la communication. Ce graphe est dynamique, car l'ajout, l'évolution, ou le retrait d'une entité ou d'une communication se traduit par l'ajout, l'évolution ou le retrait du nœud ou arc correspondant.

Le problème évolue donc dans le temps, et le graphe que nous utilisons $G(t) = \{E(t), V(t)\}$ aussi, d'où le paramètre t représentant le temps.

Ici, il ne s'agit pas de construire des chemins entre deux points, mais de trouver des grappes d'entités au sein du graphe, qui sont en forte interaction. Ainsi, on pourra détecter les groupes d'entités en forte communication et les placer sur la même machine. Ceci est réalisé grâce à des colonies de fourmis qui collaborent au sein de la colonie pour coloniser les zones de forte interaction, et qui entrent en compétition au niveau des colonies, pour s'approprier une zone ou une autre. Ce problème peut donc s'apparenter à un problème de partionnement dynamique.

En cela, ce problème découle des problèmes de détection de communautés ou d'organisations qui seront étudiés au chapitre 5 du volume 2 [DUT 09], en y ajoutant les problèmes de répartition de charge propres à la distribution d'applications.

Ce problème étant en grande partie décrit dans le chapitre 5 du volume 2 [DUT 09], nous n'en donnons ici qu'une description rapide. Le principe repose sur le marquage, dans l'environnement représenté par le graphe, des zones en forte interaction, par les phéromones des fourmis. Le comportement des fourmis va donc viser à favoriser leur attraction dans les zones en forte interaction. L'intérêt de l'approche par fourmis ici réside dans le fait que les phéromones s'évaporent et doivent donc être entretenues par les fourmis.

Les zones du graphe en forte interaction seront colonisées et des phéromones déposées. Mais si les entités composant cette organisation arrêtent de communiquer, les fourmis n'entretiendront plus la zone, les phéromones s'évaporeront, et ainsi la zone

ne sera plus détectée comme une organisation. De cette manière, il est possible de suivre la dynamique de chaque organisation au sein du graphe représentant l'application à distribuer.

Chaque colonie de fourmis est identifiée par une couleur, et les phéromones déposées sont elles-aussi colorées. Ainsi, il est facile d'identifier les organisations comme des ensembles connexes de nœuds du graphe colorés uniformément par une colonie donnée.

3.4. Conclusion

Les algorithmes de fourmis ont montré leur capacité à traiter avec succès un ensemble très conséquent de problèmes d'optimisation traditionnels caractérisés par des informations fiables et constantes [SOL 08]. La résolution de ces problèmes consiste dans la plupart des cas à minimiser une fonction de coût.

Lorsque la dynamique, l'incertitude ou l'absence de vision globale sont au cœur de la définition des problèmes, certains travaux d'une inspiration proche de ABC ont été proposés ces dernières années.

La majorité de ces travaux reposent sur une approche décentralisée dépourvue de fonction de coût globale et les méthodes proposées sont capables de s'adapter aux changements qui se produisent en cours de résolution.

3.5. Bibliographie

[ANG 09] Angus D., Woodward C., « Multiple Objective Ant Colony Optimisation », *Swarm Intelligence*, vol. 3, n° 1, p. 69–85, 2009.

[BAR 03] Barán B., Schaerer M., « A Multiobjective Ant Colony System for Vehicle Routing Problem with Time Windows », *Proceedings of the 21st IASTED International Conference*, Innsbruck, Autriche, février 2003.

[BES 00] den Besten M., Stutzle T., Dorigo M., « Ant colony optimization for the total weighted tardiness problem », *Proceedings of PPSN-VI, Sixth International Conference on Parallel Problem Solving from Nature*, vol. 1917, p. 611–620, Springer-Verlag, Berlin, Heidelberg, 2000.

[BIL 96] Bilchev G., « Evolutionary Metaphors for the Bin Packing Problem », *Evolutionary Programming*, p. 333–341, 1996.

[BUL 99] Bullnheimer B., Hartl R., Strauss C., « A new rank-based version of the ant system : a computational study », *Central European Journal for Operations Research and Economics*, vol. 7, n° 1, p. 25–38, 1999.

[CAR 06] Cardon A., Dutot A., Guinand F., Olivier D., « Competing Ants for Organization Detection : application to Dynamic Distribution », M. A. Alaoui, C. Bertelle (dir.),

Emergent Properties in Natural and Artificial Dynamical Systems, Springer complexity : Understanding Complex Systems, p. 25–52, Springer-Verlag, Berlin, Heidelberg, 2006.

[COR 00] Cordon O., de Viana I., Herrera F., Moreno L., « A new ACO model integrating evolutionary computation concepts : the best-worst ant system », M. Dorigo, L. M. Gambardella, M. Middendorf, T. Stützle (dir.), *Abstract proceedings of ANTS 2000 – From Ant Colonies to Artificial Ants : Second International Workshops on Ant Algorithms*, p. 22–29, IEEE Transactions on Evolutionary Computation, 2000.

[COS 97] Costa D., Hertz A., « Ants Can Colour Graphs », *The Journal of the Operational Research Society*, vol. 48, n° 3, p. 295–305, 1997.

[DIC 98] Di Caro G., Dorigo M., « AntNet : Distributed Stigmergetic Control for Communications Networks », *Journal of Artificial Intelligence Research*, vol. 9, p. 317–365, 1998.

[DIC 05] Di Caro G., Ducatelle F., Gambardella L., « AntHocNet : an ant-based hybrid routing algorithm for mobile ad hoc networks », *Proceedings of Parallel Problem Solving from Nature (PPSN VIII)*, vol. 3242 de *Lecture Notes in Computer Science*, p. 461–470, Springer-Verlag, Berlin, Heidelberg, 2005.

[DOE 01] Doerner K., Gutjahr W., Hartl R., Strauss C., Stummer C., « Ant colony optimization in multiobjective portfolio selection », *Proceedings of the 4th Metaheuristics Intl. Conf., MIC'2001*, p. 243–248, Porto, Portugal, 16-20 juillet 2001.

[DOE 04] Doerner K., Gutjahr W. J., Hartl R. F., Strauss C., Stummer C., « Pareto Ant Colony Optimization : A Metaheuristic Approach to Multiobjective Portfolio Selection », *Annals of Operations Research*, vol. 131, n° 1-4, p. 79–99, octobre 2004.

[DOR 91] Dorigo M., Maniezzo V., Colorni A., Positive feedback as a search strategy, Rapport, Dipartimento di Elettronica e Informatica, Politecnico di Milano, 1991.

[DOR 92] Dorigo M., Optimization, Learning and Natural Algorithms (in Italian), PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italie, 1992.

[DOR 96] Dorigo M., Maniezzo V., Colorni A., « Ant system : optimization by a colony of cooperating agents », *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 26, n° 1, p. 29–41, 1996.

[DOR 97a] Dorigo M., Caro G. D., « The Ant Colony Optimization Meta-Heuristic », D. Corne, M. Dorigo, F. Glover (dir.), *New Ideas in Optimization*, p. 11–32, McGraw-Hill, Maidenhead, Royaume Uni, 1997.

[DOR 97b] Dorigo M., Gambardella L., « Ant colony system : a cooperative learning approach to the travelingsalesman problem », *IEEE Transactions on Evolutionary Computation*, vol. 1, n° 1, p. 53–66, 1997.

[DOR 97c] Dorigo M., Gambardella L. M., « Ant colonies for the travelling salesman problem », *Biosystems*, vol. 43, n° 2, p. 73–81, juillet 1997.

[DUT 09] Dutot A., Olivier D., « La detection d'organisations dans les systèmes complexes par colonies de fourmis », N. Monmarché, F. Guinand, P. Siarry (dir.), *Fourmis Artificielles, nouvelles directions pour une intelligence collective*, vol. 2 de *Traité IC2*, chap. 5, Hermès, Paris, 2009.

[EYC 02] Eyckelhof C., Snoek M., « Ant Systems for a Dynamic TSP », M. Dorigo, G. Di Caro, M. Sampels (dir.), *Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002)*, vol. 2463 de *Lecture Notes in Computer Science*, p. 88–99, Bruxelles, Belgique, Springer-Verlag, Berlin, Heidelberg, 12-14 septembre 2002.

[FEN 03] Fenet S., Solnon C., « Searching for Maximum Cliques with Ant Colony Optimization », *Applications of evolutionary computing (EvoCOP 2003)*, vol. 2611 de *Lecture Notes in Computer Science*, p. 236–245, Springer-Verlag, Berlin, Heidelberg, avril 2003.

[GAM 95] Gambardella L. M., Dorigo M., « Ant-Q : A Reinforcement Learning Approach to the Traveling Salesman Problem », *International Conference on Machine Learning*, p. 252–260, 1995.

[GAM 99a] Gambardella L., Taillard E., Dorigo M., « Ant colonies for the quadratic assignment problem », *Journal of the Operational Research Society*, vol. 50, n° 2, p. 167–176, JSTOR, 1999.

[GAM 99b] Gambardella L. M., Taillard E., Agazzi G., « MACS-VRPTW : A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows », D. Corne, M. Dorigo, F. Glover (dir.), *New Ideas in Optimization*, p. 63–76, McGraw-Hill, Maidenhead, Royaume Uni, 1999.

[GUN 01] Guntsch M., Middendorf M., « Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP », E. Boers, J. Gottlieb, P. Lanzi, R. Smith, S. Cagnoni, E. Hart, G. Raidl, H. Tijink (dir.), *Applications of Evolutionary Computing : EvoWorkshops 2001 : EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM*, vol. 2037 de *Lecture Notes in Computer Science*, p. 213–222, Como, Italy, Springer-Verlag, Berlin, Heidelberg, 18-20 avril 2001.

[GUN 02] Guntsch M., Middendorf M., « Applying Population Based ACO to Dynamic Optimization Problems », M. Dorigo, G. Di Caro, M. Sampels (dir.), *Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002)*, vol. 2463 de *Lecture Notes in Computer Science*, p. 111–122, Springer-Verlag, Berlin, Heidelberg, Bruxelles, Belgique, 12-14 septembre 2002.

[LEG 99] Leguizamon G., Michalewicz Z., « A new version of ant system for subset problems », *Proceedings of the Congress on Evolutionary Computation (CEC 99)*, vol. 2, 6-9 juillet 1999.

[LEG 01] Leguizamon G., Michalewicz Z., « An ant system for the maximum independent set problem », *Proceedings of the 2001 Argentinian Congress on Computer Science*, p. 1027–1040, 2001.

[MAN 99a] Maniezzo V., « Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem », *INFORMS Journal on Computing*, vol. 11, n° 4, p. 358–369, 1999.

[MAN 99b] Maniezzo V., Colorni A., « The ant system applied to the quadratic assignment problem », *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, n° 5, p. 769–778, 1999.

[MON 03] Montemanni R., Gambardella L. M., Rizzoli A. E., Donati A. V., « A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System », *In Second*

International Workshop on Freight Transportation and Logistics, p. 27–30, 2003.

[SCH 97] SCHOONDERWOERD R., HOLLAND O., BRUTEN J., ROTHKRANTZ L., « Ant-Based Load Balancing in Telecommunications Networks », *Adaptive Behavior*, vol. 5, n° 2, p. 169–207, ISAB, 1997.

[SHA 01] SHAWE-TAYLOR J., ZEROVNIK J., « Ants and graph coloring », *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA'01*, p. 276–279, 2001.

[SOL 08] SOLNON C., *Optimisation par colonies de fourmis*, Hermès, Paris, 2008.

[STU 00] STUTZLE T., Hoos H. H., « MAX-MIN Ant System », *Future Generation Computer Systems*, vol. 16, n° 8, p. 889–914, 2000.

[TAL 01] TALBI E. G., ROUX O., FONLUPT C., ROBILLARD D., « Parallel Ant Colonies for the quadratic assignment problem », *Future Generation Computer Systems*, vol. 17, n° 4, p. 441–449, 2001.

[TK 02] T'KINDT V., MONMARCHÉ N., TERCINET F., LAÜGT D., « An Ant Colony Optimization Algorithm to Solve a 2-machine Bicriteria Flowshop Scheduling Problem », *European Journal of Operational Research*, vol. 142, n° 2, p. 250–257, 2002.

[ZWA 99] VAN DER ZWAAN S., MARQUES C., « Ant colony optimisation for job shop scheduling », *Proceedings of the Third Workshop on Genetic Algorithms and Artificial Life (GAAL 99)*, 1999.

Chapitre 4

Les fourmis artificielles pour l'optimisation en variables continues

Les métaheuristiques, en particulier à base de fourmis artificielles, ont été le plus souvent conçues pour le traitement des problèmes combinatoires (voir les chapitres 2 et 3). Cependant, il existe un type de problème d'optimisation fréquemment rencontré en ingénierie, où une partie des variables de décision, voire la totalité d'entre elles, sont continues. Le recours aux métaheuristiques peut, dans ce cas, être indiqué si le problème est « difficile » : fonction objectif non dérivable, *minima* locaux multiples, grand nombre de dimensions, non-convexité, etc.

Les méthodes évolutionnaires se sont bien développées pour ce type d'espace de recherche (voir [LER 07]) et plusieurs démarches ont été proposées dans la littérature pour concevoir ou adapter les algorithmes de colonies de fourmis au domaine continu. Outre les difficultés habituelles inhérentes à l'adaptation d'une métaheuristique discrète, cette opération soulève des problèmes spécifiques. Ainsi, le principal obstacle surgit si l'on veut se situer au plus près du formalisme originel ACO des colonies de fourmis, avec une construction de la solution composant par composant. Un problème continu peut en effet, selon l'approche retenue, présenter une infinité de composants, ce qui rend délicate une telle construction. C'est pourquoi la plupart des algorithmes de colonies de fourmis continus s'inspirent des caractéristiques d'auto-organisation et de mémoire externe des colonies de fourmis, laissant de côté la construction itérative de la solution. Il y a, cependant aussi, des approches plus récentes, qui exploitent le caractère probabiliste du formalisme *ACO*.

Chapitre rédigé par Patrick Siarry, Johann Dréo et Nicolas Monmarché.

Nous présentons dans ce chapitre une description introductive des principaux algorithmes de colonies de fourmis pour l'optimisation continue qui ont été proposés dans la littérature. Pour structurer la présentation, nous avons regroupé ces algorithmes en quatre familles.

Nous décrivons d'abord les algorithmes qui reposent sur une modélisation directe du comportement des fourmis : l'algorithme CACO, de G. Bilchev *et al.*, construit en hybridant une approche de colonie de fourmis et un algorithme évolutionnaire ; l'algorithme API, de N. Monmarché *et al.*, inspiré du comportement d'une fourmi primitive et les algorithmes CIAC et HCIAC, de J. Dréo et P. Siarry.

La seconde famille concerne les algorithmes exploitant une discrétisation des nombres réels sous la forme de chaînes binaires : une première version de l'adaptation de ACO au cas continu en utilisant le même formalisme que les premiers algorithmes à estimation de distribution par N. Monmarché *et al.* ; puis l'algorithme *Adaptive Ant Colony* (AAC), de Y.J. Li et T.J. Wu et *Binary Ant System* (BAS), de M. Kong et P. Tian. Enfin, nous présentons API-HMM.

Puis nous présentons les algorithmes à base d'échantillonnage probabiliste : tout d'abord l'algorithme ACO-continu, de K. Socha et M. Dorigo ; l'algorithme voisin CACS, de S. Pourtakdoust et H. Nobahari ; puis la variante ACO_R toujours de K. Socha et M. Dorigo ; l'algorithme MACACO de F. Franca *et al.* ; *Aggregation Pheromone System* (APS), de S. Tsutsui ; *Direct ACO*, de M. Kong et P. Tian et l'algorithme CANDO, de W. Tfaili et P. Siarry.

Enfin, nous décrivons quelques « méthodes hybrides », qui associent un algorithme de colonie de fourmis et un autre algorithme d'optimisation : un algorithme hybride avec un algorithme évolutionnaire, développé par C. Ling *et al.* ; ACO-LM, de C. Blum et K. Socha, exploitant la méthode de descente de Levenberg-Marquardt ; *Improved ACO*, de L. Chen *et al.*, utilisant un algorithme génétique ; *Continuous Orthogonal Ant Colony*, de J. Zhang *et al.*, qui met en œuvre une approche de « conception orthogonale » ; PSACO, de P. Shelokar *et al.*, exploitant un algorithme d'optimisation par essaim particulaire ; et *Immunity-based ACO*, de Y. Feng et Z. Feng, qui est une variante d'algorithme immunitaire.

Nous aurons ainsi un panorama assez varié de la problématique des espaces continus explorés par les fourmis artificielles.

4.1. Modélisation directe du comportement des fourmis

4.1.1. *L'algorithme CACO*

Le premier algorithme de colonies de fourmis en variables continues, qui fut tout naturellement dénommé CACO (*Continuous Ant Colony Algorithm*) [BIL 95], met en

œuvre deux approches successives : un algorithme de type évolutionnaire détermine d'abord, *via* des sélections et des croisements, des régions d'intérêt ; puis ces régions sont explorées et évaluées par des fourmis artificielles.

Une fourmi sélectionne une région avec une probabilité proportionnelle à la concentration en phéromone de cette région, de la même manière que, dans l'algorithme *Ant System*, une fourmi sélectionne une piste allant d'une ville à une autre :

$$p_i(t) = \frac{\tau_i^{\alpha}(t) \cdot \eta_i^{\beta}(t)}{\sum_{j=1}^{N} \tau_j^{\alpha}(t) \cdot \eta_j^{\beta}(t)} \tag{4.1}$$

où N est le nombre de régions, $\tau_i(t)$ la quantité de phéromone associée à une région, $\eta_i^{\beta}(t)$ est utilisé pour incorporer une heuristique spécifique au problème, et classiquement, α et β sont les deux paramètres servant à régler l'importance relative de τ et η.

Chaque fourmi part alors du centre de la région et se déplace selon une direction choisie aléatoirement, tant qu'elle observe une amélioration de la fonction objectif. Le pas de déplacement entre deux évaluations successives est de :

$$\delta r(t, R) = R \cdot \left(1 - u^{(1-\frac{t}{T})^c}\right) \tag{4.2}$$

où R est le diamètre de la région explorée, $u \in [0, 1]$ un nombre aléatoire, t le numéro de l'itération, T le nombre total d'itérations de l'algorithme et c un paramètre permettant de réduire le pas après chaque itération (effet analogue à la température du recuit simulé).

Si la fourmi découvre une solution meilleure, la région est déplacée de façon à ce que son centre coïncide avec cette solution et la quantité de phéromone de la région est augmentée proportionnellement à l'amélioration trouvée. L'évaporation des phéromones se fait classiquement, selon un taux ρ.

Une des difficultés de la méthode peut provenir du fait qu'une région abandonnée, suite au tarissement en nourriture de la zone, peut être, ultérieurement, à nouveau explorée par une fourmi sans que la notion d'absence de nourriture soit prise en compte. Cette caractéristique, la mémorisation de la réussite et non des échecs, se retrouve dans d'autres méthodes à base de fourmis pour des problèmes à variables continues.

Des améliorations à cet algorithme ont été apportées par M. Wodrich et G. Bilchev [WOD 97]. Ainsi, indépendamment de l'action des « fourmis locales », que nous venons de décrire, l'algorithme met en œuvre des « fourmis globales », chargées d'explorer l'espace de recherche (figure 4.1) : l'objectif est cette fois de remplacer les régions peu intéressantes par de nouvelles régions, non visitées jusqu'ici. A cette fin, chaque région est affectée d'un âge, qui augmente si elle n'apporte aucune amélioration de

la fonction objectif ; le paramètre t, qui joue sur le pas $\delta r(t, R)$ de déplacement des fourmis, est maintenant l'âge de la région explorée.

Une refonte de l'algorithme a été proposée ensuite par d'autres auteurs [MAT 00], en vue de relier plus étroitement CACO au paradigme des colonies de fourmis, en abandonnant les emprunts aux algorithmes évolutionnaires. Les auteurs ont ainsi introduit le concept de « diffusion », pour décider de la création de nouvelles régions.

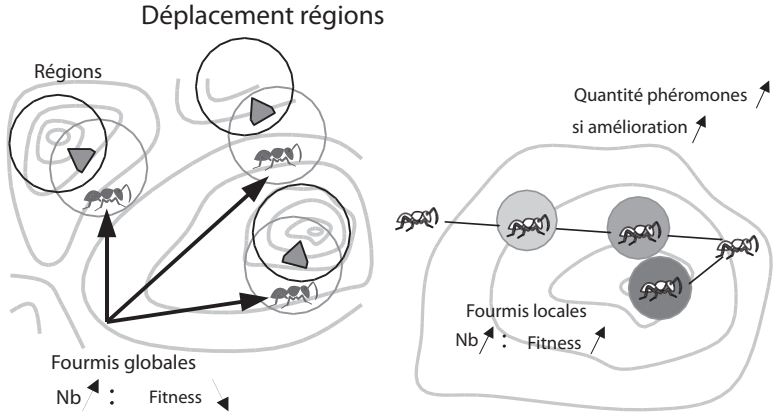


Figure 4.1. *L'algorithme CACO : les fourmis globales (a) participent au déplacement des régions que les fourmis locales (b) évaluent*

4.1.2. *L'algorithme API*

Dans de nombreux algorithmes de fourmis, le processus d'échange d'informations est un processus de communication indirecte par pistes de phéromone. Il existe cependant un algorithme adapté au cas continu qui ne fait pas usage de ce mode de communication : l'algorithme API [MON 00c], qui s'inspire du comportement de fourmis primitives (ce qui ne veut pas dire inadaptées), de l'espèce *Pachycondyla apicalis*.

L'algorithme API est une métaheuristique, dans le sens où il n'est pas attaché à un type d'espace de recherche particulier et il peut être adapté à de nombreux problèmes de natures très différentes. Historiquement, c'est sur des problèmes à variables continues qu'il a été évalué et c'est encore sur ce type d'espace qu'il a montré un intérêt (voir le chapitre 12 concernant l'apprentissage de modèles de Markov cachés). Pour illustrer sa généricité, il a été utilisé pour des problèmes combinatoires, comme le problème du voyageur de commerce [MON 00a] ou, plus récemment, il a été adapté au problème de classification non supervisée [ANT 08].

La caractéristique principale des fourmis qui a été retenue concerne leur aptitude à mémoriser les sites de chasse qui leur sont profitables et de systématiquement utiliser cette information pour tenter de capturer de nouvelles proies, afin de les ramener au nid. D'autres caractéristiques des fourmis de l'espèce *Pachycondyla apicalis* ont également été introduites dans la traduction algorithmique : par exemple, la capacité des fourmis à recruter des congénères une par une (recrutement en tandem) pour leur indiquer le nouveau site du nid, ainsi qu'une variabilité de l'amplitude de déplacement, en fonction de l'âge de la fourmi (les plus jeunes s'éloignent peu du nid).

L'espace de recherche du problème considéré est donc exploré par API par des explorations locales successives sur les solutions mémorisées par chaque fourmi : comme dans la nature, une fourmi se spécialise sur une zone de l'espace de recherche d'autant plus facilement qu'elle y trouve une proie à chaque sortie du nid. Dans le contexte de l'optimisation, la découverte d'une proie par une fourmi correspond à la découverte d'une meilleure solution au sens de la fonction objectif. La stratégie globale de l'algorithme est assez simple : tant que les fourmis améliorent leur solution, elles continuent à explorer les mêmes zones. En cas de stagnation de sa recherche, la fourmi perd patience et fabrique un nouveau site de chasse à explorer. La simplicité de la stratégie permet d'atteindre une certaine robustesse, lorsque l'on aborde un nouveau type d'espace de recherche. Cette robustesse a été particulièrement améliorée quand les variations dans les paramètres des fourmis ont été introduites : cela peut correspondre à des fourmis d'âges différents.

Comme régulièrement dans la nature, le nid, qui représente un point central dans la stratégie d'exploration de l'espace de recherche, est déplacé et provoque ainsi une réinitialisation de l'exploration par la colonie. Enfin, le recrutement en tandem peut consister en l'échange d'informations entre fourmis quand elles sont dans le nid. Ce qui correspond à une intensification des explorations autour de certains sites de chasse.

On peut résumer l'effort de conception et d'adaptation de API pour un problème particulier à la conception de deux opérateurs :

– un opérateur de génération de solution (par exemple, pour déterminer la position initiale du nid). Cet opérateur, noté O_{init}, est souvent un tirage aléatoire uniforme ;

– un opérateur d'exploration autour d'un site de chasse. L'exploration d'une solution procède souvent d'une recherche aléatoire dans un voisinage de cette solution. Cet opérateur est noté O_{explo}. Par exemple, dans le cas de valeurs réelles, on génère une nouvelle valeur selon une densité de probabilité gaussienne centrée en l'ancienne valeur, et l'écart-type est utilisé comme paramètre d'amplitude d'exploration.

Pour conclure, cet algorithme (voir l'algorithme 4.1) présente comme principal avantage sa simplicité de mise en œuvre : il est asynchrone, adaptable à différents types d'espace de recherche, facile à hybrider avec une heuristique dédiée et finalement assez robuste. Ses inconvénients sont assez classiques : le nombre de paramètres à

régler peut être important si l'on ne prend pas garde à rendre adaptatifs ceux qui peuvent l'être et certains espaces de recherche, même s'ils peuvent se prêter au jeu d'être parcourus par API, ne sont pas propices à l'obtention de résultats compétitifs avec des méthodes dédiées.

Algorithme 4.1 Algorithme API

Initialisation : positionner le nid avec l'opérateur O_{init}
tantque la condition de terminaison n'est pas vérifiée **faire**
  pour chaque fourmi f_i, en parallèle **faire**
    si f_i n'a pas assez de sites en mémoire **alors**
      créer un site pour f_i avec l'opérateur O_{explo} et l'amplitude A^i_{site}
    sinon
      si la dernière sortie du nid a été fructueuse **alors**
        f_i explore le dernier site visité avec l'opérateur O_{explo} et l'amplitude A^i_{local}
      sinon
        f_i choisit un site dans sa mémoire et l'explore avec l'opérateur O_{explo} et l'amplitude A^i_{local}
      fin si
      si l'exploration a été fructueuse **alors**
        f_i mémorise la nouvelle position du site
      sinon
        si le site a été exploré sans succès de trop nombreuses fois **alors**
          f_i oublie le site
        fin si
      fin si
    fin si
  fin pour
  Recrutement : éventuellement un recrutement en tandem entre fourmis est organisé (diffusion de sites prometteurs)
  Redémarrage : régulièrement, le nid est déplacé sur la meilleure solution trouvée jusqu'alors. Les fourmis oublient leurs sites et recommencent à partir du nouvel emplacement.
fin tantque
retourner s^+, la meilleure solution trouvée

Il existe des extensions, par exemple introduisant la méthode tabou [HO 06], en tenant compte des caractéristiques de la fonction objectif [SIU 08]. Egalement, API a été utilisé conjointement avec l'algorithme AS [LUH 08] pour l'optimisation de structures portantes (comme des charpentes). Dans ce dernier cas, API a été utilisé pour l'optimisation des valeurs réelles, alors que AS a été utilisé préalablement pour déterminer la topologie de la structure.

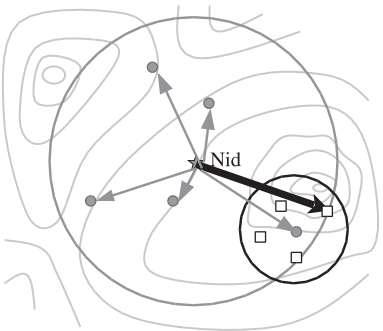


Figure 4.2. *L'algorithme API : une méthode à démarrage multiple, inspirée par une espèce de fourmi primitive. Les fourmis (cercles pleins) explorent des sites de chasse (petits carrés) dans un périmètre (grand cercle) autour du nid. Le nid est positionné sur le meilleur point, au moment de la réinitialisation (flèche en trait gras).*

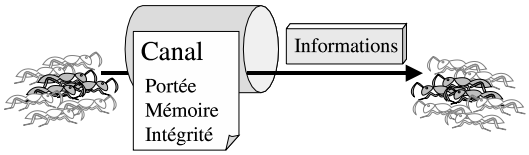


Figure 4.3. *Un canal de communication structure les caractéristiques de la transmission de l'information : portée, mémoire et intégrité*

4.1.3. *Les algorithmes CIAC et HCIAC*

Un autre algorithme, se focalisant sur les principes de communication au sein des colonies de fourmis, propose de ne pas limiter les échanges d'informations aux processus stigmergiques (communication indirecte, *via* la phéromone déposée dans l'environnement), mais de prendre en considération, en outre, les échanges directs d'informations entre les individus deux à deux (approche hétérarchique, observée chez les fourmis réelles) [DRE 04].

Une formalisation des échanges d'informations est proposée, *via* la notion de « canaux de communication ». On peut décrire ces canaux en s'intéressant aux caractéristiques du transport de l'information. Pour ce qui concerne les métaheuristiques, il y a trois caractéristiques principales (voir figure 4.3) :

– la « portée » : c'est le nombre d'individus mis en cause dans l'échange d'informations. L'information peut ainsi être émise par un individu et perçue par plusieurs autres, et inversement ;

– la « mémoire » : elle caractérise la persistance de l'information dans le système. L'information peut séjourner un certain temps, ou n'être que passagère ;

– l'« intégrité » : elle est relative aux modifications engendrées par l'utilisation du canal de communication. L'information peut évoluer dans le temps, ou être biaisée lors de sa transmission.

De plus, l'information transitant par un canal de communication peut être n'importe quelle information d'intérêt, comme par exemple la valeur ou la position d'un point de l'espace de recherche.

L'algorithme CIAC (*Continuous Interacting Ant Colony*) utilise deux canaux de communication :

– le canal stigmergique fait appel à des spots de phéromone déposés dans l'espace de recherche, qui sont plus ou moins attractifs pour les fourmis artificielles, selon leur concentration et leur éloignement. L'information portée par un spot concerne implicitement la position d'un point et explicitement la valeur de l'amélioration observée par la fourmi ayant déposé le spot ;

– le canal direct est implémenté sous la forme d'un échange de messages entre deux individus. Une fourmi artificielle possède une pile de messages reçus et peut envoyer des messages à une autre fourmi.

L'expérimentation de CIAC a mis en évidence un comportement intéressant, lorsque les deux canaux de communication sont utilisés en synergie : en particulier, une certaine capacité de l'algorithme à osciller entre des phases d'intensification et des phases de diversification.

Il s'est avéré cependant nécessaire, pour rendre l'algorithme compétitif, de l'hybrider avec la technique de recherche locale de J. Nelder et R. Mead. Le nouvel algorithme, appelé HCIAC, utilise en outre des processus décisionnels stochastiques : il exploite des fonctions de type stimulus/réponse, qui permettent de définir un seuil de choix pour une action. Concrètement, il s'agit d'une fonction sigmoïde :

$$p(s) = \frac{1}{1 + e^{(-\rho \cdot s + \rho\tau)}} \tag{4.3}$$

qui donne la probabilité $p(s)$ d'une décision, selon un stimulus s, fonction de l'état d'une fourmi, d'un seuil τ et d'une puissance ρ (quantifiant la « douceur » du choix). L'utilisation de ce type de fonction permet de s'affranchir d'un paramétrage délicat, par exemple en distribuant les seuils selon une loi normale sur toute la population. On peut aussi, par ce biais, mettre en place une méthode très simple d'apprentissage, en faisant varier les seuils.

L'algorithme HCIAC est décrit par la figure 4.4.

L'hybridation avec une méthode de descente locale a permis — comme souvent avec les algorithmes de colonies de fourmis — d'atteindre des résultats comparables à ceux procurés par les métaheuristiques concurrentes en optimisation continue.

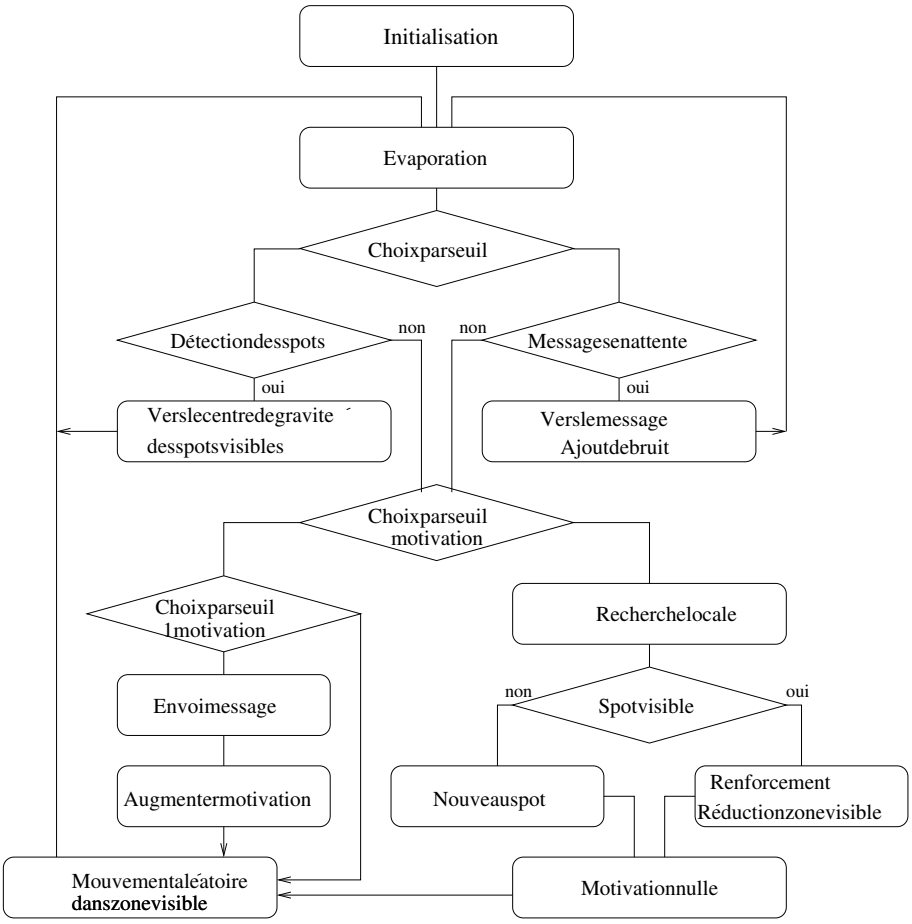


Figure 4.4. *L'algorithme HCIAC*

4.2. Algorithmes à base de discrétisation binaire

Dans cette catégorie, les algorithmes à base de fourmis artificielles se différentient des autres par la représentation, et la manipulation, des solutions en mémoire. L'espace de recherche est discrétisé en codant les nombres réels sous forme de chaînes de bits (par exemple en discrétisant l'intervalle réel pour que la chaîne binaire soit traduite en un entier qui indiquera la position de la valeur dans l'intervalle). Bien que cette façon de représenter l'espace de recherche ait perdu de son attrait dans le domaine des algorithmes évolutionnaires, en particulier car les connaissances liées au problème d'optimisation sont difficilement incorporables à l'algorithme d'optimisation qui utilise ce type de représentation, la généralité de ce codage permet de proposer des méthodes

assez simples, qui sont ainsi plus faciles à appréhender et à comparer d'un point de vue structurel (c'est-à-dire ne se limitant pas à des comparaisons expérimentales uniquement). Cette façon d'envisager le codage des solutions permet de comparer les mécanismes issus des fourmis à d'autres paradigmes bio-inspirés [DOE 08].

4.2.1. *ACO-canonique*

Un premier travail utilisant une représentation binaire de variables réelles a été présenté dans [MON 00b]. Le terme « canonique » a ici été utilisé en référence à l'algorithme génétique canonique (AGC) proposé par D. Goldberg [GOL 89] et qui manipulait des chaînes binaires, mais la comparaison s'arrête là. L'objectif de cette étude était de montrer la similitude de la structure algorithmique des algorithmes de type ACO (en l'occurrence AS et ACS) et les premières méthodes évolutionnaires à base de population infinie (qui sont ensuite devenues des méthodes d'optimisation par estimation de distributions). Ces méthodes partagent toutes une structure de données permettant d'estimer la probabilité d'apparition d'une valeur binaire pour chaque position de la chaîne de bits. Cette chaîne est ensuite utilisée pour donner une valeur réelle dans l'intervalle d'étude d'une fonction que l'on cherche à minimiser. C'est ainsi qu'il a été montré que les algorithmes BSC et PBIL pouvaient être présentés dans le même schéma algorithmique qu'une version binaire de ACO (voir l'algorithme 4.2).

Les notations communes aux trois algorithmes utilisés sont les suivantes :

– la fonction à optimiser est notée f et l représente le nombre de bits utilisés pour coder une solution de f ;

– $V = (p_1, \ldots, p_l)$, avec $p_i \in [0, 1]$, qui représente le vecteur de probabilité qui est utilisé pour générer des points de l'espace de recherche $\mathcal{S} = \{0, 1\}^l$ (p_i représente donc 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 sont générées à chaque cycle. P peut être considérée comme la population dans les algorithmes évolutionnaires.

Algorithme 4.2 Algorithme commun aux algorithmes BSC, PBIL et ACO

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: Evaluer $f(s_1), ..., f(s_n)$
5: Mettre à jour V selon $(s_1, ..., s_n)$ et $f(s_1), ..., f(s_n)$
6: **fin tantque**
7: **retourner** s^+, la meilleure solution trouvée

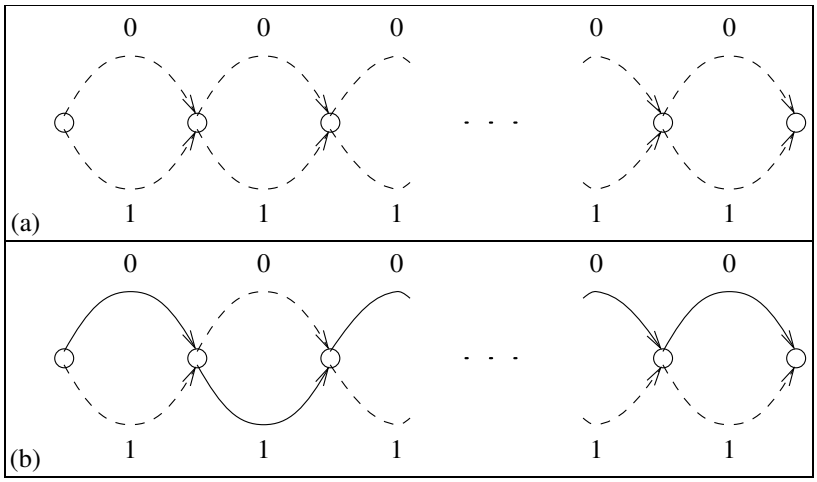


Figure 4.5. *Adaptation de ACO au problème d'optimisation binaire. Le choix des l bits est modélisé par le choix d'un arc « 0 » ou « 1 » entre les l + 1 sommets*

Pour utiliser le formalisme ACO, le problème d'optimisation a été reformulé de la façon suivante : un graphe est construit, où chaque sommet correspond à la position d'un bit et où les arcs correspondent au choix de la valeur du bit. La figure 4.5a 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 phéromone dans ACO, mais qui peut être ramenée à une seule valeur correspondant à la probabilité de suivre l'arc « 1 ». Notons 0_i et 1_i les deux arcs correspondant à la position i de la chaîne de bits. Les quantités de phéromone 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 » :

$$p_i = \frac{\tau_{1_i}}{\tau_{1_i} + \tau_{0_i}} \tag{4.4}$$

Les traces de phéromone sont donc équivalentes au vecteur V introduit précédemment. La figure 4.5b 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.

Par la suite, nous décrivons les détails des versions binaires de AS et ACS.

4.2.2. *L'algorithme AS$_b$ (*Ant System binaire*)*

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 dans AS$_b$:

$$\tau_{k_i} \leftarrow (1-\rho)\tau_{k_i} + \sum_{j=1}^{n} \Delta_{k_i}^{j} \tag{4.5}$$

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 :

$$\Delta_{k_i}^{j} = \begin{cases} \frac{1}{1+f(s_j)} & \text{si } s_j(i) = k \\ 0 & \text{sinon} \end{cases} \tag{4.6}$$

4.2.3. *L'algorithme ACS$_b$ (*Ant Colony System binaire*)*

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

$$\tau_{k_i} \leftarrow (1-\rho)\tau_{k_i} + \rho\Delta_{k_i} \tag{4.7}$$

où :

$$\Delta_{k_i} = \begin{cases} \frac{1}{1+f(s^{++})} & \text{si } s^{++}(i) = k \\ 0 & \text{sinon} \end{cases} \tag{4.8}$$

ACS introduit un moyen de contrôler le compromis entre l'exploration et l'exploitation[1]. Pour chaque bit généré pour la solution i à l'étape de génération de l'algorithme 4.2 :

– 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 :

$$s_i(j) = \begin{cases} 1 & \text{si } p_i > 0,5 \\ 0 & \text{sinon} \end{cases} \tag{4.9}$$

De plus, ACS utilise une règle de mise à jour locale mise en œuvre à l'étape de génération de la population dans l'algorithme 4.2 :

$$\tau_{k_i} \leftarrow (1-\alpha)\tau_{k_i} + \alpha\tau^0 \quad \text{si l'arc } k_i \text{ a été choisi} \tag{4.10}$$

1. On peut aussi parler d'intensification et de diversification, par analogie avec la recherche tabou.

où $\alpha \in [0, 1]$ est un paramètre. Cette mise à jour locale a pour but de modifier très légèrement la quantité de phéromone 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 phéromone sur l'arc k_i est supérieure à τ^0, après le passage d'une fourmi, la quantité de phéromone aura diminué. Si la quantité τ_{k_i} est inférieure à τ^0, la formule (4.10) 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 se sera rapprochée de $0,5$.

La principale différence entre les algorithmes de fourmis et les deux algorithmes évolutionnaires concernait la mise à jour de la distribution de probabilités après la génération d'une population (pour les fourmis, cela s'appelle la mise à jour des phéromones). Sans chercher à présager des performances de chaque méthode, il a été observé que la dynamique d'évolution des phéromones était sensiblement différente de la dynamique d'évolution de la distribution de probabilités mise en œuvre par les deux autres méthodes. Il a été constaté que les phéromones des fourmis convergent assez rapidement, et ceci est d'autant plus vrai pour AS_b que pour ACS_b (ce qui s'explique par le caractère élitiste de la mise à jour dans ACS_b).

4.2.4. *L'algorithme* Adaptive Ant Colony *(AAC)*

On peut décrire d'abord le principe d'une version non adaptative de l'algorithme AAC (*Adaptive Ant Colony*) [LI 03]. Une solution candidate est un vecteur de nombres réels. Chaque composante x de ce vecteur est codée par une chaîne de N bits : $\{b_N, b_{N-1}, \ldots, b_1\}$, avec $b_i \in \{0, 1\}$. On considère, pour chaque nombre x, le graphe orienté reliant les valeurs possibles 0 et 1 de chacun des bits de la chaîne (voir figure 4.6).

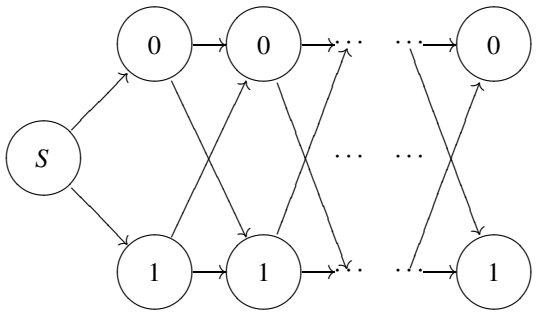


Figure 4.6. *Chemins entre les bits codant le nombre réel x.*

Les fourmis se déplacent le long de la chaîne, depuis la position de départ S jusqu'au bit de poids fort, puis en arrière jusqu'au bit de poids faible, en choisissant entre 0 et 1, pour chaque bit. Une fois le chemin effectué, la chaîne binaire est convertie en un entier B puis en nombre réel x, en supposant fixés $x_{\min}$ et $x_{\max}$:

$$x = \frac{B}{2^N - 1} \cdot (x_{\max} - x_{\min}) + x_{\min} \tag{4.11}$$

La solution candidate est évaluée, et une certaine quantité de phéromone est affectée aux arêtes visitées, ce qui permet de guider les parcours ultérieurs des fourmis, selon le principe classique du *Ant System*.

Telle quelle, cette procédure conduit à une exploration chaotique de l'espace de recherche, du fait des changements fréquents affectant les bits les plus significatifs. Les auteurs se sont efforcés de remédier à cette difficulté, en proposant une version adaptative, où la probabilité de changement des bits de poids fort décroît, lorsque la qualité de la solution augmente. Il en résulte une diminution du pas de déplacement, dans les régions prometteuses de l'espace de recherche.

4.2.5. *L'algorithme* Binary Ant System *(BAS)*

Cet algorithme *Binary Ant System* (BAS), décrit dans [KON 05], est une variante du précédent. La principale contribution de ce travail tient dans une formalisation du choix des incréments de phéromone.

4.2.6. *L'algorithme API-HMM*

Dans [SOU 01], l'algorithme API (voir le paragraphe 4.1.2 de ce chapitre) a été utilisé pour apprendre des modèles probabilistes plus perfectionnés qu'un simple vecteur, à savoir des modèles de Markov cachés (MMC), afin de résoudre des problèmes d'optimisation à variables continues. Les MMC sont utilisés pour générer des chaînes binaires ensuite traduites en valeurs réelles. La qualité d'un MMC est alors déterminée par un échantillonnage de la valeur de la fonction objectif, qui a pu être atteinte avec les séquences binaires générées par le MMC. Quand un MMC a permis d'améliorer les résultats, celui-ci est alors considéré comme un site de chasse fructueux et la fourmi tentera une amélioration de ce modèle à sa prochaine sortie du nid.

L'évaluation de l'intérêt de cet algorithme API-HMM n'est pas très facile, car les fourmis sont utilisées pour apprendre à optimiser une fonction et cela implique plusieurs hypothèses : la représentation binaire peut poser des problèmes, notamment lorsque la modification d'un bit ne provoque pas un mouvement de même amplitude suivant sa place dans la chaîne (d'où l'intérêt d'utiliser un codage binaire réfléchi – ou

code de Gray). De plus, on ne sait pas si la dépendance entre les bits successifs, modélisée par les MMC, correspond à une dépendance interne à une variable du problème. Enfin, comme les différentes variables de la fonction objectif ne sont pas obligatoirement liées par leurs expressions binaires, ce sont des MMC multidimensionnels qui ont été utilisés : la composante cachée du modèle est commune à toutes les variables, mais l'expression (c'est-à-dire la traduction en observation) est particulière à chaque dimension de la fonction objectif.

4.3. Algorithmes à base d'échantillonnage probabiliste

Comme nous l'avons vu avec l'algorithme ACO-canonique, l'algorithme d'optimisation peut tenter d'utiliser les explorations antérieures pour alimenter une structure (un vecteur, un arbre ou un graphe) qui le guidera vers l'*optimum*. Cette structure est alimentée, le plus souvent, par un échantillonnage opéré par la multitude des fourmis. Les algorithmes précédemment présentés construisaient un modèle de l'espace de recherche en se limitant à des représentations binaires des variables réelles. Dans la présente section, nous avons rassemblé les algorithmes qui se sont affranchis du codage binaire.

4.3.1. *L'algorithme ACO-continu*

Cet algorithme [SOC 04] tente de maintenir, dans le cas des variables continues, la construction itérative des solutions, en adoptant un point de vue différent des précédents. En effet, les auteurs considèrent que les composants de toutes les solutions sont constitués par les différentes variables optimisées. De plus, l'algorithme manipule la colonie globalement, les différentes fourmis artificielles n'étant plus que des points à évaluer.

Dans cette méthode, dénommée simplement « ACO pour l'optimisation continue » (ACO-continu), on tire aléatoirement, à chaque itération, une population de fourmis, selon une distribution de probabilité donnée. De cet ensemble de points ne sont conservés que les meilleurs, qui servent alors à construire une distribution de probabilité « améliorée » : la technique exploite ainsi le mécanisme des « algorithmes à estimation de distribution ».

Le type de distribution de probabilité utilisée est un « amalgame pondéré de noyaux normaux », c'est-à-dire un ensemble de distributions normales combinées :

$$P(x) = \sum_{j=1}^{k} \omega_j \cdot \left(\frac{1}{\sigma_j \sqrt{2\pi}} e^{\frac{(x-\mu_j)^2}{2\sigma_j^2}} \right) \text{ avec } x \in \mathbb{R} \tag{4.12}$$

en désignant par k le nombre de noyaux utilisés, μ_j et σ_j^2 la moyenne et la variance d'un noyau et ω_j la pondération.

Chaque distribution n'est utilisée que pour une variable, indépendamment des autres. La modification des distributions, dénommée « mise à jour de phéromone », consiste à renforcer ou à diminuer l'influence des noyaux correspondant aux solutions. L'évaporation des phéromones se traduit notamment par la diminution des poids ω_j.

Le principe de la méthode est présenté dans l'algorithme 4.3.

Algorithme 4.3 Algorithme ACO-continu [SOC 04]

Construire la distribution de probabilité initiale : $\tau_i^0 = P_i^0(x_i), i \in \{1 \ldots n\}$
tantque critère d'arrêt non atteint **faire**
  pour chaque fourmi, de $a = 1$ à m **faire**
    pour chaque variable, de $i = 1$ à n **faire**
      Choisir aléatoirement une valeur x_i selon la distribution $P_i(x_i)$
      Ajouter à la solution en construction : $S^a = \left\{s_1^a, \ldots, s_{i-1}^a\right\} \cup \{x_i\}$
    fin pour
  fin pour
  Mémoriser les k meilleures solutions trouvées : $S^* = \left\{s_1^*, \ldots, s_k^*\right\}$
  Reconstruire la distribution de probabilité selon les meilleures solutions : $\tau = P(S^*)$
fin tantque

4.3.2. *L'algorithme CACS*

Cette méthode, appelée *Continuous Ant Colony System* [POU 04] est très proche de la précédente, bien qu'ayant été présentée au même moment.

En effet, dans CACS comme dans ACO-continu, le cœur de l'algorithme consiste à faire évoluer une distribution de probabilité. Le principe de la méthode est celui déjà décrit dans l'algorithme 4.3. Dans CACS, la distribution utilisée est dite « normale », mais son expression diffère légèrement de la formule classique (équation (4.13)) et la variance utilisée est en fait un nouvel indice de dispersion (voir équation (4.14)).

$$P(x) = e^{-\frac{(x-x_{\min})^2}{2\sigma^2}} \tag{4.13}$$

en désignant par $x_{\min}$ le mode de la distribution et σ^2 l'indice de dispersion suivant :

$$\sigma^2 = \frac{\sum_{j=1}^{m} \frac{1}{f_j - f_{\min}} (x_j - x_{\min})^2}{\sum_{j=1}^{m} \frac{1}{f_j - f_{\min}}} \tag{4.14}$$

en désignant par m le nombre de fourmis, f_j la valeur de la fonction associée à la fourmi j et $f_{\min}$ la meilleure valeur trouvée.

Dans CACS, la seule distribution utilisée est centrée sur le mode de la distribution de l'itération précédente, et non sur la moyenne. L'avantage de cet algorithme est qu'il ne demande comme paramètre que le nombre de fourmis à utiliser. De plus, la génération d'une fourmi reste un mécanisme simple. En revanche, le principal défaut est que cette méthode se concentre rapidement sur une seule zone d'intérêt de l'espace de recherche et la fonction de densité gaussienne se concentre sur un *optimum* local, ce qui se traduit par le phénomène de convergence prématurée.

4.3.3. *L'algorithme ACO$_R$*

L'algorithme ACO$_R$ [SOC 08] a été conçu pour contourner le problème de convergence prématurée évoqué pour CACS. L'algorithme maintient une liste-mémoire des k meilleures solutions trouvées depuis le début de l'exécution. Chacune de ces solutions représente le centre d'une fonction de densité de probabilité gaussienne.

Le fonctionnement de ACO$_R$ peut être résumé de la façon suivante : initialement, la liste-mémoire de k solutions est générée de façon uniformément aléatoire sur l'espace de recherche et les solutions sont rangées par ordre décroissant de qualité. Une probabilité de sélection (c'est-à-dire d'attribution à une fourmi) est calculée pour chaque solution de la liste :

$$w_l = \frac{1}{qK\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2K^2}} \tag{4.15}$$

où K représente le nombre de fourmis, l est le rang de la solution dans la liste-mémoire et q est un paramètre permettant de régler le ratio exploration/exploitation. Le centre de chaque gaussienne est alors défini par la solution correspondante de la liste-mémoire et la variance est calculée par :

$$\sigma_l^i = \xi \sum_{e=1}^{K} \frac{\|s_e^i - s_l^i\|}{K-1}, i = 1, \ldots, n \tag{4.16}$$

où ξ est un paramètre réglant la vitesse de convergence, s est une solution de la liste-mémoire et n est la dimension de l'espace de recherche.

L'ensemble des distributions défini ci-dessus est utilisé pour générer de nouvelles solutions qui viennent compléter la liste-mémoire, en éliminant de celle-ci les solutions moins performantes.

Le coût calculatoire de ACO_R est plus important que pour CACS, mais cela lui confère une inertie qui le protège d'une convergence prématurée. Il a cependant été montré que la liste-mémoire peut s'uniformiser et ainsi bloquer la recherche dans un optimum local [FRA 08].

4.3.4. *L'algorithme MACACO*

L'algorithme MACACO (qui vient de *Multivariate Ant Colony Algorithm for Continuous Optimization*) [FRA 08] représente une évolution intéressante des algorithmes ACO_R et CACS, puisque l'apprentissage de la matrice de covariance entre les variables du problème a été intégrée dans l'apprentissage de la distribution de probabilité.

Afin de rendre l'exploration de l'espace de recherche plus robuste, les points initiaux sont générés selon une loi uniforme. Ensuite, à chaque itération, la matrice de covariance est calculée en retenant 70 % des meilleurs points, puis est utilisée pour mettre à jour la distribution qui servira à générer les prochains points. Cette étape de mise à jour représente un coût calculatoire important. Cette étape est présentée comme l'étape de mise à jour des phéromones, mais cela semble de plus en plus éloigné du modèle biologique...

Les auteurs ont remarqué que, lorsque la distribution initiale est centrée non loin de l'*optimum* global, l'apprentissage de la distribution est efficace... Ils ont donc utilisé un mécanisme de redémarrage de l'exploration, quand l'optimum connu n'est pas amélioré pendant dix itérations successives. Une distribution uniforme est alors utilisée pour reformer la matrice de covariance (comme pour l'initialisation).

De plus, probablement pour limiter la convergence prématurée, l'algorithme fonctionne selon deux phases : la première, où le meilleur point connu sert de centre pour la distribution multivariée puis une phase 2, activée lorsque la méthode semble stagner, dans laquelle le centre utilisé correspond à la moyenne des populations récentes.

En conclusion, l'algorithme MACACO montre bien une convergence des algorithmes ACO continus vers des méthodes du type CMA-ES [HAN 04], où la matrice des covariances est apprise dans le contexte d'une stratégie d'évolution. Cependant, le lien avec les fourmis se perd dans la notion de population de solutions servant à l'échantillonnage.

4.3.5. *L'algorithme APS*

L'algorithme *Aggregation Pheromone System* (APS) [TSU 04] est analogue à ACO-continu. Il introduit la fonction de densité de probabilité de « phéromone d'agrégation », définie comme suit :

$$p_t(t,x) = \frac{\tau(t,x)}{\int_X \tau(t,x)\mathrm{d}x} \tag{4.17}$$

où x désigne une variable dans un espace de recherche X et $\tau(t,x)$ est la densité de phéromone d'agrégation au temps t.

Un individu de rang r dépose autour de $x_{t,r}$ une quantité $\Delta\tau'(t,r,x_{t,r},x)$ de phéromone.

En une itération, m fourmis déposent la quantité totale de phéromone :

$$\Delta\tau(t,x) = \sum_{r=1}^{m} \Delta\tau'(t,r,x_{t,r},x) \tag{4.18}$$

Les auteurs supposent que, pour toute itération :

$$\int_X \tau(t,x)\mathrm{d}x = C \tag{4.19}$$

Le dépôt de phéromone est de la forme suivante :

$$\Delta\tau'(t,r,x_{t,r},x) = \frac{C}{\sum_{k=1}^{m} k^{\alpha}} r^{\alpha} N(x_{t,r},\beta^2\Sigma_t) \tag{4.20}$$

α est un paramètre qui permet d'ajuster l'importance relative du rang, Σ est la matrice de covariance estimée à partir de la distribution de m individus dans l'espace de recherche X à l'itération t, β est un paramètre qui permet de contrôler l'étendue de la distribution de phéromone, $N(x_{t,r},\beta^2\Sigma_t)$ est une distribution normale multivariée.

L'échantillonnage est analogue à celui mis en œuvre dans ACO-continu, mais l'approche, qui se veut plus générale, est aussi plus complexe.

4.3.6. *L'algorithme DACO*

L'algorithme *Direct ACO* (DACO) [KON 06] met en œuvre deux types de phéromones : l'un pour les valeurs moyennes et l'autre pour les écarts-types. Ces grandeurs

sont utilisées par les fourmis pour créer de nouvelles solutions et elles sont actualisées selon le processus ACO classique.

A chaque variable x_i est associée une distribution $N(\mu_i, \sigma_i^2)$.

Chaque itération comporte une phase d'évaporation et une phase d'intensification. L'évaporation est opérée classiquement :

$$\begin{array}{l} \vec{\mu}(t) = (1-\rho)\vec{\mu}(t-1) \\ \vec{\sigma}(t) = (1-\rho)\vec{\sigma}(t-1) \end{array} \tag{4.21}$$

en désignant par $\rho \in [0, 1]$ le facteur d'évaporation.

La phase d'intensification est plus originale :

$$\begin{array}{l} \vec{\mu}(t) = \vec{\mu}(t) + \vec{\rho} \\ \vec{\sigma}(t) = \vec{\sigma}(t) + \vec{\rho}|\vec{x} - \vec{\mu}(t-1)| \end{array} \tag{4.22}$$

Cette phase concerne la meilleure solution x rencontrée au cours de l'itération.

4.3.7. *L'algorithme CANDO*

L'algorithme CANDO (*Charged ANt colony for Dynamic Optimization*) reprend le principe de l'algorithme ACO-continu, pour traiter des problèmes d'optimisation continue dynamique, où la fonction objectif varie au cours du temps [TFA 08].

Dans cet algorithme, des charges électrostatiques répulsives ou attractives sont attribuées aux différentes fourmis, pour réaliser une diversification continue efficace de la recherche au cours du temps. Plus précisément, la fourmi i reçoit la charge a_i suivante :

$$a_i = \sum_{l=1; l \neq i}^{k} a_{il} \tag{4.23}$$

$$a_{il} = \begin{cases} \dfrac{Q_i Q_l}{|x_i - x_l|^3}(x_i - x_l) & \text{si } r_c \leq |x_i - x_l| \leq r_p \\ \dfrac{Q_i Q_l}{r_c^2 |x_i - x_l|}(x_i - x_l) & \text{si } |x_i - x_l| < r_c \\ 0 & \text{si } r_p < |x_i - x_l| \end{cases} \tag{4.24}$$

où Q_i et Q_l sont les charges initiales des fourmis i et l, $|x_i - x_l|$ est leur distance euclidienne, r_p et r_c sont les rayons de « perception » et de « noyau » respectivement. Le rayon de perception est attribué à chaque fourmi, alors que le rayon de noyau est le rayon de perception de la meilleure fourmi repérée dans l'itération courante.

Les fourmis artificielles sont dispersées dans l'espace de recherche. Les valeurs des charges, qui peuvent être positives ou négatives, sont ajustées pendant l'exécution de l'algorithme, en fonction de la qualité de la meilleure solution rencontrée.

L'algorithme a été testé sur une batterie de fonctions de test dynamiques intégrée dans la plate-forme de test OMetah [DRE 07], disponible sous licence libre.

4.4. Approches hybrides

Nous avons déjà présenté des méthodes utilisant des fourmis artificielles conjointement à d'autres techniques ou métaheuristiques (par exemple l'algorithme CACO du paragraphe 4.1.1). En mettant à part les cas où l'hybridation concerne les fourmis et des méthodes d'optimisation locale, c'est-à-dire des heuristiques plus ou moins spécialisées, dédiées au problème considéré, nous présentons dans la suite quelques exemples de coopération entre les métaheuristiques.

4.4.1. *Fourmis et évolution artificielle*

Une approche, que nous nommerons CSUACA (*Continuous Space Using Ant Colony Algorithm*), utilisant à la fois les techniques de colonie de fourmis et de type évolutionnaire, a été proposée par C. Ling *et al.* [LIN 02] pour résoudre un problème d'optimisation à variables continues dans $\mathbb{R}^n$. Dans ce travail, chaque fourmi construit un vecteur de $\mathbb{R}^n$ représentant une solution, et ceci composante par composante. Le choix d'une valeur pour une composante est fait parmi une population de valeurs possibles pour cette composante, en utilisant classiquement des phéromones (la probabilité de choisir une valeur augmente avec la quantité de phéromone présente sur cette valeur et le renforcement des phéromones dépend de la performance de la solution complète construite par la fourmi). L'originalité de ce travail réside dans la gestion des valeurs possibles pour une composante : si la population de fourmis est de taille m, pour une composante de la solution donnée, les m fourmis choisissent parmi les m valeurs possibles issues de l'itération précédente. A l'origine, pour chacune des n composantes d'un vecteur solution, m valeurs sont générées aléatoirement. Ensuite, à chaque itération (ou, devrait-on dire, génération), les valeurs sélectionnées par les fourmis sont croisées, mutées (de façon adaptative) et les m valeurs les plus marquées en phéromone sont conservées pour l'itération suivante. Ainsi, du point de vue des algorithmes évolutionnaires, ce système se compose de n populations indépendantes (chacune contient les valeurs candidates pour une dimension), chaque population est composée de m individus-valeurs, dont la survie dépend de la dynamique de dépôt des phéromones. Le dépôt de phéromone est fait dans le style de l'algorithme AS. Malheureusement, une expérimentation complète de l'algorithme reste à mener.

4.4.2. *L'algorithme ACO-LM*

Il s'agit d'une variante de ACO-continu. Chaque solution générée par l'algorithme ACO-continu est améliorée en exécutant une seule itération améliorante de la méthode de descente locale de Levenberg-Marquardt (d'où le nom de ACO-LM), qui requiert un calcul de gradient [BLU 05].

4.4.3. *L'algorithme* Improved *ACO*

Cet algorithme[CHE 05] est construit par hybridation de ACO avec un algorithme génétique. Des opérateurs de croisement adaptatif et de mutation sont appliqués à des groupes de fourmis, qui sont constitués classiquement, en exploitant des traces de phéromone.

4.4.4. *L'algorithme COAC*

L'algorithme *Continuous Orthogonal Ant Colony* (COAC) [ZHA 06] est obtenu par hybridation de ACO avec la méthode de « conception orthogonale ». Cette dernière est une technique éprouvée pour le traitement des plans d'expérience, et en particulier les « plans factoriels ». Elle est applicable ici en assimilant chaque variable du problème d'optimisation à un « facteur ».

L'algorithme COAC comporte les étapes suivantes :

– pour chaque variable continue indépendante de la fonction à optimiser, les valeurs possibles sont affectées aléatoirement à un certain nombre de nœuds d'un graphe ;

– les dépôts de phéromone de ACO sont effectués sur les nœuds ;

– un chemin solution est constitué par chaque fourmi en choisissant, pour chaque variable indépendante, l'un des nœuds disponibles ;

– le meilleur chemin obtenu est amélioré dans un second temps, *via* la procédure de conception orthogonale.

4.4.5. *L'algorithme PSACO*

L'algorithme PSACO repose sur une hybridation de ACO avec un algorithme d'optimisation par essaim particulaire [SHE 07]. Le rôle confié ici aux fourmis artificielles est celui de recherches locales, menées autour des meilleures solutions découvertes par les particules de l'essaim.

4.4.6. *L'algorithme* Immunity-based *ACO*

Cet algorithme est fondé sur une hybridation de ACO avec un algorithme immunitaire [FEN 04].

Algorithme	**Référence**	**Caractéristique principale**
AAC	[LI 03]	les fourmis utilisent un graphe et des phéromones pour générer des chaînes binaires
ACO-continu	[SOC 04]	les fourmis utilisent une combinaison de gaussiennes pour générer des valeurs réelles dans chaque dimension
ACO-LM	[SOC 06]	ACO-continu avec un algorithme de gradient
ACO_R	[SOC 08]	utilisation de fonctions de densité de probabilité gaussienne avec mémoire des meilleures solutions rencontrées
API	[MON 00c]	les fourmis utilisent une mémoire individuelle pour gérer leur effort d'exploration/exploitation
API-HMM	[SOU 01]	l'algorithme API est utilisé pour apprendre des MMC qui génèrent des chaînes binaires
APS	[TSU 04]	ressemble à ACO-continu avec la notion de phéromone d'agrégation
AS_b, ACS_b	[MON 00b]	les fourmis utilisent un graphe et des phéromones pour générer des chaînes binaires
BAS	[KON 05]	les fourmis utilisent un graphe et des phéromones pour générer des chaînes binaires
CACO	[BIL 95]	les fourmis déposent des phéromones sur des régions de l'espace de recherche
CACS	[POU 04]	les fourmis utilisent une combinaison de gaussiennes pour générer des valeurs réelles dans chaque dimension
CANDO	[TFA 08]	reprend ACO-continu en ajoutant une attraction/répulsion entre les fourmis
CIAC, HCIAC	[DRE 04]	les fourmis utilisent de la communication directe et indirecte
COAC	[ZHA 06]	hybridation de ACO et de la conception orthogonale
Immunity-based ACO	[FEN 04]	hybridation de ACO avec un algorithme immunitaire
CSUACA	[LIN 02]	les fourmis sélectionnent, par phéromone, des valeurs pour chaque dimension, qui sont modifiées par mutation adaptative
DACO	[KON 06]	chaque variable est modélisée par une distribution normale et deux types de phéromones sont employées pour la moyenne et l'écart-type
MACACO	[FRA 08]	prise en compte de la covariance
PSACO	[SHE 07]	hybridation de ACO avec PSO

Tableau 4.1. *Tableau récapitulatif des algorithmes présentés dans ce chapitre*

4.5. Comparaison des approches concurrentes

Des comparaisons empiriques partielles entre certaines des méthodes précédentes ont été décrites dans les papiers cités plus haut. Elles reposent sur des tests menés sur des batteries de fonctions analytiques de la littérature. Une étude plus exhaustive et statistiquement plus rigoureuse reste cependant à mener.

Le travail comparatif le plus complet est à notre connaissance celui présenté dans le mémoire de master de O. Kovarik, de l'université de Prague [KOV 06]. L'auteur conclut à la supériorité de ACO-continu et de DACO, sur la base de tests menés sur sept applications industrielles. Les performances de ces deux algorithmes de colonies de fourmis restent toutefois en deçà de celles d'une méthode de type Quasi-Newton, qui est disponible dans la plate-forme de test exploitée par O. Kovarik.

4.6. Conclusion

Nous avons présenté dans ce chapitre de nombreux travaux entrant dans le cadre des problèmes d'optimisation possédant des variables continues. Le tableau 4.1 vise à synthétiser ce tour d'horizon. Comme nous l'avons parfois constaté, les algorithmes proposés s'éloignent beaucoup de l'origine de la métaphore des fourmis pour se rapprocher des techniques d'optimisation développées dans les domaines voisins (algorithmes évolutionnaires, etc.). Il est fort probable que d'autres algorithmes à base de fourmis soient prochainement proposés, notamment dans des conditions de problèmes particuliers (multi-objectifs, dynamiques, etc.).

4.7. Bibliographie

[ANT 08] Antoine V., Monmarché N., Slimane M., « Data Clustering with Artificial Ants : the API algorithm case study », *Proceedings of META'08*, Hammamet, Tunisie, 29-31 octobre 2008.

[BIL 95] Bilchev G., Parmee I., « The Ant Colony Metaphor for Searching Continuous Design Spaces », T. C. Fogarty (dir.), *Proceedings of the AISB Workshop on Evolutionary Computation*, p. 25–39, 3-4 avril 1995.

[BLU 05] Blum C., Socha K., « Training feed-forward neural networks with ant colony optimization : An application to pattern classification », *Proceedings of Hybrid Intelligent Systems Conference (HIS-2005)*, p. 233–238, 2005.

[CHE 05] Chen L., Shen J., Qin L., Fan J., « A Method for Solving Optimization Problem in Continuous Space Using Improved Ant Colony Algorithm », Y. Shi, W. Xu, Z. Chen (dir.), *Proceedings of Data Mining and Knowledge Management (DMKM)*, vol. 3327 de *Lecture Notes in Computer Science*, p. 61–70, Springer-Verlag, Berlin, Heidelberg, 2005.

[DOE 08] Doerr B., Johannsen D., Tang C. H., « How Single Ant ACO Systems Optimize Pseudo-Boolean Functions », *Parallel Problem Solving from Nature – PPSN X*, vol. 5199

de *Lecture Notes in Computer Science*, p. 378–388, Springer-Verlag, Berlin, Heidelberg, 2008.

[DRE 04] Dréo J., Siarry P., « Continuous interacting ant colony algorithm based on dense heterarchy », *Future Generation Computer Systems (Special issue : Computational chemistry and molecular dynamics)*, vol. 20, n° 5, p. 841–856, juin 2004.

[DRE 07] Dréo J., Aumasson J., Tfaili W., Siarry P., « Adaptive learning search, a new tool to help comprehending metaheuristics », *International Journal on Artificial Tools*, vol. 16, n° 3, p. 483–505, 2007.

[FEN 04] Feng Y., Feng Z., « An immunity-based ant system for continuous space multi-modal function optimization », *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, p. 1050–1054, Shanghai, 26-29 août 2004.

[FRA 08] Franca F., Coelho G., Von Zuben F., Attux R., « Multivariate ant colony optimization in continuous search spaces », *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, p. 9–16, ACM New York, NY, Etats-Unis, 2008.

[GOL 89] Goldberg D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[HAN 04] Hansen N K. S., « Evaluating the CMA evolution strategy on multimodal test functions », Y. Xin, *et al.* (dir.), *Parallel Problem Solving from Nature - PPSN VIII*, p. 282–291, Springer-Verlag, Berlin, 2004.

[HO 06] Ho S., Yang S., Ni G., Machado J., « A modified ant colony optimization algorithm modeled on tabu-search methods », *IEEE Transactions on Magnetics*, vol. 42, n° 4, p. 1195–1198, avril 2006.

[KON 05] Kong M., Tian P., « A Binary Ant Colony Optimization for the Unconstrained Function Optimization Problem », *Computational Intelligence and Security*, vol. 3801 de *Lecture Notes in Computer Science*, p. 682–687, Springer-Verlag, Berlin, Heidelberg, 2005.

[KON 06] Kong M., Tian P., « A Direct Application of Ant Colony Optimization to Function Optimization Problem in Continuous Domain », *Ant Colony Optimization and Swarm Intelligence*, vol. 4150 de *Lecture Notes in Computer Science*, p. 324–331, Springer-Verlag, Berlin, Heidelberg, 2006.

[KOV 06] Kovářík O., Ant Colony Optimization for Continuous Problems, Master thesis, Czech Technical University in Prague, The Faculty of Electrical Engineering, janvier 2006.

[LER 07] Le Riche R., Schoenauer M., Sebag M., « Un état des lieux de l'optimisation évolutionnaire et de ses implications en sciences pour l'ingénieur », P. Breitkopf, C. Knopf-Lenoir (dir.), *Modélisation Numérique : défis et perspectives*, vol. 2 de *Traité Mécanique et Ingénierie des Matériaux*, p. 187–259, Hermes, février 2007.

[LI 03] Li Y.-J., Wu T.-J., « An adaptive ant colony system algorithm for continuous-space optimization problems », *Journal of Zhejiang University Science*, vol. 4, n° 1, p. 40–46, 2003.

[LIN 02] Ling C., Jie S., Ling Q., Hongjian C., « A Method for Solving Optimization Problem in Continuous Space Using Ant Colony Algorithm », M. Dorigo, G. Di Caro, M. Sampels (dir.), *Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002)*,

vol. 2463 de *Lecture Notes in Computer Science*, p. 288–289, Bruxelles, Belgique, Springer-Verlag, 12-14 septembre 2002.

[LUH 08] Luh G.-C., Lin C.-Y., « Optimal design of truss structures using ant algorithm », *Structural and Multidisciplinary Optimization*, vol. 36, p. 365–379, 2008.

[MAT 00] Mathur M., Karle S., Priye S., Jayaraman V., Kulkarni B., « Ant Colony Approach to Continuous Function Optimization », *Ind. Eng. Chem. Res.*, vol. 39, n° 10, p. 3814–3822, 2000.

[MON 00a] Monmarché N., Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation, thèse de doctorat, Laboratoire d'Informatique, Université de Tours, décembre 2000.

[MON 00b] Monmarché N., Ramat E., Desbarats L., Venturini G., « Probabilistic Search with Genetic Algorithms and Ant Colonies », A. Wu (dir.), *Proceedings of the Optimization by Building and Using Probabilistic Models workshop, Genetic and Evolutionary Computation Conference*, p. 209–211, Las Vegas, Nevada, 8-12 juillet 2000.

[MON 00c] Monmarché N., Venturini G., Slimane M., « On how *Pachycondyla apicalis* ants suggest a new search algorithm », *Future Generation Computer Systems*, vol. 16, n° 8, p. 937–946, 2000.

[POU 04] Pourtakdoust S. H., Nobahari H., « An Extension of Ant Colony System to Continuous Optimization Problems », M. Dorigo, M. Birattari, C. Blum, L. Gambardella, F. Mondada, T. Stützle (dir.), *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, vol. 3172 de *Lecture Notes in Computer Science*, p. 294–301, Bruxelles, Belgique, Springer-Verlag, 5-8 septembre 2004.

[SHE 07] Shelokar P., Siarry P., Jayaraman V., Kulkarni B., « Particle swarm and ant colony algorithms hybridized for improved continuous optimization », *Applied Mathematics and Computation*, vol. 188, p. 129–142, 2007.

[SIU 08] Siu-Lau H., Yang S., « A Computationally Efficient Vector Optimizer Using Ant Colony Optimization Algorithm for Multiobjective Designs », *IEEE Transactions on Magnetics*, vol. 44, n° 6, p. 1034–1037, juin 2008.

[SOC 04] Socha K., « ACO for Continuous and Mixed-Variable Optimization », M. Dorigo, M. Birattari, C. Blum, L. Gambardella, F. Mondada, T. Stützle (dir.), *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, vol. 3172 de *Lecture Notes in Computer Science*, p. 25–36, Bruxelles, Belgique, Springer-Verlag, 5-8 septembre 2004.

[SOC 06] Socha K., Blum C., « Ant Colony Optimization », E. Alba, R. Martí (dir.), *Metaheuristic Procedures for Training Neutral Networks*, vol. 36 de *Operations Research/Computer Science Interfaces Series*, p. 153–180, Springer-Verlag, New York, 2006.

[SOC 08] Socha K., Dorigo M., « Ant colony optimization for continuous domains », *European Journal of Operational Research*, vol. 185, n° 3, p. 1155–1173, 2008.

[SOU 01] Soukhal A., Monmarché N., Laügt D., Slimane M., « How Hidden Markov Models can help Artificial Ants to Optimize », *Proceedings of the Optimization by Building and Using Probabilistic Models workshop, Genetic and Evolutionary Computation Conference*,

p. 226–229, 2001.

[TFA 08] Tfaili W., Siarry P., « A new charged ant colony algorithm for continuous dynamic optimization », *Applied Mathematics and Computation*, vol. 197, n° 2, p. 604–613, 2008.

[TSU 04] Tsutsui S., « Ant Colony Optimization for continuous domains with aggregation pheromones metaphor », *5th International Conference on Recent Advances in Soft Computing (RASC'04)*, p. 207–212, Nottingham, Royaume-Uni, 2004.

[WOD 97] Wodrich M., Bilchev G., « Cooperative distributed search : the ant's way », *Control & Cybernetics*, vol. 26, p. 413–445, 1997.

[ZHA 06] Zhang J., Chen W.-N., Zhong J.-H., Tan X., Li Y., « Continuous Function Optimization Using Hybrid Ant Colony Approach with Orthogonal Design Scheme », *Simulated Evolution and Learning*, vol. 4247 de *Lecture Notes in Computer Science*, p. 126–133, Springer-Verlag, Berlin, Heidelberg, 2006.

Chapitre 5

Optimisation par colonie de fourmis pour la configuration en programmation par contraintes

5.1. Introduction

La *configuration* en programmation par contraintes est un formalisme permettant de représenter des problèmes combinatoires. L'objectif est de générer des structures sous contraintes, soit un ensemble de composants typés et interconnectés. Ces recherches ont pour but d'augmenter la taille des problèmes de configuration pouvant être traités par une recherche énumérative des solutions. Des domaines d'application récents, comme la configuration de machines, le traitement du langage ou le *web* sémantique, ont mis à jour la nécessité pour les solveurs de configuration de traiter un grand nombre de composants.

Une des difficultés inhérentes à la recherche énumérative est l'explosion combinatoire. L'espace de recherche de ces problèmes augmente en effet très rapidement avec le nombre de composants. De plus, dans le cas général, des problèmes ayant un nombre infini de modèles finis peuvent facilement être construits, ce qui indique clairement les limites des méthodes de recherche exhaustives. Une alternative connue dans la communauté SAT/CSP pour juguler cette explosion dans le cas de problèmes satisfiables est l'utilisation de procédures incomplètes. Parmi les algorithmes incomplets qui tentent de résoudre ce problème, l'optimisation par colonie de fourmis (ACO : *Ant Colony Optimization*), qui combine des méthodes aléatoires et heuristiques avec

Chapitre rédigé par Patrick Albert, Laurent Henocque et Mathias Kleiner.

l'apprentissage par renforcement, a prouvé son efficacité sur de nombreux problèmes de satisfaction de contraintes (CSP : *Constraint Satisfaction Problem*).

A notre connaissance, l'utilisation de méthodes incomplètes pour la configuration n'a pas encore été étudiée. Nous décrivons comment la nature des problèmes non bornés de configuration influe sur l'approche ACO, notamment à cause de la présence de variables ensemblistes et de domaines ouverts. Ce chapitre propose une extension générique de l'optimisation par colonie de fourmis aux problèmes de configuration sous contraintes. Nous fournissons des résultats expérimentaux, tant pour la satisfiabilité que pour l'optimisation, sur des problèmes aléatoires et sur un *benchmark* historique de la configuration : le problème des *racks*.

La section courante est une brève introduction à la configuration et la métaheuristique ACO. La section 5.2 détaille un cadre ACO pour la configuration. Nous proposons tout d'abord un modèle phéromonal original qui adapte ACO aux variables ensemblistes sur domaines ouverts. Nous présentons ensuite un algorithme permettant d'exploiter ce modèle pour construire des solutions. La section 5.3 décrit les conditions expérimentales et fournit des résultats. Nous montrons notamment l'utilisation de PSO (*Particule Swarm Optimisation*) pour converger vers de bons jeux de paramètres pour notre implémentation. La section 5.4 conclut et ouvre les perspectives de l'approche.

5.1.1. *Brève introduction à la configuration*

Configurer consiste à simuler la création d'un produit complexe à partir de *composants* choisis dans un *catalogue de types*. Dans le contexte général, ni le nombre ni le type des composants requis ne sont connus à l'avance. Les composants sont connectés par des *relations*, et leur type est soumis à l'*héritage*. Des *contraintes* définissent les produits valides. Un configurateur prend en entrée un fragment de la structure ciblée et l'étend à une solution du problème, si une solution existe, en ajoutant tous les éléments nécessaires durant la recherche. Ce problème de logique du premier-ordre est semi-décidable dans le cas général. Le lecteur pourra trouver dans [JUN 06] une introduction complète à la configuration.

Pratiquement, un problème de configuration nécessite la définition d'un modèle objet sous contraintes, un concept qui est bien représenté par un modèle objet selon la notation graphique d'UML. La figure 5.1 en donne un exemple. Les modèles objets ne donnent pas d'indication sur la manière dont les instances valides de tels modèles peuvent être produites. Généralement, les modèles servent à valider par de simples tests des instances produites par des opérateurs extérieurs, qui n'utilisent pas le modèle de façon automatique. Configurer consiste précisément à exploiter automatiquement un modèle objet pour en produire des instances satisfaisant certains critères, parfois de façon optimale. Cette approche « automatique » possède des applications industrielles

(configuration de produit) et en intelligence artificielle (reconnaissance de langages, composition de *workflows* notamment). D'un point de vue pratique, la production d'instances est un moyen de valider formellement un modèle, un problème qui intéresse la communauté de modélisation orientée objet. Les articles [GOG 03, GOG 07] évoquent ASSL, une méthode de génération d'instances qui n'est pas générique, mais qui se rapproche des enjeux de la configuration.

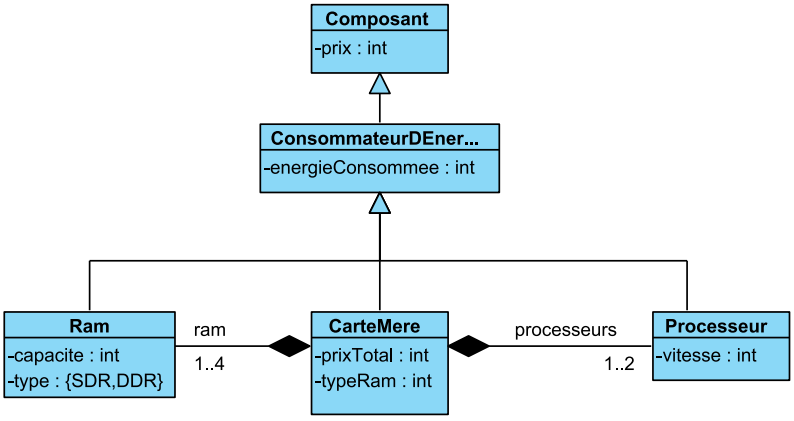


Figure 5.1. *Un modèle objet, extrait de la structure interne des ordinateurs*

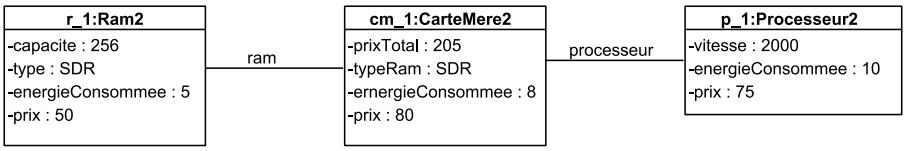


Figure 5.2. *Un graphe solution du problème de configuration de la figure 5.1*

La figure 5.2 donne un exemple d'instance « solution » du modèle décrit par la figure 5.1. Une solution est bien représentée par un graphe, dont les éléments connectés désigne cette fois des objets, et non plus des classes. Ici encore, les langages de modélisation graphique comme UML ne donnent pas d'indication sur la manière technique de décrire les liens entre les objets. Selon les cas, et les possibilités, cela peut se faire en utilisant des tables dans des bases de données relationnelles, ou bien des pointeurs intégrés aux objets, ou bien des pointeurs vers des constructions intermédiaires (tableaux, ensembles, etc.). Bien entendu, toute approche « automatique » du problème de génération d'instances pour de tels modèles requiert de faire un choix technique pour la représentation des relations.

Nous considérons une approche dans laquelle les relations sont décrites grâce à des *variables de ports* au niveau des composants : une variable de port modélise les

composants distants auxquels un objet est connecté pour une relation donnée. Le terme de « port » a été popularisé par les CSP génératifs [FLE 98, STU 93].

5.1.2. *Presentation formelle du problème*

Soit U un ensemble (potentiellement infini) d'identifiants d'objets, appelé *univers* ou *domaine des objets*. Par la suite, les éléments de U seront appelés composants ou objets. U peut être interprété par (ou sans perte de généralité être identifié à) l'ensemble $\mathbb{N}$ des entiers.

Définition 5.1.– *Un* modèle de configuration *est défini par :*

– trois ensembles mutuellement disjoints T (noms des types), A (noms des attributs) et P (noms des ports) ;

– un ensemble D de toutes les valeurs possibles des attributs ;

– une application supertypes : $T \mapsto \mathbb{P}T$ *modélisant l'héritage : pour chaque type $t \in T$, $supertypes(t)$ est l'ensemble de ses parents directs ;*

– une application attributes : $T \mapsto \mathbb{P}A$ *modélisant la structure d'une classe : pour chaque type $t \in T$, $attributes(t)$ est l'ensemble des attributs de la classe t ;*

– une application ports : $T \mapsto \mathbb{P}P$ *modélisant les connexions d'une classe : pour chaque type $t \in T$, $ports(t)$ est l'ensemble des ports de la classe t ;*

– une application domain : $A \mapsto \mathbb{P}D$ *modélisant les domaines d'un attribut : pour chaque attribut $a \in A$, $domain(a)$ est le domaine des variables CSP modélisant chaque occurrence de l'attribut dans les objets ;*

– une application type : $P \mapsto T$ *modélisant le type des objets pouvant être connectés* via *un port donné ;*

– une application $card_{min} : P \mapsto \mathbb{N}$ et une fonction partielle $card_{max} : P \mapsto \mathbb{N}$ modélisant la cardinalité minimum (et éventuellement maximum) des ports ;

– un ensemble de contraintes C : prédicats de premier ordre ou supérieur sur le vocabulaire $T \cup A \cup P \cup U \cup D$;

Pour chaque objet $o \in U$: le type de o est modélisé par une variable CSP $t(o)$ sur le domaine T (ou $\mathbb{P}T$ dans certaines implémentations) ; chaque attribut $a \in attributes(t(o))$ est modélisé par une variable CSP $a(o)$ sur le domaine $domain(a)$; chaque port $p \in ports(t(o))$ est modélisé par une variable ensembliste CSP $p(o)$ sur le domaine $\mathbb{P}U$ sous la contrainte que chaque élément de $p(o)$ appartient à $type(p)$. Par la suite, nous appellerons port, attribut ou type, la variable CSP correspondante d'un objet lorsque cela n'est pas ambigu.

DÉFINITION 5.2.– *Un ensemble de composants (un sous-ensemble de U) dont le type, les attributs et les ports sont instanciés et tel que toutes les contraintes de C soient satisfaites, est appelé une* instance de configuration *ou plus simplement une* configuration *du modèle de configuration.*

D'un point de vue logique, une configuration est un *modèle* du modèle de configuration. Pour éviter la confusion nous utiliserons donc les termes *instance* ou *configuration*.

DÉFINITION 5.3.– *Un* problème de configuration *est composé :*

– d'un modèle de configuration CM ;

– d'une requête R contenant un ensemble de contraintes additionnelles et/ou de préférences.

Un configurateur ou procédure de résolution *doit produire un ou plusieurs modèles de* $CM \wedge R$ *s'il en existe.*

Résoudre le problème d'énumération associé peut se faire en utilisant différents formalismes ou approches techniques : extensions du paradigme CSP [AMI 02] [MIT 90, SAB 96, STU 93], approches à base de connaissances [STU 97], logiques terminologiques (ou de description) [BRA 85, MCG 98], programmation logique (chaînage avant ou arrière, et sémantique non standard) [SOI 01], approches orientées objet [JUN 03].

Ces techniques ont été utilisées avec succès sur un certain nombre de problèmes industriels. Cependant, ces approches peuvent difficilement traiter l'explosion combinatoire qui accompagne l'augmentation des données en entrée. De plus, la plupart d'entre elles posent des restrictions sur le contexte de configuration, en bornant le nombre de composants disponibles et/ou les cardinalités des relations.

Les approches basées sur des extensions des CSP, telles que les CSP génératifs [FLE 98, STU 93], peuvent, quant à elles, traiter un contexte de configuration non borné :

– l'instantiation du type et des attributs d'un composant est modélisée par une variable CSP classique : choix d'une valeur unique dans un domaine fini et discret ;

– l'instantiation d'un port de composant peut susciter la génération dynamique de nouveaux composants, afin de satisfaire les contraintes de cardinalité (min). La cardinalité d'un port $card(p)$ consiste en un choix de valeur unique dans un domaine discret et potentiellement non borné. Les cibles sont choisies dans un ensemble de composants du type destination $type(p)$, modélisé par une variable ensembliste sur un domaine également non borné. Le choix des cibles peut se faire en sélectionnant itérativement $card(p)$ valeurs différentes dans l'ensemble des composants de type $type(p)$, après en avoir éventuellement introduit de nouveaux.

Une procédure de recherche dans le cas des CSP génératifs démarre avec un ensemble de composants à instancier, contenant au moins un composant *racine*. Chaque fois qu'un composant est sélectionné, ou créé dynamiquement à travers l'instantiation d'un port, il est ajouté à la liste. Pour finir sur cette introduction, les problèmes de configuration, tout comme les CSP, peuvent être des problèmes de satisfaction ou d'optimisation.

5.1.3. *Brève introduction à ACO*

Les recherches sur le comportement des colonies de fourmis ont montré que leur communication est essentiellement basée sur la production et la détection d'agents chimiques appelés *phéromones*. Parmi les différents types de phéromones, certaines sont déposées sur le sol. Un chemin marqué par de telles phéromones, par exemple d'une source de nourriture au nid, est alors suivi par les autres fourmis avec des fluctuations qui sont aléatoires ou dues à la manière dont la fourmi perçoit son environnement et y réagit (sa « visibilité locale »). Ces phéromones sont également volatiles. Leur évaporation progressive permet de diversifier les chemins empruntés ou d'en explorer de totalement nouveaux. Des expérimentations biologiques et des résultats théoriques ont montré que les fourmis utilisent ces phéromones pour trouver le chemin le plus court vers une source de nourriture, et donc de ce fait résolvent un problème d'optimisation.

Déterminer le chemin le plus court est en effet une tâche similaire à de nombreux problèmes combinatoires tels que le problème du voyageur de commerce. Ceci a conduit les chercheurs à s'inspirer du comportement des fourmis pour élaborer de nouveaux algorithmes de résolution [DOR 97]. L'approche fut ensuite étendue à une *métaheuristique* pour les *problèmes d'optimisation discrets* [DOR 99] connue sous le nom de *Ant Colony Optimization*. En plus des chapitres 2 et 3, [DOR 05] présente une étude complète d'ACO.

5.1.3.1. *Métaheuristique ACO et algorithmes*

M. Dorigo propose dans [DOR 99] une métaheuristique ACO et un algorithme (*Ant System*) pour les problèmes d'optimisation discrets combinatoires. Il existe de nombreux travaux sur des variantes d'*Ant System*. Une des améliorations les plus connues est *MAX-MIN Ant System* (MMAS) [STU 00]. La modélisation du problème est associée dans ACO à un modèle phéromonal : une valeur phéromonale est associée à chaque affectation possible d'une valeur à une variable X_i. Formellement, la valeur phéromonale τ_{ij} est associée au *composant de solution* c_{ij} , qui consiste en une affectation $X_i = v_i^j$. L'ensemble de tous les composants de solution potentiels est appelé C.

Une fourmi artificielle construit une solution à partir d'une instantiation partielle vide $s^p = \emptyset$. Elle est étendue à chaque étape de la construction en ajoutant un composant de solution c_{ij} de l'ensemble $N(s^p) \subseteq C$, où $N(s^p)$ est l'ensemble des composants

pouvant être ajoutés, sans violer les contraintes du problème. Le choix du composant est un choix probabiliste influencé par les phéromones et les heuristiques (correspondant à la visibilité locale d'une fourmi biologique). Cette probabilité peut varier en fonction des algorithmes ACO. Dans *Ant System*, la probabilité de choisir un composant c_{ij} pour une fourmi k est définie par :

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha}.\eta_{ij}^{\beta}}{\sum_{c_{il} \in N(s^p)} \tau_{il}^{\alpha}.\eta_{il}^{\beta}}$$

où les paramètres α and β contrôlent l'importance relative des phéromones par rapport à l'information heuristique η_{ij}.

Un *graphe de construction* $G_C(V, E)$ peut être obtenu à partir de C, dans lequel les sommets V et les arêtes E sont les composants de solution. Une fourmi artificielle construit une solution en traversant ce graphe (complet) de construction de sommet en sommet. De plus, les fourmis déposent une quantité $\Delta\tau$ de phéromone sur les composants dépendant de la qualité de la solution.

Les ACO sont utilisés pour résoudre des CSP dans [SOL 02]. L'algorithme présenté inclut des améliorations provenant de la variante *MAX-MIN Ant System*. Cependant, contrairement aux ACO classiques, le choix probabiliste ne dépend pas de la dernière variable instanciée, car tous les sommets déjà visités sont d'égale importance. L'heuristique de choix de valeur est inversement proportionnelle au nombre de contraintes violées. L'heuristique de choix de variable est classique en optimisation, celle du plus petit domaine.

5.2. ACO pour la configuration

Dans le contexte général de la configuration, nous avons vu que la modélisation s'appuie en partie sur des variables ensemblistes à domaines ouverts. Ces problèmes n'entrent donc pas dans le cadre de la définition des ACO, et ne peuvent pas non plus être résolus avec une approche CSP standard. En particulier, il n'est pas possible d'exhiber *a priori* un graphe de construction, puisque le nombre de composants d'une solution n'est pas borné. Toutefois, il existe des similitudes, et un graphe utile peut être identifié : le graphe d'interconnexion des composants. Cela fonde une intuition majeure de notre approche.

5.2.1. *Graphe de construction*

La structure d'une (instance de) configuration peut être vue comme un graphe. Dans ce *graphe de structure*, les sommets représentent les composants et les arcs les relations qu'ils entretiennent. La superposition de tous les graphes de structures créés

lors de précédentes tentatives d'un algorithme à redémarrage aléatoire reste également un graphe. Des phéromones peuvent donc être déposées sur les arcs, qui représentent alors pour une fourmi artificielle le choix d'un composant (déjà existant) en tant que cible d'un port.

Cependant, à un sommet donné de ce graphe, une fourmi a non seulement la possibilité de suivre un ou plusieurs arcs, mais peut également créer des arcs vers de nouveaux composants quand la décision est prise de générer dynamiquement une nouvelle cible pour le port considéré. Le nombre d'arcs (c'est-à-dire les cibles d'un port), suivis ou créés par une fourmi, est défini par le choix (généralement non borné) de la cardinalité du port.

Enfin, puisque chaque sommet est un composant de la configuration, les fourmis doivent le classifier (c'est-à-dire sélectionner son type) et choisir une valeur pour chacun de ses attributs (c'est-à-dire configurer le composant). Nous avons vu que ces décisions peuvent être modélisées par l'instantiation de variables CSP classiques. Les types et les attributs peuvent être ajoutés au graphe de construction en tant que sommets connectés au sommet du composant. Cette dernière représentation, similaire à celle proposée par [SOL 02], permet de fournir un point de vue homogène sur la construction d'une solution.

Le graphe de construction d'ACO pour la configuration, obtenu par combinaison de tous ces éléments, est donc généré dynamiquement durant la recherche, par superposition de toutes les instances précédemment créées. Les fourmis artificielles peuvent, guidées par les phéromones et les heuristiques, explorer et étendre ce graphe, afin de construire des solutions potentielles.

5.2.2. *Instantiation des variables*

Dans ACO, chaque décision prise par une fourmi est associée à une variable sur un domaine discret. Ce principe peut également être appliqué à la configuration dans le cas des variables à domaines fermés (CSP ou ensembliste). Cependant, en configuration, certaines variables (les ports et leurs cardinalités) peuvent avoir un domaine ouvert. Nous introduisons un mécanisme original pour traiter le choix probabiliste associé : les *simu-finite sets*.

5.2.2.1. Simu-finite sets

Nous simulons le choix d'une valeur dans un ensemble non borné *via* un ensemble fini *évolutif* appelé *simu-finite set*. Un *simu-finite set* contient un certain nombre de valeurs du domaine, plus une valeur *évolutive* (ou *wild card*) représentant toutes les autres valeurs possibles. L'ensemble évolue (c'est-à-dire est augmenté pour la prochaine itération ACO) chaque fois qu'une fourmi choisit la valeur *wild card*. L'idée d'utiliser de tels *wild cards* au premier ordre est bien connue.

Par exemple, considérons le port p, $card_{min}(p) = 2$, et le domaine associé aux choix de la cardinalité $card(p)$. Soit l'ensemble de départ $p_{card} = \{2, 3, 4, 5\}$, ou 5 est la *wild card*. Si la valeur choisie est inférieure à 5, l'ensemble reste inchangé. Si la valeur 5 est choisie, l'ensemble est modifié pour la prochaine itération en $p_{card} = \{2, 3, 4, 5, 6\}$ avec la valeur 6 comme nouvelle *wild card*. Cette *wild card* représente donc en intention l'infinité potentielle des composants disponibles.

Nous présentons d'abord une définition générale des *simu-finite sets*. Nous montrerons ensuite son application au choix de la cardinalité et des cibles d'un port dans le modèle phéromonal.

DÉFINITION 5.4.– *(*simu-finite sets*) Soit la sélection d'une valeur v dans l'ensemble fini* $V = \{v_0, \ldots, v_i, v_*\}$ *où* $\{v_0, \ldots, v_i\}$ *est un ensemble fini de valeurs différentes tel que :*

1) il est possible d'exhiber une valeur $v_{i+1} \notin V$ *appartenant au domaine ouvert de la variable considérée ;*

2) quand la valeur v_* *est choisie, l'ensemble est modifié en* $V = V \cup \{v_{i+1}\} = \{v_0, \ldots, v_{i+1}, v_*\}$ *pour la prochaine itération.*

5.2.3. *Modèle phéromonal*

Nous proposons un modèle phéromonal original permettant d'utiliser la métaheuristique ACO dans le cadre de la configuration. Nous présentons d'abord comment exploiter les *simu-finite sets* pour les décisions associées à l'instantiation des ports. Nous complétons ensuite le modèle avec les autres types de décisions (classification et attributs). Enfin, nous détaillons comment les phéromones de ce modèle sont mises à jour à chaque itération d'ACO.

5.2.3.1. *Instantiation d'un port*

L'approche consiste, pour un port p, à choisir d'abord une cardinalité $card(p)$ puis à choisir itérativement le même nombre de cibles. Nous introduisons donc un *simu-finite set* pour le choix d'une cardinalité (non bornée) et un autre pour le choix d'une cible (lorsqu'il est possible de créer dynamiquement des composants).

5.2.3.1.1. Choix de la cardinalité

Le choix d'une cardinalité non bornée est le choix d'une valeur parmi un ensemble non borné d'entiers. Nous appliquons les *simu-finite sets* comme suit :

1) soit la sélection d'une valeur $card(p)$ dans l'ensemble fini $p_{card} = \{card_{min}(p), \ldots, card_{min}(p) + i, card(p)_*\}$ où :

- $card_{min}(p)$ est la borne inférieure de la cardinalité pour le port p,
- $\{card_{min}(p), \ldots, card_{min}(p) + i\}$ est un intervalle d'entiers, c'est-à-dire $p_k = p_{k-1} + 1$,

- $i >= 0$ est une valeur d'initialisation choisie au départ,
- $card(p)_* = card_{min}(p) + i + 1$ est la *wild card* du *simu-finite set* ;

2) quand la cardinalité $card(p)_*$ est choisie, l'ensemble est modifié en $p_{card} = \{r_{card_{min}}, \ldots, card_{min}(p) + i + 1, card(p)_*\}$ avec $card(p)_* = card_{min}(p) + i + 2$ comme nouvelle *wild card*.

5.2.3.1.2. Choix d'une cible

Le caractère non borné de l'ensemble associé au choix d'une cible provient de la possibilité de créer dynamiquement des composants. Nous appliquons les *simu-finite sets* comme suit :

1) soit la sélection d'une valeur p_{t_i} dans l'ensemble fini $p_t = \{p_{t_0}, \ldots, p_{t_i}, p_{t_*}\}$ où :
- $\{p_{t_0}, \ldots, p_{t_i}\}$ est l'ensemble des composants *existants*, qui peuvent être cibles du port p,
- p_{t_*} est le choix de créer un nouveau composant (c'est-à-dire la *wild card*) ;

2) quand la cible p_{t_*} est choisie, l'ensemble est modifié en $p_t = \{p_{t_0}, \ldots, p_{t_i}, p_{t_{i+1}}, p_{t_*}\}$ où $p_{t_{i+1}}$ est le nouveau composant généré, et p_{t_*} est le choix de créer un nouveau composant. De plus, on peut décider de n'autoriser la création que d'un (ou un nombre limité de) composant(s) durant une itération (afin de favoriser l'intensification). Dans ce cas, l'ensemble modifié ne contiendra pas la valeur p_{t_*}. Enfin, puisqu'une cible ne peut être choisie qu'une seule fois pour un port donné, elle est supprimée de l'ensemble jusqu'à ce que le port soit entièrement configuré. Le choix probabiliste associé à ces deux *simu-finite sets*, maintenant équivalent au choix d'une valeur unique dans un domaine fini, est obtenu grâce à l'approche standard ACO. Nous définissons donc l'instantiation d'un port pour un problème de configuration dans le modèle phéromonal, suivi des autres types de décisions qui devront être prises par les fourmis artificielles.

5.2.3.2. *Modèle phéromonal pour les ports*

Définition 5.5.– *(*Instantiation d'un port*)*

– instancier un port p consiste à choisir une cardinalité p_{card_j} dans le simu-finite set *p_{card}, puis à choisir p_{card_j} cibles depuis le* simu-finite set *p_t ;*

– la probabilité de choisir une cardinalité p_{card_j} est :

$$p_{p_{card_j}} = \frac{\tau^{\alpha}_{p_{card_j}} . \eta^{\beta}_{p_{card_j}}}{\sum_{l=0}^{min+i} (\tau^{\alpha}_{p_{card_l}} . \eta^{\beta}_{p_{card_l}}) + \tau^{\alpha}_{p_{card_*}} . \eta^{\beta}_{p_{card_*}}}$$

où $\tau_{p_{card_j}}$ est la valeur phéromonale pour la cardinalité $card_j$ et $\eta_{p_{card_j}}$ l'information heuristique ;

– la probabilité de choisir une cible p_{t_j} est de :

$$p_{p_{t_j}} = \frac{\tau^{\alpha}_{p_{t_j}}.\eta^{\beta}_{p_{t_j}}}{\sum_{k=0}^{i}(\tau^{\alpha}_{p_{t_k}}.\eta^{\beta}_{p_{t_k}}) + \tau^{\alpha}_{p_{t_*}}.\eta^{\beta}_{p_{t_*}}}$$

où $\tau_{p_{t_j}}$ est la valeur phéromonale pour la cible p_{t_j} et $\eta_{p_{t_j}}$ l'information heuristique.

5.2.3.3. *Modèle phéromonal pour la classification*

Puisque la classification est un choix dans un ensemble fini d'éléments (l'ensemble des sous-types), il est possible d'utiliser l'approche standard.

Définition 5.6.– Classification *:*

– classifier un composant de type t_i consiste à choisir un sous-type depuis l'ensemble $finalsubtypes(t_i) = \{t_i^j, \ldots, t_i^k\}$;

– la probabilité de choisir un type t_i^j est de :

$$p_{t_i^j} = \frac{\tau^{\alpha}_{t_i^j}.\eta^{\beta}_{t_i^j}}{\sum_{l=0}^{k}\tau^{\alpha}_{t_i^l}.\eta^{\beta}_{t_i^l}}$$

où $\tau_{t_i^j}$ est la valeur phéromonale pour le type t_i^j et $\eta_{t_i^j}$ l'information heuristique.

5.2.3.4. *Modèle phéromonal pour l'instantiation des attributs*

Ici encore, nous avons un choix standard dans un ensemble fini.

Définition 5.7.– Instantiation d'un attribut *:*

– instancier une variable X_i consiste à choisir une valeur dans l'ensemble $D_i = \{x_i^0, \ldots, x_i^k\}$:

– la probabilité de choisir une valeur x_i^j est de :

$$p_{x_i^j} = \frac{\tau^{\alpha}_{x_i^j}.\eta^{\beta}_{x_i^j}}{\sum_{l=0}^{k}\tau^{\alpha}_{x_i^l}.\eta^{\beta}_{x_i^l}}$$

où $\tau_{x_i^j}$ est la valeur phéromonale pour la valeur x_i^j et $\eta_{x_i^j}$ l'information heuristique.

5.2.3.5. *Mise à jour des phéromones*

La mise à jour des phéromones, de manière similaire à l'approche de *MAX-MIN Ant System*, est basée sur la meilleure instance créée par la colonie de fourmis à chaque itération :

$$\tau_{ij} \longleftarrow \left[(1-\rho).\tau_{ij}^{best} + \Delta\tau_{ij}^{best}\right]_{\tau_{min}}^{\tau_{max}}$$

où $\Delta\tau_{ij}^{best}$ est une valeur dépendant de la qualité de la solution. Dans le cas de la satisfiabilité, la qualité peut être définie par le rapport $\frac{numberOfConstraintsFulfilled}{totalNumberOfConstraints}$.

Cependant, des questions originales se posent pour les problèmes de configuration. Nous introduisons donc dans ce qui suit un certain nombre de mécanismes supplémentaires, dont les effets positifs ont été étudiés par de nombreuses expérimentations.

Premièrement, l'intérêt de l'évaporation est d'oublier les *mauvais* choix (qui ne participent pas ou peu aux meilleures instances ultérieures). Mais, si un composant c_i ne participe pas à la meilleure instance courante, le bénéfice d'évaporer les phéromones qui lui sont associées n'est pas évident. Nous proposons donc une *évaporation restreinte* comme alternative à l'évaporation totale standard. Avec l'évaporation restreinte, les phéromones associées à c_i (classification, attributs et ports ayant c_i comme source) ne sont pas modifiées. Les phéromones associées à un port $c_j \rightarrow c_i$ sont tout de même évaporées, prenant ainsi en compte le fait que le composant n'a pas été choisi comme cible du port dans la meilleure instance courante.

Deuxièmement, nous devons considérer les phéromones associées aux *wild cards* des *simu-finite sets*. Dans le cas des cibles d'un port, la valeur associée à la création d'un composant n'est mise à jour que si la meilleure solution a effectivement créé une cible durant l'itération. La valeur modifiée est alors également utilisée pour initialiser la valeur associée au nouveau composant de l'ensemble. Dans le cas du choix de cardinalité, toutes les valeurs ajoutées dynamiquement à l'ensemble partagent la même valeur d'initialisation (définie par un paramètre).

Pour finir, différents types de choix peuvent nécessiter des paramètres différents. Chaque paramètre relatif aux phéromones est donc dupliqué pour chaque type de décision. Par exemple, la valeur minimum autorisée pour une phéromone peut être différente pour la classification ou pour le choix des cibles d'un port.

5.2.4. *Algorithmes*

Nous proposons un algorithme (ACOC$_{graph}$) qui exploite le modèle phéromonal présenté pour construire des solutions aux problèmes de configuration. ACOC$_{graph}$ est proche de l'algorithme ACO originel, mais diffère sur la façon dont les fourmis construisent les solutions. La partie commune (la métaheuristique ACO) est présentée dans la figure 5.3.

ACOC$_{graph}$ démarre par la classification d'un composant racine. Les attributs et les ports de ce composant sont instanciés, puis les cibles des ports sont classifiées.

```
function instance configure
  for i := 0 to numberOfiterations do
    for j := 0 to numberOfAnts do
      instance:=generateInstance
      quality:=evaluateInstance
      if (quality>bestQuality) then
        bestInstance:=instance
        bestQuality:=quality
    updatepheromones
    updateSimuFiniteSets
  return bestInstance
```

Figure 5.3. *La fonction* configure

Chaque cible est ensuite traitée récursivement (parcours en profondeur de la structure). L'algorithme est présenté dans la figure 5.5.

La figure 5.4 donne un exemple graphique de création d'une structure. Un nombre représente un composant classifié et un « + » représente un composant dont les attributs ont été instanciés.

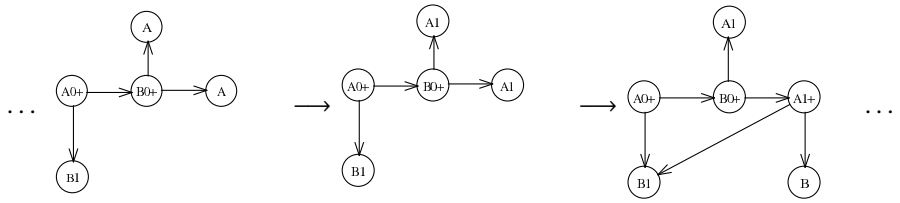


Figure 5.4. *$ACOC_{graph}$: instantiation des attributs et des ports, classification des cibles, itération sur les nouveaux composants ajoutés*

5.2.4.1. *Chemin de construction*

Nous avons montré que le graphe de construction de notre approche n'est pas calculé à l'avance mais défini par la superposition de toutes les instances générées par les fourmis précédentes. Une autre différence avec la métaheuristique ACO originelle est le *chemin de construction* suivi par les fourmis à l'intérieur de ce graphe.

Dans $ACOC_{graph}$, le chemin de construction est lié à la recherche en profondeur effectuée sur la partie structurelle du graphe de construction (c'est-à-dire les composants et les ports). Cependant, à chaque sommet, une fourmi va suivre (ou créer) une arête puis retourner au sommet précédent, jusqu'à ce que toutes les décisions liées à ce

composant soient prises et que les cibles des ports aient été configurées. Ce comportement est illustré sur la figure 5.4. Une fois que les composants « B » (partie basse) ont été configurés, la fourmi va retourner au composant central « B0+ » et se déplacer vers le composant « A1 » (partie haute).

Les choix des fourmis artificielles sont également indépendants des décisions précédentes effectuées sur l'instance courante. Dans ce sens, notre approche est similaire à l'application d'ACO aux CSP [SOL 02].

```
function instance generateInstance
    instance=initializeInstance
    constructionStack :=root
    classifyRoot
    while (constructionStack≠ ∅) do
        constructObject
        removeObjectFromConstructionStack
    return instance
procedure constructObject
    for i := 0 to numberOfAttributes do
        instantiateAttribute
    for i := 0 to numberOfPorts do
        instantiatePort
        classifyTargets
        addTargetsToConstructionStack
    return
```

Figure 5.5. *La fonction* generateInstance *pour* $ACOC_{graph}$

5.3. Implémentation et expérimentations

Le programme utilisé pour nos expérimentations a été développé en langage Java et inclut une librairie de modèles objets contraints destinée à comparer différentes méthodes de recherche sur une plate-forme commune. Ces expérimentations ont été conduites sur un Pentium IV 3.2 Ghz / 1go RAM. Nous ne décrivons que les résultats les plus significatifs issus d'une large gamme d'expérimentations.

5.3.1. *Heuristiques*

Dans ces expérimentations, nous n'avons pas utilisé d'heuristiques de valeurs. Les fourmis artificielles sont donc uniquement guidées par les phéromones. Concernant les heuristiques de choix de variables, l'ordre d'instantiation est partiellement induit par la recherche en profondeur dans le graphe de construction. Les variables de classification, d'attributs et de ports sont traitées dans un ordre aléatoire prédéfini.

5.3.2. *Paramètres*

Les paramètres ont une grande influence sur le comportement des algorithmes ACO. Ils sont soit liés à l'algorithme (nombre d'itérations, nombre de fourmis, heuristiques, satisfiabilité ou optimisation) soit aux phéromones (valeurs d'initialisation, taux d'évaporation, poids des phéromones par rapport aux heuristiques, paramètres propres aux *simu-finite sets*). Nous considérons les paramètres suivants :

– nbIte pour le nombre maximum d'itérations ;

– nbAnts pour le nombre de fourmis à chaque itération ;

– pMax (pMin) pour la valeur maximum (minimum) d'augmentation des phéromones ;

– ρ pour le taux d'évaporation ;

– evaS pour le choix entre évaporation totale ou restreinte ;

– ObjBV, ObjMin, ObjMax pour respectivement les valeurs d'initialisation, minimum et maximum des phéromones associées aux ports ;

– RelBV(...), AttBV(...), ClassBV(...) : idem pour les cardinalités des ports, les attributs et les types ;

– NOBV pour la valeur d'initialisation des *wild card* associées aux ports.

Les paramètres constituent un problème récurrent dans les ACO, et sont liés à l'équilibre entre exploration et intensification. En configuration, en plus de s'appliquer aux attributs et types des objets participants (similaires à une affectation dans les CSP), ils influencent également la croissance en taille des solutions. Ce point est primordial, puisque les structures sont de taille *a priori* inconnue. Le grand nombre de paramètres est également une difficulté et requiert des techniques avancées pour leur identification et leur optimisation. Bien sûr, on tente d'isoler des jeux de paramètres qui possèdent une portée générale et qui ne sont pas spécifiques à un problème donné.

5.3.2.1. *Optimisation par essaim de particules*

L'espace multidimensionnel des jeux de paramètres est tellement vaste qu'il est impossible de l'explorer exhaustivement par programme. Nous avons choisi d'utiliser l'optimisation par essaim de particules (PSO) : une technique d'optimisation stochastique à base de population. Elle fut développée pour la première fois dans [EBE 95]. Intuitivement, chaque jeu de paramètres définit la position d'une particule, qui se déplace dans l'espace avec une certaine vitesse. La meilleure des particules joue le rôle de pôle d'attraction, et infléchit les trajectoires des autres, qui restent partiellement attirées par leur meilleure position passée. On vise ainsi à converger vers des *optima* locaux des jeux de paramètres. Un *problème PSO* est défini par une fonction objectif sur un espace multidimensionnel $f : \mathbb{R}^m \rightarrow \mathbb{R}$, avec $[min_j, max_j]$, $0 <= j < m$ comme borne des domaines pour chaque dimension j. Soit *n particules* p_i, $0 <= i < n$. Chaque particule représente une solution potentielle et est définie par une *position* $x_i \in \mathbb{R}^m$ et

une *vitesse* $v_i \in \mathbb{R}^m$. La position est l'affectation courante. La vitesse est la direction courante de la particule dans l'espace du problème. Chaque particule a la connaissance de sa meilleure position et de la meilleure position globale.

La figure 5.6 présente un algorithme PSO. ω est une constante d'inertie, généralement de valeur légèrement inférieure à 1 et décroissante au cours du temps. $c1$ et $c2$ sont des constantes contrôlant l'attrait des positions *optimum* pour une particule. Elles sont respectivement appelées composantes « cognitive » et « sociale ». Les valeurs $c1 = c2 = 2$ sont habituellement retenues. $r1$ et $r2$ sont des nombres aléatoires dans $[0, 1]$. $x_{i,j}$ (la position d'une particule) est généralement initialisée aléatoirement, alors que $v_{i,j}$ (sa vitesse) est initialisée à 0. Les *conditions d'arrêt* sont déterminées ici par un nombre donné d'itérations de l'algorithme.

```
procedure pso
    for i := 0 to n − 1 do
        initialize x_i and v_i
    initialize x̂_i and ĝ
    while terminationConditionsNotMet do
        for i := 0 to n − 1 do
            for j := 0 to m − 1 do
                v_{i,j} = ω ∗ v_{i,j} + c1 ∗ r1 ∗ (x̂_{i,j} − x_{i,j})+
                    c2 ∗ r2 ∗ (ĝ_j − x_{i,j})
                x_{i,j} = x_{i,j} + v_{i,j}
                if ( f(x_{i,j}) > f(x̂_{i,j}) ) then x̂_{i,j} = x_{i,j}
                if ( f(x_{i,j}) > f(ĝ_j) ) then ĝ_j = x_{i,j}
    return
```

Figure 5.6. *Un algorithme PSO*

5.3.2.2. *Utilisation de PSO pour le choix des paramètres ACO*

Nous avons appliqué PSO de manière directe pour trouver les meilleurs paramètres d'ACOC$_{graph}$. Chaque paramètre est une dimension du problème PSO, les domaines pouvant être discrets (booléens, entiers) ou continus (flottants). La fonction objectif est une combinaison de la qualité de la solution obtenue par ACOC$_{graph}$ avec ces paramètres et du temps de calcul de cette solution.

5.3.3. *Résultats expérimentaux et analyse*

Nous présentons des expérimentations sur des problèmes aléatoires et sur un *benchmark* de configuration de la littérature (problème des *racks*) qui est très structuré. Ces deux problèmes ont des propriétés fondamentalement différentes. Pour chaque type de problème, nous avons utilisé l'algorithme PSO, avec vingt particules, pour trouver des paramètres efficaces. Il va de soi que les points de convergence des jeux de

paramètres ainsi déterminés sont potentiellement spécifiques au problème. En appliquant l'approche à des problèmes de natures différentes, il devient donc possible de s'approcher de jeux de paramètres génériques. Dans les expérimentations qui suivent, *nbIte*, *nbAnts*, *NOBV* et toutes les valeurs maximum ont été fixées. *rho*, *evaS*, valeurs minimum et d'initialisation, *pMin* et *pMax* varient avec chaque particule.

5.3.3.1. *Problèmes aléatoires*

Nous avons généré des problèmes aléatoires prenant en compte : le nombre de types de composants, la densité des relations (probabilité d'avoir une relation entre deux classes), et la difficulté des relations (écart moyen entre cardinalités maximum et minimum). Les problèmes n'ont pas de contraintes additionnelles et sont créés de telle manière qu'une solution finie existe. Nous considérons donc ici un problème de satisfaction. Nous avons généré dix problèmes aléatoires hétérogènes, qui sont tous résolus 200 fois par une particule dans une position donnée. Nous montrons ici la moyenne des résultats obtenus sur ces 200 essais. Le tableau 5.1 montre les résultats de la meilleure particule après cinquante itérations de PSO.

paramètres d'$ACOC_{graph}$								
nbIte	nbAnts	ρ	evaS	pMin	pMax	noBV	objMin	objMax
200	30	4	restricted	0	9	100	2	90
attBV	attMin	attMax	relBV	relMin	relMax	classBV	classMin	classMax
n/a	n/a	n/a	100	12	90	n/a	n/a	n/a

résultats d'$ACOC_{graph}$			
% de solutions trouvées	nbiterations	taille	temps
100	11	13	89

Tableau 5.1. *$ACOC_{graph}$ sur les problèmes aléatoires, temps en millisecondes*

5.3.3.2. *Problème des* racks

Le problème des *racks* est un *benchmark* d'optimisation pour la configuration[1] dans lequel des cartes doivent être branchées à des *racks*. L'objectif est de minimiser le coût total (somme des prix des *racks*). Le *benchmark* propose quatre instances définissant chacune un nombre défini de cartes à brancher. De nouveau, chaque particule résout les quatre instances 200 fois avec les mêmes paramètres et nous retenons les résultats moyens. Le tableau 5.2 montre les résultats de la meilleure particule après cinquante itérations PSO. Notre approche est comparée à une méthode de recherche complète basée sur l'élimination des structures isomorphes [HEN 05], et aux résultats

1. http ://www.csplib.org, problème 31.

présentés dans [KIZ 01] (qui ont à notre connaissance obtenu les meilleurs résultats sur ce problème[2]).

paramètres d'$ACOC_{graph}$								
nbIte	nbAnts	ρ	evaS	pMin	pMax	objBV	objMin	objMax
500	30	2	restricted	0	11	100	0	90
attBV	attMin	attMax	relBV	relMin	relMax	classBV	classMin	classMax
100	5	90	100	13	90	100	8	90

instance		$ACOC_{graph}$						[HEN 05]	[KIZ 01]
	solution optimale	% de sol. trouvées	% de sol. optimales	prix moyen	nbitérations	taille	temps	temps	temps
1	550	100	88	632	79	21	2113	51	340
2	1100	100	37	1521	240	43	6753	36000	3700
3	1200	100	18	1886	301	56	10690	66	45000
4	1150	97	12	1643	249	33	4729	1800	/

Tableau 5.2. *$ACOC_{graph}$ sur le problème des racks, temps en millisecondes*

5.3.3.3. *Analyse expérimentale*

5.3.3.3.1. PSO et les paramètres ACO

Afin d'analyser les effets des nombreux paramètres de notre outil, nous avons utilisé PSO séparément sur chaque type de problème.

Le premier résultat important est que les meilleurs jeux de paramètres d'ACOC$_{graph}$ trouvés sont très proches pour les problèmes aléatoires et pour les *racks*. Ils sont pourtant de nature très différente, puisque les instances aléatoires sont des problèmes de satisfaction sur des structures aléatoires non bornées, alors que les *racks* sont des problèmes d'optimisation structurés sur un nombre borné de composants. Ces expérimentations pourraient donc suggérer que les paramètres trouvés sont peu dépendants du problème. Ceci constitue un élément important, puisque le calcul de paramètres optimaux est un long processus (environ vingt jours de calcul pour un jeu de problèmes).

Nous pouvons également observer que PSO converge vers des valeurs d'initialisation (RelBV, ObjBV, ClassBV, AttBV) proches du maximum. Le bénéfice de ce type d'initialisation avait déjà été souligné pour les algorithmes ACO existants.

2. Les deux méthodes exhaustives trouvent évidemment toujours la solution optimale.

Enfin, la nécessité d'une évaporation restreinte est également confirmée par ces expérimentations. Ce paramètre provient des propriétés originales de la configuration : tous les composants générés ne sont pas nécessairement utilisés dans une instance donnée. L'évaporation restreinte permet de « préserver » les phéromones propres à un composant, jusqu'à ce qu'il soit sélectionné à nouveau.

5.3.3.3.2. Résultats d'ACOC$_{graph}$

Les résultats sur les problèmes aléatoires montrent que l'approche peut résoudre efficacement des problèmes de satisfaction. Ces instances aléatoires ne sont pas faciles à résoudre. En effet, l'instance la plus difficile requiert la construction d'une structure contenant (au moins) environ 45 composants et n'offre que peu de solutions. Il faut en moyenne 47 itérations pour trouver une solution sur cette instance.

Les résultats sur les *racks* sont plus contrastés. D'un côté, l'approche ACO trouve des solutions correctes avec des temps de calcul acceptables. D'un autre côté, les solutions trouvées sont en moyenne relativement éloignées de l'*optimum*. La solution optimale n'est pas non plus trouvée suffisamment fréquemment pour une application réelle. Nous pouvons cependant noter que la meilleure solution est trouvée après un petit nombre d'itérations. L'incapacité actuelle d'améliorer cette solution par les itérations restantes semble donc indiquer un bon potentiel d'amélioration du comportement de l'algorithme.

Ces expérimentations sont encore préliminaires, mais peuvent néanmoins servir de « preuve de concept », car elles représentent les premières tentatives d'utilisation de méthodes stochastiques en configuration. De nombreuses perspectives sont envisagées afin d'améliorer l'efficacité de l'outil. Tout d'abord, comme dans toutes les approches ACO existantes, nous pensons que l'ajout d'heuristiques permettra d'améliorer nettement les résultats. De plus, certains paramètres ont été fixés dans ces expérimentations et des variantes efficaces connues de l'algorithme ACO doivent être essayées. Nous voulons également observer l'évolution des structures après que la meilleure solution ait été trouvée afin d'implanter, si cela se révèle pertinent, des périodes d'intensification. Le contrôle de la taille des solutions est un problème propre à la configuration. Nous pensons qu'il constitue un élément primordial pour l'application d'ACO à ce formalisme. Enfin, nous prévoyons de combiner ACO et recherche locale (combinaison appliquée avec succès aux CSP dans [SOL 02]).

5.4. Conclusion

Nous avons présenté un cadre théorique et les premiers résultats expérimentaux d'utilisation de l'optimisation par colonie de fourmis pour la recherche de modèles finis en configuration. A notre connaissance, l'utilisation de méthodes stochastiques n'avait pas encore été explorée dans ce domaine. De plus, les solutions apportées aux

problèmes posés par la logique du premier ordre dans les ACO peuvent être réutilisées en dehors de la configuration. Etant donné la difficulté des problèmes non bornés de configuration, les résultats sont encourageants. Parmi les nombreuses perspectives, nous prévoyons l'ajout d'heuristiques de choix de variables et de valeurs, l'implémentation de variantes de l'algorithme ACO, ou encore la combinaison d'ACO avec une recherche locale.

5.5. Bibliographie

[AMI 02] Amilhastre J., Fargier H., Marquis P., « Consistency restoration and explanations in dynamic CSPs Application to configuration », *Artificial Intelligence*, vol. 135, n° 1-2, p. 199–234, 2002.

[BRA 85] Brachman R. J., Schmolze J. G., « An Overview of the KL-ONE Knowledge Representation System », *Cognitive Science*, vol. 9, n° 2, p. 171–216, 1985.

[DOR 97] Dorigo M., Gambardella L. M., « Ant colony system : a cooperative learning approach to the traveling salesman problem », *IEEE Trans. Evolutionary Computation*, vol. 1, n° 1, p. 53–66, 1997.

[DOR 99] Dorigo M., Caro G. D., Gambardella L. M., « Ant Algorithms for Discrete Optimization », *Artificial Life*, vol. 5, n° 2, p. 137–172, 1999.

[DOR 05] Dorigo M., Blum C., « Ant colony optimization theory : A survey », *Theoretical Computer Science*, vol. 344, n° 2-3, p. 243–278, 2005.

[EBE 95] Eberhart R., Kennedy J., « A new optimizer using particle swarm theory », *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, p. 39–43, 1995.

[FLE 98] Fleischanderl G., Friedrich G., Haselböck A., Schreiner H., Stumptner M., « Configuring large-scale systems with generative constraint satisfaction », *IEEE Intelligent Systems - Special issue on Configuration*, vol. 13(7), 1998.

[GOG 03] Gogolla M., Bohling J., Richters M., « Validation of UML and OCL Models by Automatic Snapshot Generation », P. Stevens, J. Whittle, G. Booch (dir.), *UML*, vol. 2863 de *Lecture Notes in Computer Science*, p. 265–279, Springer-Verlag, Berlin, Heidelberg, 2003.

[GOG 07] Gogolla M., Büttner F., Richters M., « USE : A UML-based specification environment for validating UML and OCL », *Science of Computer Programming*, vol. 69, n° 1-3, p. 27–34, 2007.

[HEN 05] Henocque L., Kleiner M., Prcovic N., « Advances in Polytime Isomorph Elimination for Configuration », *Proceedings of Principles and Practice of Constraint Programming - CP 2005*, p. 301–313, Springer, Sitges Barcelone, Espagne, 2005.

[JUN 03] Junker U., Mailharro D., « The Logic of ILOG (J)Configurator : Combining Constraint Programming with a Description Logic », *Proceedings of Workshop on Configuration, IJCAI'03*, 2003.

[JUN 06] JUNKER U., « Configuration », F. Rossi, P. van Beek, T. Walsh (dir.), *Handbook of Constraint Programming*, chap. 13, Elsevier Science Inc., New York, 2006.

[KIZ 01] KIZILTAN Z., HNICH B., « Symmetry breaking in a rack configuration problem », *Proc. of the IJCAI'01 Workshop on Modelling and Solving Problems with Constraints, Seattle*, 2001.

[MCG 98] MCGUINNESS D. L., WRIGHT J. R., « Conceptual modelling for configuration : A description logic-based approach », *AI EDAM*, vol. 12, n° 4, p. 333–344, 1998.

[MIT 90] MITTAL S., FALKENHAINER B., « Dynamic Constraint Satisfaction Problems », *AAAI*, p. 25–32, 1990.

[SAB 96] SABIN D., FREUDER E., « Configuration as composite constraint satisfaction », *Artificial Intelligence and Manufacturing Research Planning Workshop*, p. 153–161, 1996.

[SOI 01] SOININEN T., NIEMELÄ I., TIIHONEN J., SULONEN R., « Representing Configuration Knowledge With Weight Constraint Rules », *Answer Set Programming*, 2001.

[SOL 02] SOLNON C., « Ants can solve constraint satisfaction problems », *IEEE Trans. Evolutionary Computation*, vol. 6, n° 4, p. 347–357, 2002.

[STU 93] STUMPTNER M., HASELBÖCK A., « A Generative Constraint Formalism for Configuration Problems », P. Torasso (dir.), *AI*IA*, vol. 728 de *Lecture Notes in Computer Science*, p. 302–313, Springer-Verlag, Berlin, Heidelberg, 1993.

[STU 97] STUMPTNER M., « An overview of knowledge-based configuration », *AI Communications*, vol. 10(2), p. 111–125, juin 1997.

[STU 00] STÜTZLE T., Hoos H. H., « MAX-MIN Ant System », *Future Generation Computer Systems*, vol. 16, n° 8, p. 889–914, 2000.

Chapitre 6

Colonies de fourmis pour le problème d'affectation d'unités : de l'optimisation à la commande prédictive

6.1. Introduction

L'optimisation à court terme (de quelques heures à quelques jours) des installations de production d'énergie apparaît désormais comme un enjeu industriel crucial, et ce pour plusieurs raisons. D'une part, l'ouverture des marchés de l'énergie conduit les producteurs à rationaliser leur production, afin de diminuer les coûts de production globaux. D'autre part, les moyens techniques sont plus performants, avec l'apparition d'installations de production aux rendements élevés, telles que les usines de cogénération (production simultanée d'énergies électrique et thermique) ou l'amélioration de la prédiction des demandes de consommateur. Enfin, la pression environnementale entraîne une prise de conscience accrue de la nécessaire diminution des émissions de polluants. C'est dans ce contexte pluridisciplinaire qu'une collaboration a été initiée entre le département automatique de SUPELEC d'une part et EDF recherche et développement d'autre part, dans le cadre d'un projet plus général de SUPELEC portant sur l'optimisation technico-économique des réseaux d'énergie.

Le problème d'optimisation d'un site de production est un problème d'optimisation mixte de très grande dimension, les variables binaires caractérisant l'état de marche/arrêt des installations et les variables réelles les quantités d'énergie produites.

Chapitre rédigé par Guillaume Sandou, Stéphane Font, Sihem Tebbani, Christian Mondon et Arnaud Hiret.

De nombreuses méthodes ont été testées afin de résoudre ce problème éminemment difficile et sont listées dans [SEN 98]. Les méthodes exactes, de type *Branch and Bound* [COH 83] ou *programmation dynamique* [HOB 88, RAV 94], souffrent rapidement de l'explosion combinatoire pour des problèmes de grandes dimensions. Pour remédier à ce problème, la *relaxation lagrangienne* est très largement utilisée [DOT 99, ROO 94]. Dans ce cas, des multiplicateurs de Lagrange sont introduits pour chaque contrainte couplant plusieurs unités. Le problème obtenu est séparable (un problème par unité), et les problèmes élémentaires, de petite dimension, peuvent être résolus de manière exacte par dualité. La non-convexité du problème entraîne néanmoins un *saut de dualité* et la nécessité d'introduire des mécanismes de restauration de la faisabilité. Afin de lutter contre l'explosion combinatoire, une autre voie est l'utilisation d'algorithmes de résolution approchés stochastiques, les métaheuristiques. Ainsi les *algorithmes génétiques* sont-ils utilisés dans [SAK 02] ou le *recuit simulé* dans [YIN 98]. Ces méthodes sont basées sur une alternance de phases d'exploration aléatoire de l'espace et de phases d'exploitation des résultats. Toute l'intelligence de l'algorithme repose dans la façon d'orienter le hasard vers les zones intéressantes de l'espace de recherche. Une des difficultés sous-jacentes à ce type de méthodes concerne la gestion des contraintes du problème : l'algorithme stochastique se déplaçant de manière aléatoire dans l'espace de recherche, aucune garantie ne peut être donnée quant à la faisabilité de la solution trouvée. Des mécanismes doivent alors être introduits pour gérer les contraintes (fonction de pénalités, « réparation » de solution pour rejoindre une solution faisable, déplacement dans le sous-espace faisable, etc.). Afin de satisfaire la faisabilité des solutions, un algorithme constructif a été choisi. Dans ce sens, l'algorithme de colonie de fourmis est apparu comme un bon candidat pour la résolution du problème d'affectation d'unités, permettant à la fois de lutter contre l'explosion combinatoire du problème, et d'autre part de gérer explicitement les contraintes de production.

Dans la section 6.2, nous rappelons la définition du problème d'*affectation d'unités*, ou *Unit Commitment* pour la terminologie anglaise. Ce problème est résolu par un algorithme de colonie de fourmis dans la section 6.3. Nous présentons tout d'abord la résolution du problème par un algorithme de colonie de fourmis discret, pour lequel le calcul des variables réelles est effectué de manière exacte, grâce à des hypothèses simplificatrices. Afin d'améliorer encore l'exploration de l'espace de recherche, nous présentons ensuite un couplage entre cet algorithme et un algorithme génétique. Enfin, nous relâchons les hypothèses simplificatrices et présentons la résolution du problème mixte, grâce à l'utilisation d'un algorithme de colonie de fourmis en variables réelles.

A l'issue de cette phase d'optimisation, la solution trouvée constitue une commande en boucle ouverte du système de production. Or, celui-ci est soumis à de nombreuses incertitudes, notamment sur la demande des consommateurs. Un pilotage en boucle fermée est donc nécessaire. Les principes de la commande prédictive permettent ici d'étendre les résultats d'optimisation dans un contexte de boucle fermée. Des résultats dans ce sens sont présentés dans la section 6.4.

6.2. Problème d'affectation d'unités, ou *Unit Commitment*

Le problème d'affectation d'unités consiste en la minimisation des coûts de production d'un ensemble d'unités satisfaisant une demande d'énergie d'un consommateur. Un tel système est représenté sur la figure 6.1.

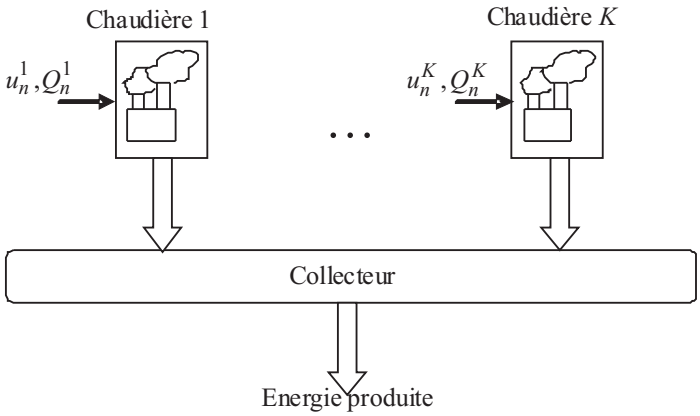


Figure 6.1. *Problème d'*Unit Commitment

Considérons ainsi un site de production constitué de K unités. L'horizon temporel est découpé en N périodes (typiquement une journée découpée en 24 tranches horaires). Notons :

– u_n^k l'état de marche de l'unité k durant la n^{ieme} période temporelle (1 pour allumée et 0 pour éteinte) ;

– Q_n^k la quantité d'énergie produite par l'unité k durant la n^{ieme} période temporelle ;

– $\mathrm{U}_n = \{u_n^k, k \in 1, \ldots, K\}$;

– $\mathrm{Q}_n = \{Q_n^k, k \in 1, \ldots, K\}$;

– $\mathbf{U} = \{\mathrm{U}_n, n \in 1, \ldots, N\}$;

– $\mathbf{Q} = \{\mathrm{Q}_n, n \in 1, \ldots, N\}$.

Le problème de minimisation des coûts d'exploitation s'écrit :

$$\min_{\mathbf{U},\mathbf{Q}} \sum_{n=1}^{N} \sum_{k=1}^{K} c_{prod}^k(u_n^k, Q_n^k) + c_{on/off}^k(u_n^k, u_{n-1}^k) \tag{6.1}$$

avec :

– $c_{prod}^k(u_n^k, Q_n^k)$ les coûts de production de l'unité k :

$$c_{prod}^k(u_n^k, Q_n^k) = (a_2^k(Q_n^k)^2 + a_1^k Q_n^k + a_0^k)u_n^k \tag{6.2}$$

– $c^k_{on/off}(u^k_n, u^k_{n-1})$ les coûts de démarrage et d'extinction de l'unité k :

$$c^k_{on/off}(u^k_n, u^k_{n-1}) = c^k_{on} u^k_n (1 - u^k_{n-1}) + c^k_{off} u^k_{n-1}(1 - u^k_n) \tag{6.3}$$

$a^k_2, a^k_1, a^k_0, c^k_{on}, c^k_{off}$ sont des données techniques du problème d'optimisation. Notons que ces valeurs sont difficiles à identifier et sont donc entachées d'incertitudes. Les résultats d'optimisation doivent donc être robustes vis-à-vis de ces incertitudes. Les contraintes à respecter sont :

– *les contraintes de capacités*. Une unité allumée ne peut produire qu'entre ses bornes techniques :

$$Q^k_{min} u^k_n \leq Q^k_n \leq Q^k_{max} u^k_n \tag{6.4}$$

– *les contraintes de satisfaction de la demande*. L'énergie totale produite doit satisfaire la demande des consommateurs :

$$\sum_{k=1}^{K} Q^k_n \geq Q^{dem}_n, \forall n \in 1, ..., N \tag{6.5}$$

– *les contraintes de temps minimum de marche*. Une unité k allumée ne peut être éteinte avant T^k_{up} unités temporelles :

$$\begin{array}{l}(u^k_{n-1} = 0, u^k_n = 1) \Rightarrow (u^k_{n+1} = 1, u^k_{n+2} = 1, ..., u^k_{n+T^k_{up}} = 1), \\ \forall n \in 1, ..., N, \forall k \in 1, ..., K\end{array} \tag{6.6}$$

– *les contraintes de temps minimum d'arrêt*. Une unité k éteinte ne peut être allumée avant T^k_{down} unités temporelles :

$$\begin{array}{l}(u^k_{n-1} = 1, u^k_n = 0) \Rightarrow (u^k_{n+1} = 0, u^k_{n+2} = 0, ..., u^k_{n+T^k_{down}} = 0), \\ \forall n \in 1, ..., N, \forall k \in 1, ..., K\end{array} \tag{6.7}$$

– *les contraintes de rampe*. La variation d'énergie produite par l'unité k est limitée :

$$|Q^k_{n+1} - Q^k_n| \leq \Delta Q^k, \forall n \in 1, ..., N, \forall k \in 1, ..., K \tag{6.8}$$

Les différents paramètres $Q^k_{min}, Q^k_{max}, T^k_{up}, T^k_{down}, \Delta Q^k$ sont également des données techniques. Q^{dem}_n est la demande d'énergie des consommateurs, qui doit être prédite.

6.3. Résolution du problème d'affectation d'unités par colonie de fourmis

Le problème d'affectation d'unités est un problème mixte de grande dimension fortement contraint pour lequel nous allons développer une résolution par colonie de fourmis. Rappelons que cet algorithme doit nous permettre, d'une part, de nous affranchir des problèmes d'explosion combinatoire, et d'autre part de gérer de façon explicite les nombreuses contraintes du problème.

6.3.1. *Résolution pour des coûts linéaires*

Dans un premier temps, nous considérons des coûts linéaires, c'est-à-dire que nous supposons $a_2^k = 0$ dans (6.2). Cette hypothèse permet de calculer les variables réelles de production Q_n^k à partir des variables binaires u_n^k.

6.3.1.1. *Calcul des variables réelles à partir des variables binaires*

Considérons un planning de production $u_n^k, \forall n \in 1, ...N, \forall k \in 1, ..., K$ faisable, c'est-à-dire que les contraintes de temps minimum de marche (6.6) et les contraintes de temps minimum d'arrêt (6.7) sont satisfaites. D'autre part, on suppose qu'il est possible de satisfaire les contraintes de demande des consommateurs (6.5), et ce en tenant compte des contraintes de rampe (6.8), soit :

$$\sum_{k=1}^{K} \left(\begin{array}{l} Q_{\min}^k (1 - u_n^k) + \\ \left(\min(Q_{\max}^k, Q_n^k + \Delta Q^k) \right) u_n^k \end{array} \right) u_{n+1}^k \geq Q_{n+1}^{dem}. \tag{6.9}$$

Cette équation signifie que si l'unité k est allumée à l'instant $n+1$, l'énergie maximale qu'elle peut produire est :

– $Q_{\min}^k$ si l'unité était éteinte à l'instant n ;

– $\min(Q_{\max}^k, Q_n^k + \Delta Q^k)$ si l'unité était déjà allumée à l'instant n.

Les variables réelles peuvent alors être calculées par le problème d'optimisation suivant :

$$\begin{aligned} &\arg\min_{\mathbf{Q}} \sum_{n=1}^{N} \sum_{k=1}^{K} c_{prod}^k(Q_n^k, u_n^k) + c_{on/off}^k(u_n^k, u_{n-1}^k) \\ &= \arg\min_{\mathbf{Q}} \sum_{n=1}^{N} \sum_{k=1}^{K} c_{prod}^k(Q_n^k, u_n^k) \\ &= \arg\min_{\mathbf{Q}} \sum_{n=1}^{N} \sum_{k=1}^{K} a_1^k Q_n^k u_n^k + a_0^k u_n^k \\ &= \arg\min_{\mathbf{Q}} \sum_{n=1}^{N} \sum_{k=1}^{K} a_1^k Q_n^k u_n^k \end{aligned} \tag{6.10}$$

Il est évident que l'optimum est de produire au maximum avec les installations les plus rentables. Sans perte de généralité, considérons que $a_1^1 \leq a_1^2 \leq ... \leq a_1^K$. La solution du problème d'optimisation (6.10) est donc donnée par les équations suivantes :

$$Q_n^k = \left\{ \begin{array}{ll} \tilde{Q}_n^k & \text{si } Q_{\min}^k \leq \tilde{Q}_n^k \leq Q_{\max}^k(n) \\ Q_{\min}^k & \text{si } \tilde{Q}_n^k < Q_{\min}^k \\ Q_{\max}^k & \text{si } \tilde{Q}_n^k > Q_{\max}^k \end{array} \right. \tag{6.11}$$

avec :

$$\tilde{Q}_n^k = Q_n^{dem} - \sum_{i=1}^{k-1} Q_n^i - \sum_{i=k+1}^{K} Q_{\min}^i u_n^i \tag{6.12}$$

La quantité $\tilde{Q}_n^k$ représente la quantité réelle produite par l'unité k. Elle comporte plusieurs termes :

– le terme Q_n^{dem} est la demande d'énergie des consommateurs ;

– le terme $\sum_{i=1}^{k-1} Q_n^i$ représente la quantité produite par les autres unités les plus rentables ;

– le terme $\sum_{i=k+1}^{K} Q_{\min}^i u_n^i$ représente les énergies minimales que doivent produire les unités restantes.

La quantité maximale d'énergie $Q_{\max}^k(n+1)$ que peut produire une unité k à l'instant $n+1$ est égale :

– à la valeur minimale $Q_{\min}^k$ si l'unité était éteinte au temps n ;

– à $Q_n^k + \Delta Q^k$ sinon, pour respecter les contraintes de rampe. Cette valeur doit être limitée par la quantité d'énergie maximale $Q_{\max}^k$ que peut produire l'unité.

Finalement, $Q_{\max}^k(n+1)$ est donnée par :

$$Q_{\max}^k(n+1) = Q_{\min}^k(1 - u_n^k) + \min(Q_{\max}^k, Q_n^k + \Delta Q^k)u_n^k \tag{6.13}$$

6.3.1.2. *Reformulation du problème d'*Unit Commitment *en parcours de graphe*

Le problème d'affectation d'unités peut se reformuler sous la forme d'un parcours de graphe, conformément à la figure 6.2.

Le graphe a une forme matricielle, avec en abscisse le temps et en ordonnée les états possibles du système de production. Les coûts de production sont associés à chaque nœud, tandis que les coûts de démarrage et d'arrêt sont associés aux arcs du graphe. Le but est d'aller de l'un des états possibles au temps 1, jusqu'à l'un des états possibles au temps N, tout en minimisant les coûts d'exploitation. Le problème traité ici est donc très proche du classique problème du voyageur de commerce. Les fourmis vont parcourir ce graphe, avec une loi de probabilité semblable à celle qui a été définie dans les chapitres précédents. Lorsqu'une fourmi f, lors de l'itération t de l'algorithme, est arrivée dans l'état i $\mathrm{U}_n = (u_n^1, u_n^2, ..., u_n^K)^T$ à l'instant n, la probabilité qu'elle choisisse l'état j $\mathrm{U}_{n+1} = (u_{n+1}^1, u_{n+1}^2, ..., u_{n+1}^K)^T$ à l'instant $n+1$ est donnée par la relation suivante :

$$p_t^f(i \to j) = \frac{\eta_{ij}^\alpha \tau_{ij}^\beta(t)}{\sum_{k \in J_t^f(i)} \eta_{ik}^\alpha \tau_{ik}^\beta(t)} \tag{6.14}$$

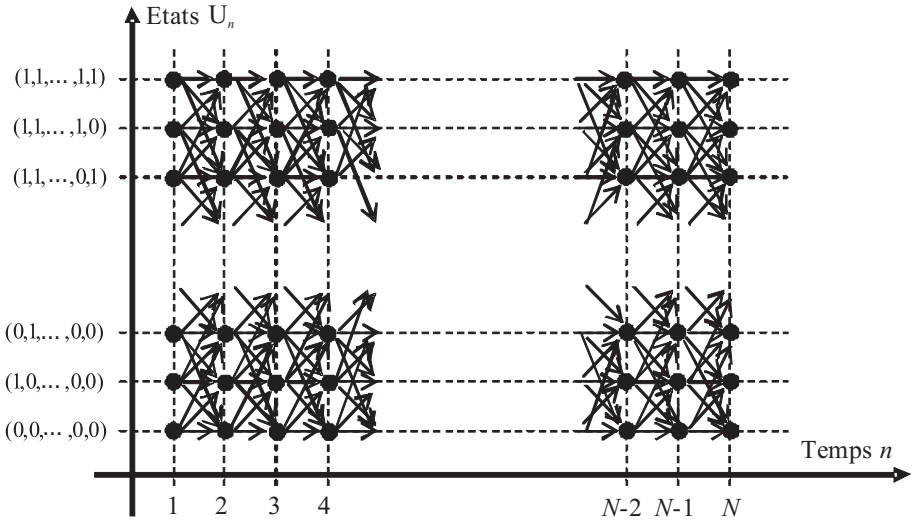


Figure 6.2. *Reformulation de l'*Unit Commitment *sous forme de parcours de graphe*

Les ensembles $J_t^f(i)$ permettent de gérer explicitement toutes les contraintes : ils contiennent tous les états possibles, c'est-à-dire ceux qui permettent de satisfaire la demande des consommateurs (6.9), ainsi que les contraintes de temps minimum de marche (6.6) et de temps minimum d'arrêt (6.7).

Dans la loi de propagation (6.14), la phéromone $\tau_{ij}(t)$ a la signification habituelle. Nous avons opté ici pour un algorithme de type *MAX-MIN Ant System* [STü 00] pour son évolution au fur et à mesure du déroulement de l'algorithme. Le paramètre d'attractivité η_{ij} doit correspondre à la physique du système. Ainsi, la décision d'allumage ou d'extinction des unités dépend essentiellement de deux points :

– les coûts d'exploitation (les unités allumées sont-elles les plus rentables ?) ;

– l'adéquation entre la production maximale possible et la demande (il est probablement sous-optimal d'avoir beaucoup d'unités allumées, si la demande des consommateurs est faible).

Le premier point est indirectement géré par la phéromone, qui va chercher à influencer la recherche vers les états les plus rentables. Nous choisissons de gérer le deuxième point avec l'attractivité. Pour ce faire, définissons la variable λ décrivant l'adéquation d'un état avec la demande :

$$\lambda(\mathrm{U}_n) = \sum_{k=1}^{K} Q_{max}^k u_n^k - Q_n^{dem} \tag{6.15}$$

Dès lors, l'attractivité peut être définie à l'aide de la courbe de la figure 6.3.

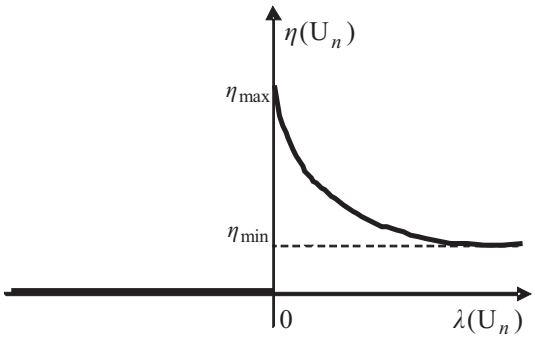


Figure 6.3. *Définition de l'attractivité pour le problème d'affectation d'unités*

Notons que l'attractivité dans ce cas est affectée aux nœuds du graphe, et non aux transitions. Ce choix pourrait être remis en cause pour d'autres versions de l'algorithme.

6.3.1.3. *Résultats de validation*

L'algorithme proposé a été testé à l'aide de Matlab 6.5 [SAN 04]. Des tests ont tout d'abord été effectués pour un système de production constitué de quatre unités sur un horizon $N = 12$ heures. Pour de relatives petites dimensions (48 variables binaires ici), une résolution exacte par une méthode de *Branch and Bound* reste possible, conduisant à un coût de 5 363 euros pour une courbe de demande caractéristique. Les paramètres utilisés sont les suivants : taille de la population = 20, $\alpha = 2, \beta = 1$. Les résultats d'optimisation correspondants sont présentés sur le tableau 6.1. Le temps de calcul par itération est de 0,5 seconde sur un Pentium IV, 2,0 GHz. Du fait du caractère stochastique de l'algorithme, les résultats présentés sont des tests statistiques obtenus en exécutant cent fois l'algorithme sur la même instance. Ainsi, les résultats obtenus sont très satisfaisants. On observe notamment que de très bonnes solutions sont trouvées, et ce, même pour un nombre d'itérations relativement faible.

Nb d'itérations	Pire cas	Meilleur cas	Moyenne	Ecart-type
30	6 077(+13 %)	5 363(+0 %)	5 690(+6 %)	175
50	6 078(+13 %)	5 363(+0 %)	5 607(+4, 5 %)	138
100	5 976(+11, 5 %)	5 363(+0 %)	5 537(+3, 2 %)	146
200	5 654(+5, 4 %)	5 363(+0 %)	5 439(+1, 4 %)	120

Tableau 6.1. *Résultats pour le cas à quatre unités*

Un cas plus réaliste de huit installations pour un horizon temporel de $N = 24$ heures a été testé pour des cas particuliers où la solution optimale est connue à l'avance. Dans ces cas, il apparaît que très rapidement (moins de cent itérations pour des populations de vingt fourmis) l'algorithme utilisé permet de trouver des solutions de qualité intéressante (écart de 10 % par rapport à l'optimum). Il est apparu néanmoins qu'avec les réglages proposés, il était difficile de trouver de meilleures solutions, l'algorithme ayant tendance à rester piéger dans des *minima* locaux. De meilleurs réglages auraient sans doute permis d'améliorer encore la qualité des solutions. Cependant, dans un contexte industriel, il est indispensable que le réglage des paramètres d'un algorithme soit simple et immédiat, et que l'utilisation de l'algorithme soit robuste vis-à-vis des paramètres choisis. Nous nous sommes donc orientés vers une autre approche, consistant à hybrider l'algorithme proposé avec un algorithme génétique.

6.3.2. *Couplage avec un algorithme génétique*

6.3.2.1. *Principe du couplage*

L'idée de ce couplage entre un algorithme de colonie de fourmis et un algorithme génétique repose sur les remarques suivantes :

– l'algorithme de colonie de fourmis permet de trouver très rapidement des solutions faisables, gère explicitement les contraintes du problème, mais converge vers des *minima* locaux, s'il n'est pas réglé de manière fine ;

– l'algorithme génétique est très robuste vis-à-vis des valeurs de ses paramètres, mais nécessite une gestion des contraintes (fonctions de pénalité, etc.).

Nous allons donc utiliser l'algorithme de colonie de fourmis comme un générateur de solutions faisables, qui va venir alimenter l'algorithme génétique. Une telle approche a été décrite dans [SAN 08].

La stratégie d'optimisation est décrite par la figure 6.4.

Plus précisément, une première optimisation est réalisée par colonie de fourmis. Toutes les solutions faisables calculées dans cette optimisation sont conservées pour créer la population initiale de l'algorithme génétique. Pour les deux algorithmes, nous considérons toujours des coûts de production linéaires, de sorte que le calcul des variables réelles puisse se faire *via* l'algorithme récursif (6.11).

6.3.2.2. *Création de la population initiale par colonie de fourmis*

Le but de l'optimisation par colonie de fourmis est donc d'échantillonner l'espace faisable. Pour ce faire, l'attractivité définie à la figure 6.3 n'est pas conservée : elle est fixée constante et égale à 1 pour tous les états, afin de ne privilégier aucune solution et ainsi obtenir un échantillonnage le plus uniforme possible. D'un point de vue industriel, ce choix est très intéressant, puisqu'aucun réglage de paramètre n'est requis pour l'attractivité.

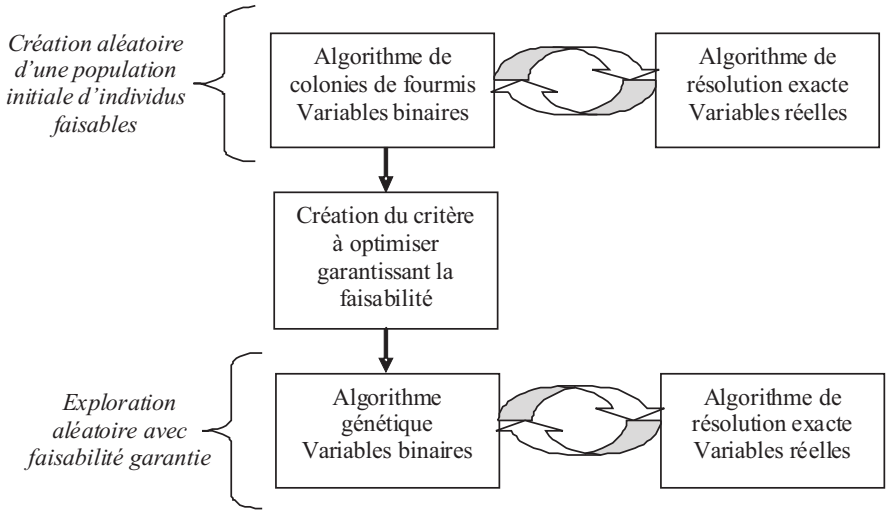


Figure 6.4. *Couplage d'un algorithme de colonie de fourmis avec un algorithme génétique*

6.3.2.3. *Création d'un critère garantissant la faisabilité*

A l'issue de l'optimisation par colonie de fourmis, il est possible de définir un critère à optimiser qui permette de garantir la faisabilité de la solution trouvée par l'algorithme génétique. Définissons ainsi :

$$\min_{\mathbf{U},\mathbf{Q}} \left\{ \sum_{n=1}^{N} \sum_{k=1}^{K} c^k_{prod}(u^k_n, Q^k_n) + c^k_{on/off}(u^k_n, u^k_{n-1}) + ((1+\varepsilon)c^f + h(\mathbf{U},\mathbf{Q})) \cdot B(\mathbf{U},\mathbf{Q}) \right\} \tag{6.16}$$

avec :

– c^f le coût de la meilleure solution trouvée par la colonie de fourmis ;

– ε un petit réel positif ;

– $B(\mathbf{U},\mathbf{Q})$ une fonction booléenne valant 1 pour les solutions non faisables ;

– $h(\mathbf{U},\mathbf{Q})$ une fonction de pénalité pour les solutions non faisables.

Avec ce critère, toute solution non faisable a un coût supérieur à la meilleure solution (faisable) trouvée par la colonie de fourmis. Ce critère peut donc être optimisé par un algorithme génétique élitiste (la meilleure solution est conservée à chaque itération), les contraintes étant implicitement prises en compte.

6.3.2.4. *Résolution par algorithme génétique*

L'algorithme génétique est une métaheuristique très connue, basée sur la théorie de l'évolution de C. Darwin. Le principe général de cet algorithme est rappelé sur la figure 6.5.

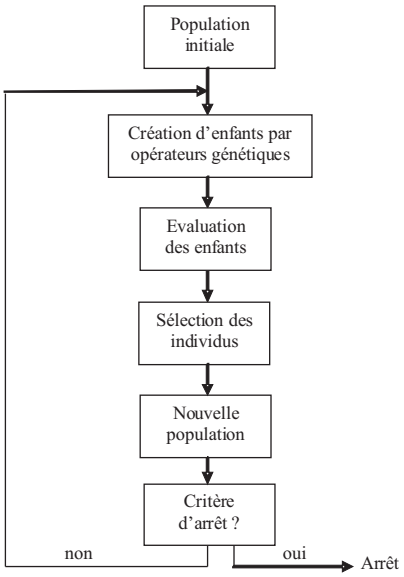


Figure 6.5. *Description générale d'un algorithme génétique*

Les opérateurs génétiques classiques de croisement et de mutation, appliqués au problème d'*Unit Commitment* sont également rappelés sur la figure 6.6, où les individus ont été représentés sous forme matricielle, pour plus de lisibilité.

Il apparaît rapidement que ces opérateurs ne sont pas parfaitement adaptés à la résolution du problème d'affectation d'unité. En effet, la mutation, telle que décrite dans la figure 6.6, conduit très souvent à des individus infaisables, du fait des contraintes de temps minimum de marche et de temps minimum d'arrêt (6.6), (6.7). Dès lors, nous avons introduit un opérateur de *mutation sélective*, décrit sur la figure 6.7.

L'idée est de détecter les instants de commutation d'une solution potentielle, et de n'autoriser les mutations que pour ces variables particulières. L'opérateur ainsi défini permet d'augmenter la probabilité de création de solutions faisables. Dans le même état d'esprit, des opérateurs *tout allumé* et *tout éteint* ont été définis, permettant d'allumer ou d'éteindre une installation de production sur une période plus ou moins longue. Une illustration de ces opérateurs est donnée sur la figure 6.8.

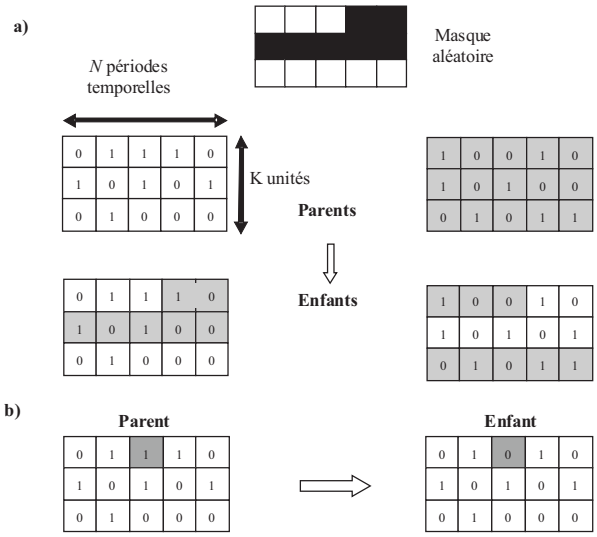


Figure 6.6. *Opérateurs génétiques classiques a) opérateur de croisement ; b) opérateur de mutation*

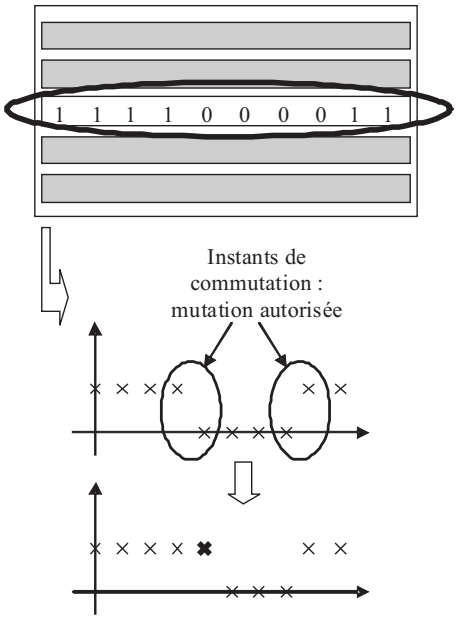


Figure 6.7. *Opérateur de mutation sélective*

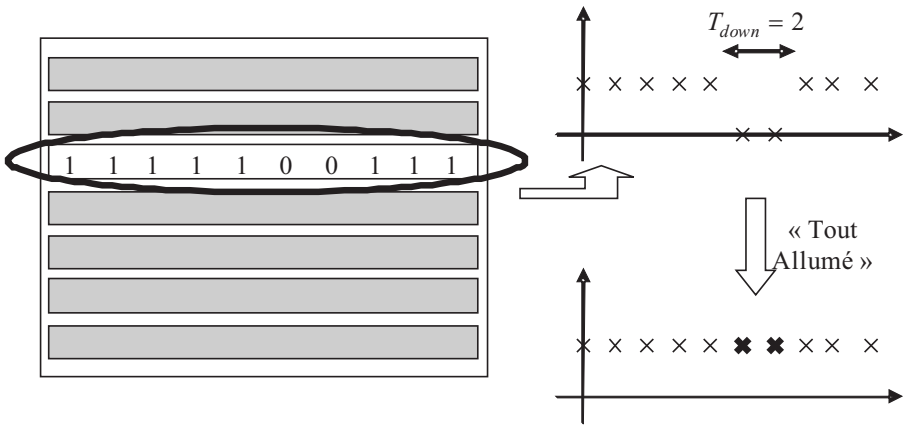


Figure 6.8. *Opérateur « tout allumé »*

Sur cette figure, le planning proposé conduirait nécessairement à des plannings infaisables par application des opérateurs de mutation ou de mutation sélective. Dès lors, l'opérateur *tout allumé* permet d'augmenter la probabilité de passer d'une solution faisable à une autre, en traversant l'espace non faisable.

6.3.2.5. *Résultats de simulation*

Nous avons testé l'algorithme proposé pour un cas académique de quatre unités, pour un horizon de $N = 24$ heures. L'apport de l'initialisation de la population par algorithme génétique a été étudié en résolvant le problème avec un algorithme génétique *pur*, pour lequel le critère à optimiser est obtenu à partir d'une solution faisable calculée à partir d'une liste de priorité, le reste de la population initiale étant complètement aléatoire. Les résultats sont donnés sur le tableau 6.2. Comme précédemment, des résultats statistiques sont donnés, l'algorithme ayant été exécuté cent fois sur la même instance. La colonne *Taux de succès* indique le pourcentage de fois où la solution globale a été trouvée.

Essai	Pire cas	Moyenne	Taux de succès	Temps de calcul
Fourmis+AG (100 iter)	+9,4 %	+2,5 %	30 %	15s
AG (100 iter)	+13,4 %	+3,1 %	20 %	13s
Fourmis+AG (200 iter)	+3,5 %	+0,4 %	80 %	25s
AG (200 iter)	+4,5 %	+0,5 %	77 %	24s

Tableau 6.2. *Résultats d'optimisation pour l'algorithme couplé*

La taille des populations traitées par l'algorithme génétique est de cinquante. Lorsque l'initialisation est faite par colonie de fourmis, dix générations de cinq fourmis

sont utilisées. Les paramètres de l'algorithme de colonie de fourmis sont identiques à ceux utilisés dans la section précédente (exception faite de l'attractivité qui est constante et égale à un pour tous les chemins possibles). En ce qui concerne l'algorithme génétique, une probabilité de 70 % est utilisée pour le croisement et de 10 % pour la mutation, la mutation sélective et les opérateurs *tout allumé* et *tout éteint*.

Pour un faible nombre d'itérations (cent), on observe que l'initialisation de l'algorithme génétique par une population faisable calculée par colonie de fourmis permet d'augmenter grandement la qualité des solutions, le taux de succès passant de 20 à 30 %. Pour un plus grand nombre d'itérations, l'amélioration de la résolution du problème ne se voit plus au niveau de la qualité de la solution trouvée (les résultats sont tout à fait comparables), mais au niveau des temps de calcul. En effet, dans le cas où la solution globale est trouvée, le nombre moyen d'itérations nécessaires est de 96 pour l'algorithme couplé, et de 126 pour l'algorithme génétique seul. Ainsi, l'utilisation d'un algorithme de colonie de fourmis comme générateur de solutions faisables peut s'avérer cruciale pour diminuer les temps de calcul lors d'une implantation temps réel. D'autre part, l'algorithme hybride proposé est beaucoup plus robuste vis-à-vis des variations des paramètres d'optimisation.

6.3.3. *Extension pour les problèmes en variables réelles*

6.3.3.1. *Description de l'algorithme*

Les résultats obtenus précédemment sont basés sur l'hypothèse simplificatrice de coûts linéaires ($a_2^k = 0$). Cette hypothèse permet en effet de calculer toutes les variables réelles à partir des variables binaires. Nous allons maintenant relâcher cette hypothèse et calculer les variables réelles de production avec un algorithme de colonie de fourmis en variables réelles [SER 07]. Les variables binaires seront toujours calculées par l'algorithme hybride colonie de fourmis/algorithme génétique, décrit dans la section précédente. Ainsi, pour chaque solution potentielle $\mathbf{U}$, des variables $\mathbf{Q}$ doivent être calculées.

Nous utiliserons ici un algorithme développé par K. Socha et M. Dorigo [SOC 08]. Dans cet algorithme, s solutions potentielles sont conservées en mémoire. Pour alléger les notations, définissons :

– $\mathbf{U}_j = \{u_{n,j}^k; n = 1, \ldots, N; k = 1, \ldots, K\}$ la j^{ieme} solution en variables binaires conservée ;

– $\mathbf{Q}_j = \{Q_{n,j}^k; n = 1, \ldots, N; k = 1, \ldots, K\} = \{x_j^1, \ldots, x_j^{KN}\}$ la j^{ieme} solution en variables réelles conservée.

Nous conservons alors en mémoire une matrice T telle que :

$$\mathbf{T} = \begin{bmatrix} x_1^1 & x_1^2 & \dots & x_1^i & \dots & x_1^{KN} \\ x_2^1 & x_2^2 & \dots & x_2^i & \dots & x_2^{KN} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_j^1 & x_j^2 & \dots & x_j^i & \dots & x_j^{KN} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_s^1 & x_s^2 & \dots & x_s^i & \dots & x_s^{KN} \end{bmatrix} \tag{6.17}$$

Le vecteur des coûts correspondants est également conservé en mémoire :

$$\begin{aligned} &\mathbf{H} = [f_1, f_2, \dots, f_s]^T \\ &\text{avec : } f_j = f(\mathbf{U}_j, \mathbf{Q}_j) = \sum_{n=1}^{N} \begin{pmatrix} \sum_{k=1}^{K} c_{prod}^k(u_{n,j}^k, Q_{n,j}^k) \\ +c_{on/off}^k(u_{n,j}^k, u_{n-1,j}^k) \end{pmatrix} \end{aligned} \tag{6.18}$$

A partir de ces coûts, supposés ordonnés ($f_1 \leq f_2 \leq \dots \leq f_s$), on définit une densité de probabilité discrète p_r par :

$$p_r = \frac{\omega_r}{\sum_{j=1}^{s} \omega_j} \quad \text{et} \quad \omega_j = \frac{1}{qs\sqrt{2\pi}} exp^{-\frac{(j-1)^2}{2q^2s^2}} \tag{6.19}$$

et q un paramètre de l'algorithme. Dès lors, l'algorithme de calcul des variables réelles est le suivant :

– étape 1 : choix d'une fourmi modèle, notée l, selon la loi de probabilité de l'équation (6.19) ;

– étape 2 : choix des nouvelles variables réelles $\{x_{new}^i, i = 1, \dots, KN\}$, tirées selon des lois de probabilité gaussiennes $N(\mu_{new}^i, \sigma_{new}^i)$ définies par :

$$\begin{aligned} \mu_{new}^i &= x_l^i \\ \sigma_{new}^i &= \frac{\xi}{s-1} \sum_{m=1}^{s} |x_m^i - x_l^i| \end{aligned} \tag{6.20}$$

ξ est un autre paramètre de l'algorithme. A l'issue de cette étape, il est possible que la quantité d'énergie produite ne respecte pas la contrainte de satisfaction de la demande (6.5). Dans ce cas, la procédure suivante est utilisée :

– choix des variables réelles selon l'algorithme précédent (étapes 1 et 2) ;

– étape 3 : si $\sum_{k=1}^{K=1} Q_n^k u_n^k < Q_n^{dem}$ (resp. >), alors choisir aléatoirement l'une des unités allumées et augmenter (resp. diminuer) la production correspondante, jusqu'à obtenir l'égalité ;

– étape 4 : si cela ne suffit pas, recommencer l'étape 3 avec une autre unité.

Essai	Meilleur	Moyenne	Taux de succès	Temps de calcul
100 iter	+0 %	+7, 6 %	12 %	22s
200 iter	+0 %	+2, 6 %	46 %	45s
500 iter	+0 %	+1, 6 %	65 %	25s

Tableau 6.3. *Résultats d'optimisation pour l'algorithme mixte (quatre unités)*

Essai	Meilleur	Moyenne	Temps de calcul
500 iter	+1, 4 %	+9, 7 %	275s
1000 iter	+0, 2 %	+7, 8 %	550s

Tableau 6.4. *Résultats d'optimisation pour l'algorithme mixte (dix unités)*

A l'issue de ces quatre étapes, de nouvelles solutions potentielles sont donc créées et évaluées. Une étape d'*évaporation* est nécessaire, afin d'éviter la convergence trop rapide vers des *minima* locaux. Les meilleures nouvelles solutions vont ainsi intégrer la matrice T en remplacement des solutions les plus mauvaises de cette matrice. Cet algorithme de colonie de fourmis en variables réelles, qui s'apparente à un algorithme d'estimation de densité, est ainsi basé sur les deux principes de l'algorithme discret : influence des résultats précédents et oubli des erreurs passées.

6.3.3.2. *Résultats de simulation*

L'algorithme de colonie mixte a été testé pour une instance semblable à celle de la section précédente, pour un cas à quatre unités et un horizon de $N = 24$ heures. Les paramètres de l'algorithme discret sont les mêmes que précédemment. Les paramètres de l'algorithme en variables réelles sont $s = 20, q = 1, \xi = 0, 8$. Les résultats sont présentés dans le tableau 6.3.

Des résultats pour un cas plus réaliste à dix unités sont présentés dans le tableau 6.4.

Les résultats présentés ici montrent ainsi l'intérêt de l'algorithme de colonie de fourmis, dans sa version discrète et/ou mixte pour la résolution de ce problème industriel.

6.4. De l'optimisation à la commande prédictive

6.4.1. *Problématique*

Les résultats d'optimisation obtenus dans les paragraphes précédents supposent que la demande des consommateurs Q_n^{dem} est parfaitement connue sur l'horizon temporel. En réalité, cette demande est prédite, et l'algorithme ne connaît que cette prédiction $\hat{Q}_n^{dem}$, nécessairement entachée d'erreurs de prédiction. Dès lors, les résultats

d'optimisation constituent une commande en boucle ouverte du système, calculée sur un modèle perturbé. Son application directe sur le vrai système peut donc poser de graves problèmes, et un pilotage en boucle fermée est nécessaire. Dans cette étude, nous avons pris le parti d'utiliser les principes de la commande prédictive, qui sont une façon naturelle d'étendre des résultats d'optimisation dans un cadre de boucle fermée [SAN 07].

6.4.2. *Commande prédictive*

La commande prédictive est une méthode de commande éprouvée des systèmes industriels. Le principe de la méthode est représenté sur la figure 6.9.

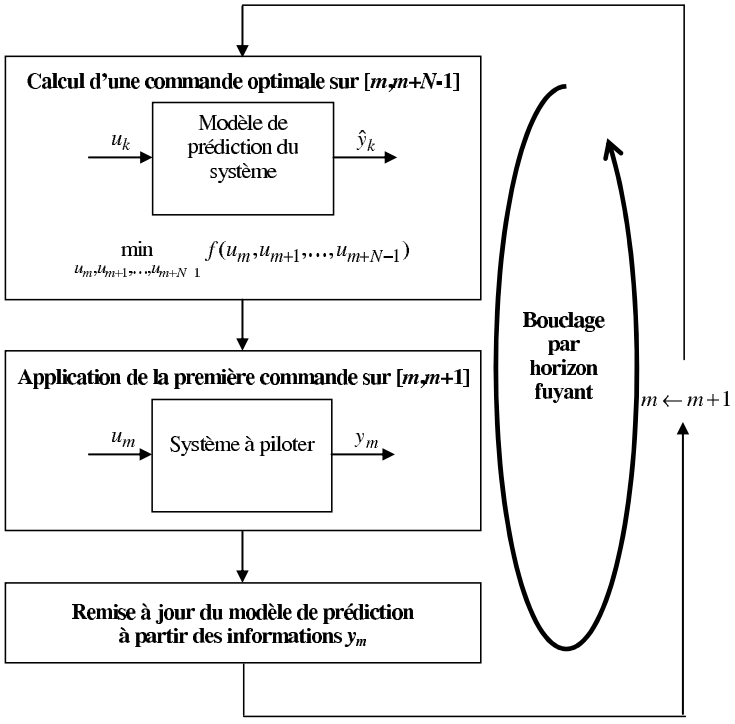


Figure 6.9. *Principe de la commande prédictive*

La commande prédictive suppose que l'on dispose d'un modèle de prédiction du système à piloter, ainsi que des objectifs à atteindre sur un horizon temporel futur $[m, m + N - 1]$. A partir de ce modèle de prédiction, une commande optimale est calculée. Seule la première valeur de la commande, u_m, est appliquée au système. Du fait des erreurs dans le modèle de prédiction, un écart apparaît entre la réponse du

système et la prédiction. Cet écart est utilisé pour remettre à jour le modèle de prédiction. La procédure est alors répétée sur l'horizon temporel $[m+1, m+N]$. Ce principe de bouclage est appelé *horizon fuyant*, sur lequel reposent toutes les techniques de commande prédictive. Le deuxième ingrédient fondamental est la résolution en ligne de problèmes d'optimisation successifs. La description fine des différentes techniques de commande prédictive sort largement du cadre de ce chapitre ; le lecteur intéressé pourra se reporter à [CLA 87], ou à [MAC 02].

6.4.3. *Application au problème d'*Unit Commitment

Dans le cas du problème d'*Unit Commitment*, le modèle de prédiction s'appuie sur la prédiction de la demande des consommateurs $\hat{Q}_n^{dem}$ sur l'horizon $[m, m+N-1]$. Les algorithmes basés sur l'optimisation par colonie de fourmis permettent alors de définir la politique optimale à adopter pour le système, c'est-à-dire les variables $\{u_n^k, Q_n^k; n = m, \ldots, m+N-1; k = 1, \ldots, K\}$) qui permettraient d'obtenir les coûts les plus bas, si la demande du consommateur était effectivement donnée par $\hat{Q}_n^{dem}$. A l'instant m, on peut donc appliquer les commandes $\{u_m^k, Q_m^k; k = 1, \ldots, K\}$ au système de production, sur lequel agit naturellement la vraie demande des consommateurs. Notons qu'à ce stade, Q_m^{dem} est connue : on peut donc retoucher légèrement les variables $\{Q_m^k; k = 1, \ldots, K\}$, sous réserve de respecter les contraintes de rampe.

A l'instant $m+1$, le modèle de prédiction est remis à jour avec les niveaux de production effectivement utilisés à l'instant m, et l'optimisation est recalculée sur l'horizon $[m+1, m+N]$ à partir des demandes prédites.

6.4.4. *Résultats de simulation*

L'algorithme de commande prédictive a été testé sur le système à quatre unités de production, avec une prédiction de la demande toujours sous-estimée. Ceci constitue le cas le plus défavorable au pilotage du système. Les résultats de simulation obtenus pour un scénario de quatre jours sont présentés sur la figure 6.10.

Sans le bouclage par horizon fuyant, la production aurait toujours été égale à la prédiction. L'introduction du bouclage permet de limiter grandement l'influence de l'erreur de prédiction.

6.4.5. *Intérêt de l'optimisation par colonie de fourmis pour la commande prédictive*

La commande prédictive est une commande très utilisée pour le pilotage des systèmes industriels, notamment pour les systèmes dont la trajectoire de référence est connue à l'avance. Comme expliqué précédemment, l'application de ce type de méthode de commande repose sur la résolution en ligne de problèmes d'optimisation.

Dans ce cadre, l'utilisation d'algorithmes stochastiques, tels que l'optimisation par colonie de fourmis, peut s'avérer très intéressante pour le pilotage de systèmes dynamiques hybrides et non linéaires. En effet, pour ce type de systèmes, les problèmes d'optimisation sont constitués d'un grand nombre de variables binaires et présentent fréquemment de nombreux *minima* locaux. D'autre part, il arrive souvent que les fonctions de coût et de contrainte ne puissent être calculées que par simulateur, rendant délicate l'utilisation d'algorithmes de type descente. Ainsi, un algorithme de colonie de fourmis est un bon candidat pour la résolution de ce type de problèmes.

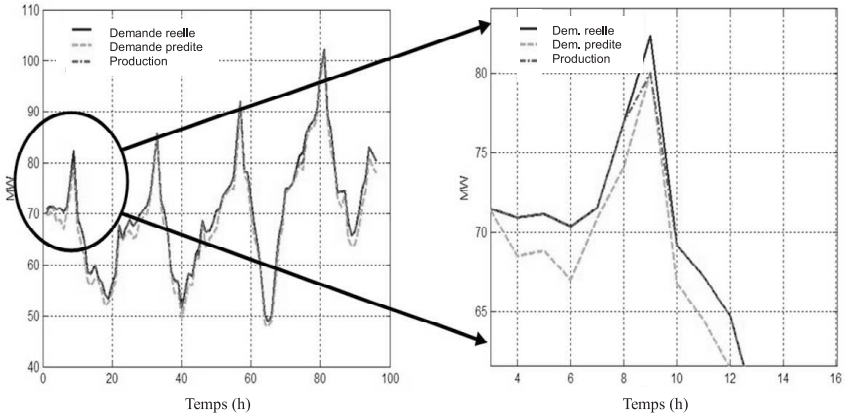


Figure 6.10. *Application de la commande prédictive*

Cependant, pour que cette méthode soit applicable, la solution du problème d'optimisation doit être obtenue en ligne, ce qui suppose que la période d'échantillonnage utilisée soit grande (les temps d'exécution de l'algorithme, quoique remarquables comparativement à la difficulté du problème résolu, restent élevés). Cela est possible si les dynamiques du système sont lentes (quelques heures pour le problème d'*Unit Commitment*, ce qui autorise une période d'échantillonnage d'une heure). En outre, des résultats de stabilité de la boucle fermée obtenue par commande prédictive ont pu être établis. Ces résultats supposent que la solution globale du problème d'optimisation est trouvée à chaque pas d'échantillonnage. Aucune garantie ne peut être donnée avec les algorithmes stochastiques, quant à l'optimalité de la solution calculée. C'est pourquoi l'utilisation d'un algorithme stochastique, tel que la colonie de fourmis, pour le calcul de commande ne peut généralement être valide que pour des systèmes stables en boucle ouverte.

En résumé, l'algorithme de colonie de fourmis est un outil très intéressant pour la commande en boucle fermée de systèmes :

– comprenant de nombreuses variables binaires ;

– ayant des expressions des coûts et des contraintes non linéaires, voire non analytiques ;

– présentant des dynamiques lentes ;

– stables en boucle ouverte.

C'est en particulier le cas du problème d'*Unit Commitment* traité dans ce chapitre.

6.5. Conclusion

Dans ce chapitre, nous avons présenté l'application de l'optimisation par colonie de fourmis au problème d'affectation d'unités. Ce type d'algorithme a ainsi permis de résoudre un problème d'optimisation mixte de relative grande dimension en un temps raisonnable. Plusieurs variantes ont été exposées : algorithme en variables binaires, couplage avec un algorithme génétique, extension aux variables réelles.

Les résultats d'optimisation obtenus constituent une commande en boucle ouverte du système de production. Cependant, cette commande ne peut être calculée qu'à partir de prédictions de la demande des consommateurs, nécessairement entachées d'erreurs. C'est pourquoi un pilotage du système en boucle fermée est indispensable. Le principe de l'horizon fuyant, utilisé dans les techniques de commande prédictive, est un moyen naturel d'étendre les résultats d'optimisation dans un cadre de boucle fermée. L'utilisation de l'optimisation par colonie de fourmis peut ainsi permettre de piloter des systèmes industriels hybrides et non linéaires.

6.6. Bibliographie

[CLA 87] Clarke D. W., Mohtadi C., Tuffs P. S., « Generalized predictive control - Part I. The basic algorithm », *Automatica*, vol. 23, n° 2, p. 137–148, 1987.

[COH 83] Cohen A. I., Yoshimura M., « A branch-and-bound algorithm for unit commitment », *IEEE Transactions on Power Apparatus and Systems*, vol. 102, n° 2, p. 444–451, 1983.

[DOT 99] Dotzauer E., Holmström K., Ravn H. F., « Optimal Unit Commitment and Economic Dispatch of Cogeneration Systems with a Storage », *Proceedings of the 13th Power Systems Computation Conference 1999 PSCC'99*, p. 738–744, 1999.

[HOB 88] Hobbs W. J., Hermon G., Warner S., Sheblé G. B., « An Enhanced Dynamic Programming Approach for Unit Commitment », *IEEE Transactions on Power Systems*, vol. 3, n° 3, p. 1201–1205, août 1988.

[MAC 02] Maciejowski J. M., *Predictive control with constraints*, Prentice-Hall, Harlow, 2002.

[RAV 94] Ravn H. F., Rygaard J. M., « Optimal scheduling of coproduction with a storage », *Engineering Optimization*, vol. 22, p. 267–281, 1994.

[ROO 94] Rooijers F. J., van Amerongen R. A. M., « Static Economic Dispatch for Cogeneration Systems », *IEEE Transactions on Power Systems*, vol. 9, p. 1392–1398, août 1994.

[SAK 02] Sakawa M., Kato K., Ushiro S., « Operational planning of district heating and cooling plants through genetic algorithms for mixed 0-1 linear programming », *European Journal of Operational Research*, vol. 137, p. 677–687, 2002.

[SAN 04] Sandou G., Font S., Tebbani S., Hiret A., Mondon C., « Optimisation par colonie de fourmis d'un site de génération d'énergie », *Journal Européen des Systèmes Automatisés*, vol. 38, n° 9/10, p. 1097–1119, 2004.

[SAN 07] Sandou G., Olaru S., « Ant colony and genetic algorithm for constrained predictive control of power systems », *HSCC 2007, Lecture Notes in Computer Science*, vol. 4416, p. 501–514, 2007.

[SAN 08] Sandou G., Font S., Tebbani S., Hiret A., Mondon C., « Feeding a genetic algorithm with an ant colony for constrained optimization - an application to the Unit Commitment problem », *5th International Conference on Informatics in Control, Automation and Robotics*, 2008.

[SEN 98] Sen S., Kothari D. P., « Optimal Thermal Generating Unit Commitment : a Review », *Electrical Power and Energy Systems*, vol. 20, n° 7, p. 443–451, 1998.

[SER 07] Serban A. T., Sandou G., « Mixed ant colony optimisation for the Unit Commitment problem », *ICANNGA 2007, Part I. Lecture Notes in Computer Science*, vol. 4431, p. 332–340, 2007.

[SOC 08] Socha K., Dorigo M., « Ant colony optimization for continuous domains », *European Journal of Operational Research*, vol. 185, n° 3, p. 1155–1173, 2008.

[STü 00] Stützle T., Hoos H. H., « MAX-MIN Ant System », *Future Generation Computer Systems*, vol. 16, p. 889–914, 2000.

[YIN 98] Yin Wa Wong S., « An Enhanced Simulated Annealing Approach to Unit Commitment », *Electrical Power and Energy Systems*, vol. 20, n° 5, p. 359–368, 1998.

Chapitre 7

Optimisation par colonie de fourmis pour la fabrication de barres d'aluminium

7.1. Introduction

Lorsqu'il s'agit de résoudre des problèmes d'optimisation combinatoire, les métaheuristiques sont des algorithmes bien adaptés aux conditions de résolution rencontrées dans les domaines pratiques. Elles offrent un compromis intéressant entre la qualité des solutions et le temps de calcul nécessaire à leur obtention. C'est en partie pourquoi ce domaine a pris beaucoup d'ampleur au cours des dernières années. De nombreuses méthodes y ont été introduites, comme l'optimisation par colonie de fourmis (OCF) (*Ant Colony Optimization* - ACO) qui est une stratégie de résolution dont le comportement est basé sur celui des fourmis réelles.

La métaheuristique OCF tente de reproduire le comportement évolutif de partage de l'apprentissage observé chez les fourmis réelles à l'aide d'une mémoire centrale, appelée « trace de phéromone » qui est alimentée par les membres de la collectivité. Cette mémoire guide les fourmis dans la construction de solutions, et celles-ci alimentent à leur tour la mémoire centrale, proportionnellement à la qualité des solutions produites. L'OCF est donc une méthode constructive dont le choix des éléments d'une solution est fait selon un compromis entre cet apprentissage commun et une heuristique gourmande (ou *visibilité*), orientée sur le problème à résoudre.

Depuis son introduction, au début des années 1990, cette métaheuristique s'est montrée efficace pour la résolution de nombreux problèmes. Le premier algorithme

Chapitre rédigé par Marc GRAVEL et Caroline GAGNÉ.

OCF, appelé *Ant System* (AS) [COL 91, DOR 91, DOR 92], a été défini afin de résoudre le problème de voyageur de commerce (VC). Il a alors produit des résultats préliminaires encourageants, sans toutefois rivaliser avec les meilleurs algorithmes connus dans la littérature pour résoudre ce problème [DOR 04]. Par la suite, plusieurs travaux ont été proposés afin d'en améliorer la performance et ont mené aux versions *Elitist Ant System* (EAS) [DOR 91, DOR 92, DOR 96], *Rank-Based Ant System* (*AS-Rank*) [BUL 99], *MAX-MIN Ant System* (MMAS) [STU 97, STU 00] et finalement *Ant Colony System* (ACS) [DOR 97]. Selon ces derniers, ACS constitue aujourd'hui la version la plus évoluée et l'une des plus performantes pour résoudre des problèmes d'optimisation combinatoire.

Nous présentons, dans ce chapitre, un cas réel d'application de la métaheuristique OCF en ordonnancement industriel dans le domaine de la fabrication de l'aluminium. Au niveau de la conception de cet algorithme, les principales difficultés résident dans les nombreuses contraintes liées à cette problématique industrielle et dans l'évaluation d'une solution qui demande une simulation complète de la production. A cet égard, la qualité de l'implémentation informatique de l'algorithme joue un rôle primordial pour produire une solution au décideur dans un délai raisonnable.

Nous décrivons l'ensemble de la démarche de développement de cette application et les travaux qui ont été réalisés par la suite. Nous allons comparer, entre autres, la performance de deux versions de l'OCF et démontrer l'importance, dans la conception, du choix de la visibilité, pour la performance de la métaheuristique. Nous proposons également des innovations à l'OCF, par l'utilisation de plusieurs visibilités liées au problème traité, et nous montrons l'importance, en contexte pratique, où il existe généralement plusieurs objectifs à optimiser, d'utiliser une approche de recherche de solutions de compromis. L'approche proposée constitue, par sa simplicité, sa flexibilité et sa performance, un outil d'aide à la décision pertinent, dans un contexte d'application en industrie, où le compromis entre le temps de calcul et la qualité des solutions est important.

Pour juger de la qualité des solutions produites dans le contexte industriel, la performance de nos algorithmes a été validée en utilisant des problèmes de même nature (machine unique avec réglages dépendant de la séquence) retrouvés dans la littérature. De cette façon, il est possible de réaliser une comparaison de performance avec d'autres algorithmes. Nous estimons qu'une telle démarche est essentielle, car pour un problème industriel, il est généralement difficile d'établir la performance des algorithmes développés.

7.2. L'ordonnancement de la fabrication de barres d'aluminium à l'usine Dubuc de la compagnie Alcan

Dans cette section, nous présentons les travaux réalisés à partir de 1998 au centre de coulée de l'usine Dubuc, propriété de la compagnie Alcan (maintenant Rio Tinto

Alcan), située au Québec dans la région du Saguenay. Ce centre de coulée est alimenté en aluminium pur, selon ses besoins, par des usines d'électrolyse situées à proximité. La problématique consiste à réaliser l'ordonnancement de commandes pour une machine à couler horizontale alimentée par deux fours de maintien de 21 tonnes métriques (TM) (1 tonne métrique = 1 000 kilogrammes). La figure 7.1 illustre les principales caractéristiques du procédé. Les fours servent à la préparation du mélange composé d'aluminium pur et d'ingrédients d'alliage appropriés pour répondre aux spécifications de chaque commande. Une fois le mélange complété, le métal est acheminé vers la machine à couler par un dalot (canal de coulée entre le four et la machine à couler) pour être formé. Pendant qu'un four réalise la coulée, l'autre four est en préparation, et celle-ci devra être terminée avant la fin de la coulée du premier four, pour éviter une interruption coûteuse de la production. Le moule de formage définit la largeur et la hauteur de chaque barre produite, tandis qu'une scie située à la sortie est utilisée pour couper la barre à la longueur désirée. Chaque commande possède des caractéristiques particulières déterminées, par le type d'alliage, le nombre de pièces identiques à produire, la dimension de ces pièces ainsi que la date de livraison.

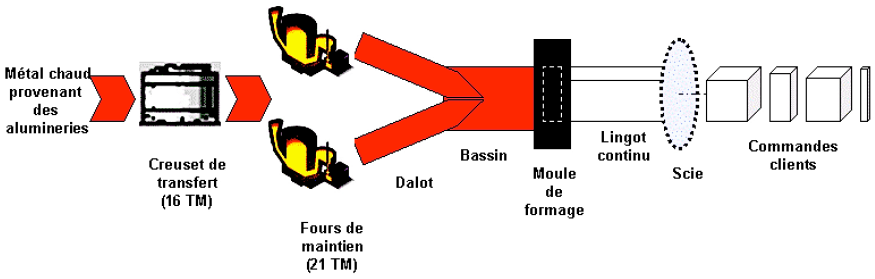


Figure 7.1. *Le processus de coulée horizontale à l'usine Dubuc*

Deux types de réglage peuvent survenir pendant la production. Un changement dans les dimensions des pièces à produire pour deux commandes successives entraîne un changement du moule de formage de la machine à couler. Une quantité de métal est alors nécessaire pour réchauffer le dalot, et celui-ci est considéré comme un rebut. Le deuxième type de réglage peut survenir au niveau des fours, lorsque deux commandes successives sont d'un alliage différent. Dans ce cas, il est possible qu'un drainage partiel ou encore un drainage à sec du four (drainage complet, suivi d'un nettoyage) soit nécessaire, avant la préparation de la recette de l'autre alliage. Bien sûr, les deux types de réglage peuvent survenir au même moment.

La coulée avec une machine horizontale étant un processus continu, l'évaluation d'un ordonnancement particulier doit établir s'il est possible d'alimenter celle-ci sans interruption à l'aide des deux fours. Un arrêt d'alimentation en métal entraîne automatiquement un changement du moule de formage, puisque des résidus de métal se sont

accumulés sur celui-ci. Un modèle de simulation est utilisé pour vérifier si la préparation d'un four peut être faite pendant la durée de la coulée de l'autre four. Cette durée dépend de la dimension des barres produites et de la vitesse de coulée propre à chaque alliage. Pour sa part, la préparation des fours se réalise avec du métal pur disponible en creusets de 16TM et de la refonte. Le niveau résiduel du four et la quantité de refonte utilisée sont les deux principaux facteurs influençant le temps de préparation du four. Une autre contrainte technologique réside dans le fait que les différents moules s'attachent à des bassins particuliers (figure 7.1) pour lesquels on ne dispose que d'un seul exemplaire dans certains cas. Alors, certaines séquences sont considérées non réalisables, parce que deux commandes successives de dimensions différentes utilisent le même bassin.

L'évaluation d'un ordonnancement est réalisée sur la base de trois objectifs, selon un traitement lexicographique. L'ordre de priorité de ceux-ci est établi par le planificateur. Ces objectifs sont :

– la minimisation de la perte de capacité de production du système entraînée par les différents réglages (L_1^+) ;

– la minimisation du retard total pour l'ensemble des commandes par rapport à leurs dates d'échéance (L_2^+) ;

– la minimisation d'une fonction de pénalité visant à regrouper les commandes devant être transportées vers une même destination, pour maximiser l'utilisation de la capacité des camions (L_3^+).

La section suivante décrit les travaux réalisés à l'aide de l'OCF pour concevoir une application et une implémentation en usine. Dans les autres sections, nous présentons la poursuite de nos travaux, sans que les résultats de ceux-ci n'aient été implémentés en usine. Le lecteur peut également consulter [GAG 01] pour plus de détails.

7.2.1. *Une application à base de l'*Ant System *(AS) pour l'ordonnancement de la production de barres d'aluminium à l'usine Dubuc*

La première version de l'OCF est l'algorithme AS. Elle a été inspirée par les études sur le comportement des fourmis réelles [DEN 83, DEN 89, GOS 90]. Le fonctionnement général de l'algorithme est présenté à la figure 7.2. Le problème classique traité avec cet algorithme est celui du VC où la distance (d_{ij}) entre chaque paire de villes ij sert à définir le concept de visibilité η_{ij} dans la règle de transition représentée par l'équation (7.1). La règle de transition établit, au cycle t, la probabilité p_{ij}^m qu'une fourmi m positionnée à la ville i se déplace vers la ville j. Une mémoire pour chacune des fourmis ($tabou_m$) permet alors de considérer uniquement les villes j ne faisant pas déjà partie de la tournée en cours de construction. Chaque probabilité est établie en considérant l'intensité de la trace (τ_{ij}) accumulée dans la matrice de phéromone sur l'arc ij et la visibilité η_{ij}, qui est définie par $1/d_{ij}$. Les paramètres α et β contrôlent

l'importance relative accordée à chacun de ces deux éléments :

$$p_{ij}^{m} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum\limits_{l \notin tabou_m} [\tau_{il}]^{\alpha} \cdot [\eta_{il}]^{\beta}} & \text{si } j \notin tabou_m \\ 0 & \text{si } j \in tabou_m \end{cases} \tag{7.1}$$

```
t = 0 ; Initialiser la matrice de trace de phéromone τ à une petite valeur τ0 pour chaque arc ij ;
POUR t < NbCycles FAIRE
    Déterminer aléatoirement une ville de départ pour chaque fourmi ;
    POUR pos = 2 à N FAIRE
        POUR m = 1 à nb_ants FAIRE
            Sélectionner la prochaine ville j, j ∉ Tabou_m, à être ajoutée à la séquence selon l'équation (7.1) ;
    POUR m = 1 à nb_ants FAIRE
        Evaluer la longueur de la tournée L^m pour chaque fourmi m ;
        Effectuer la mise à jour de la trace de phéromone à l'aide de chaque fourmi m ;
    Mettre à jour la meilleure solution connue à date ;
```

Figure 7.2. *Pseudo-code de l'algorithme AS*

Pour le problème de l'ordonnancement de la fabrication de barres d'aluminium à l'usine Dubuc, nous avons choisi, dans une première implémentation de l'algorithme, de remplacer la matrice des distances du problème de VC par une matrice de pénalités cumulatives associées à chacun des éléments du problème. De plus, la matrice D est de dimension $(N + 1) \times (N + 1)$, où N est le nombre total de commandes du carnet de commandes, pour faire le lien avec la période de planification précédente. Nous avons ainsi fait une transposition directe du problème de VC vers notre problème.

Les éléments de la matrice D sont établis de la manière suivante :

1) la matrice est initialisée à $d_{ij} = 1$ pour toutes les commandes ij ;

2) une pénalité de +2 est ajoutée à d_{ij} si les commandes i et j sont d'alliages différents et font en sorte qu'un drainage à sec soit nécessaire ;

3) une pénalité de +2 est ajoutée à d_{ij} si les commandes i et j demandent la production de pièces de dimension différente et requièrent ainsi un changement de moule ;

4) pour encourager le respect des dates d'échéance et dans le but de minimiser le retard total, une pénalité correspondant à deux fois le ratio [max(0, marge de la commande j)/marge maximale de toutes les commandes] est ajoutée à d_{ij}, où marge de la commande j = (date d'échéance) − (date de début de production) − (temps de production) ;

5) une pénalité de +2 est ajoutée à d_{ij} si les destinations de livraison des commandes i et j diffèrent ;

6) une pénalité de +500 est ajoutée à d_{ij} si les commandes i et j utilisent le même bassin et violent ainsi une contrainte technologique.

Compte tenu du traitement lexicographique des objectifs, les expérimentations numériques présentées dans [GRA 02] ont toutefois montré qu'il est préférable de construire la matrice D en incluant seulement les éléments qui sont en lien direct avec l'objectif principal, pour obtenir une meilleure performance. Par exemple, si l'objectif principal est la minimisation de la perte de capacité de production du système entraînée par les différents réglages (L_1^+), seules les étapes (1), (3), (4) et (6) devraient être utilisées pour construire la matrice D. Pour la minimisation du retard total pour l'ensemble des commandes par rapport à leurs dates d'échéance (L_2^+), seules les étapes (1), (2), (3), (4) et (6) sont nécessaires pour construire la matrice D, mais en utilisant une pénalité de +1 au lieu de +2 aux étapes (3) et (4). Pour la minimisation de la fonction reliée au transport (L_3^+), on devrait utiliser uniquement les étapes (1), (5) et (6).

Cette version de l'algorithme AS a été implémentée à l'usine Dubuc en mai 1999 en remplacement de l'algorithme génétique conçu initialement par les auteurs et utilisé depuis quelques mois. L'AS étant une approche constructive, elle a permis d'obtenir, dans ce cas, de meilleurs résultats que l'algorithme génétique. Il faut noter que, dans les deux cas, aucune méthode de recherche locale n'était intégrée à la métaheuristique. Il s'agit, à notre connaissance, de la première implémentation d'une version de l'OCF en milieu industriel. La figure 7.3 présente un aperçu de l'application logicielle implémentée à l'usine Dubuc.

La pertinence de dissocier les éléments particuliers du problème au niveau de la visibilité de AS a dirigé la suite de nos travaux vers l'utilisation de matrices de visibilité multiples plutôt qu'une seule matrice dans la règle de transition. La prochaine section décrit ces travaux que l'on retrouve dans C. Gagné *et al.* [GAG 02a] et qui utilisent l'algorithme ACS à la place de AS.

7.2.2. *Une application améliorée pour l'ordonnancement de la production de barres d'aluminium à l'usine Dubuc*

M. Dorigo et L.M. Gambardella [DOR 97] ont introduit une nouvelle version de l'OCF, nommée *Ant Colony System* (ACS). Celle-ci propose des changements au niveau de la règle de transition de AS, qui se définissent par une partie probabiliste et une partie déterministe, tout en utilisant une liste de candidats (*cl*) pour réduire les calculs. De plus, dans ACS, la mise à jour de la trace de phéromone se fait à la fois localement et globalement. Les auteurs ont également suggéré l'utilisation du 3-Opt restreint [JOH 97] à titre de méthode de recherche locale. L'équation (7.2), présentée ci-dessous, définit l'adaptation réalisée à la règle de transition de ACS pour le problème d'ordonnancement à l'étude. Le paramètre q_0 permet de déterminer la proportion des décisions prises de façon déterministe.

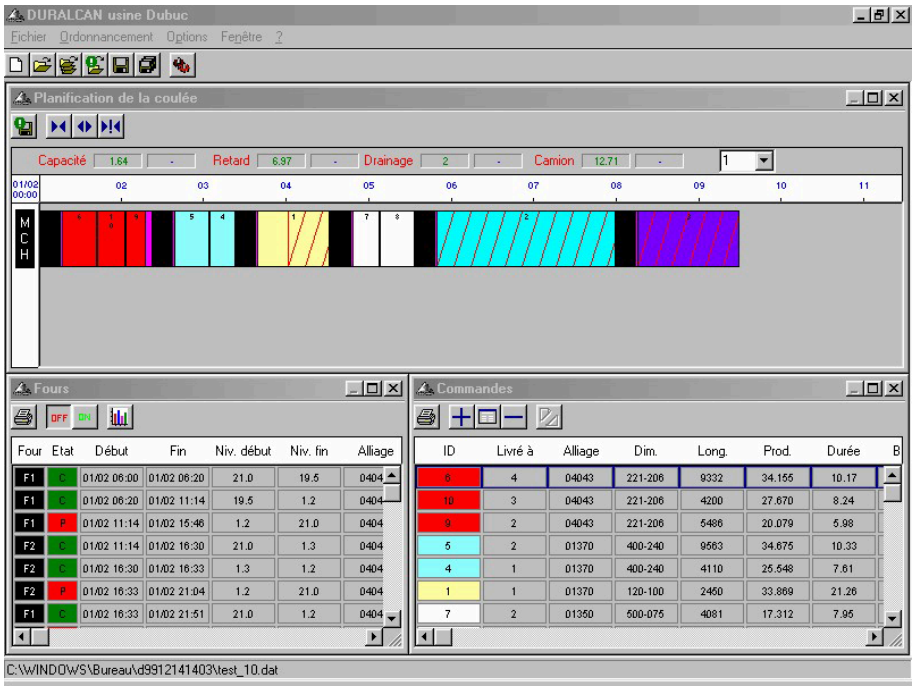


Figure 7.3. *Application logicielle de l'usine Dubuc*

Choisir la prochaine commande j, $j \notin tabou_m$, parmi les cl candidates selon :

$$j = \begin{cases} \arg\max_{l \notin tabou_m} \left\{ [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{s_{il}}\right]^{\beta} \cdot \left[\frac{1}{m_{il}}\right]^{\delta} \cdot \left[\frac{1}{c_{il}}\right]^{\lambda} \right\} & \text{si } q \leq q_0 \\ J & \text{si } q > q_0 \end{cases} \tag{7.2}$$

où J est choisie selon la probabilité :

$$p_{ij}^{m} = \frac{\left[\tau_{ij}\right]^{\alpha} \cdot \left[\frac{1}{s_{ij}}\right]^{\beta} \cdot \left[\frac{1}{m_{ij}}\right]^{\delta} \cdot \left[\frac{1}{c_{ij}}\right]^{\lambda}}{\sum_{l \notin tabou_m} [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{s_{il}}\right]^{\beta} \cdot \left[\frac{1}{m_{il}}\right]^{\delta} \cdot \left[\frac{1}{c_{il}}\right]^{\lambda}} \tag{7.3}$$

Dans [GAG 02a], nous avons proposé d'utiliser, dans les équations (7.2) et (7.3), trois éléments de visibilité, où chacun d'eux est en lien direct avec l'un des trois objectifs du problème. La matrice S contient les pénalités reliées aux deux types de réglage susceptibles d'affecter l'objectif L_1^+, la matrice M contient les marges des commandes par rapport à leurs dates d'échéance, qui sont en lien avec l'objectif L_2^+, et finalement,

la matrice C contient les pénalités reliées à l'objectif L_3^+. La figure 7.4 précise les éléments utilisés pour la construction des trois matrices de visibilité.

Il suffit alors de modifier les exposants α, β et λ des éléments de visibilité en fonction de l'ordre de priorité des objectifs, plutôt que de construire une nouvelle matrice D, comme dans l'application présentée à la section précédente.

Matrice S

(1) Initialisation de $s_{ij} = 2$ pour toutes les commandes ij.

(2) Une pénalité de +2 est ajoutée à s_{ij} si les commandes i et j demandent la production de pièces de dimension différente et obligent ainsi un changement de moule.

(3) Une pénalité de +1 est ajoutée à s_{ij} si un drainage est requis entre l'alliage de la commande i et celui de la commande j.

Matrice M

(1) Initialisation de $m_{ij} = 2$ pour toutes les commandes ij.

(2) Pour encourager le respect des dates d'échéance et ainsi minimiser le retard total, une pénalité correspondant à deux fois le ratio [max(0, marge de la commande j)/marge maximale de toutes les commandes] est ajoutée à m_{ij}. On a : marge de la commande j = date d'échéance - date de début de production - temps de production

(3) Une pénalité de +1 est ajoutée à s_{ij} si un drainage est requis entre l'alliage de la commande i et celui de la commande j.

Matrice C

(1) Initialisation de $c_{ij} = 2$ pour toutes les commandes ij.

(2) Une pénalité de +2 est ajoutée à c_{ij} si les destinations de livraison des commandes i et j diffèrent.

Figure 7.4. *Construction des matrices de visibilité S, M et C. Dans chacune des matrices, une pénalité de* +500 *est ajoutée lorsque les commandes successives i et j utilisent le même bassin et violent ainsi une contrainte technologique*

Il est maintenant intéressant de comparer la performance des deux versions de l'OCF avec l'utilisation d'une matrice de visibilité unique et de matrices de visibilité multiples. La minimisation du retard total (L_2^+) est l'objectif possédant le plus de variabilité et qui offre ainsi une meilleure possibilité de comparaison. En considérant celui-ci à titre d'objectif principal, il a été démontré dans C. Gagné *et al.* [GAG 02a], que pour les carnets de plus de trente commandes, l'utilisation de plusieurs matrices de visibilité avec AS permet d'obtenir statistiquement de meilleurs résultats qu'avec une seule matrice de visibilité. La figure 7.5 fournit une illustration de cette situation, avec un carnet de cinquante commandes. On observe, sur la base de dix essais, que le retard moyen passe de 88,68 jours avec une matrice de visibilité unique (AS-MU) à 26,41

jours avec l'utilisation de matrices de visibilité multiples (AS-MM). On n'observe toutefois pas, dans ce cas, de différences significatives en utilisant ACS (ACS-MM) au lieu de AS avec des matrices de visibilité multiples.

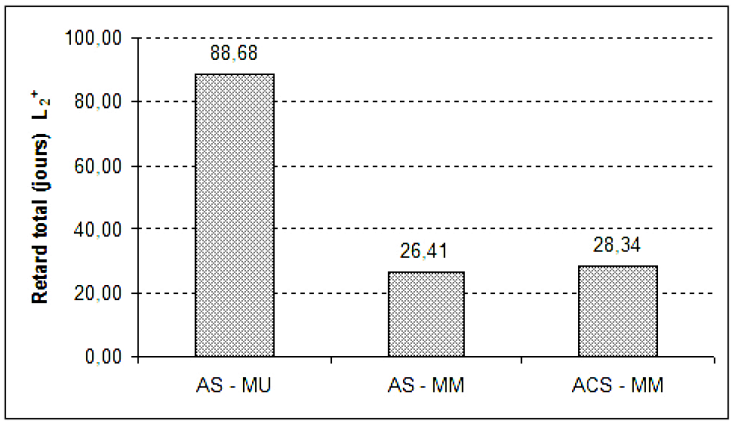


Figure 7.5. *Comparaison de performance de AS-MU, de AS-MM et de ACS-MM pour un problème de cinquante commandes*

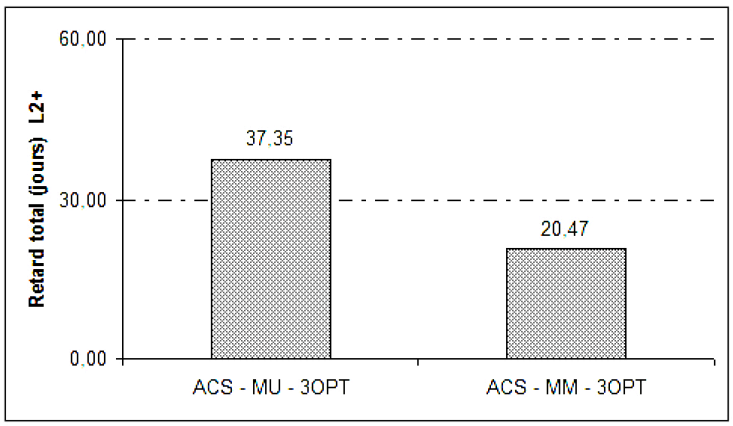


Figure 7.6. *Comparaison de performance de ACS-MU-3OPT et de ACS-MM-3OPT pour un problème de cinquante commandes*

Il a également été démontré par C. Gagné *et al.* [GAG 02a], que pour les carnets de plus de quarante commandes, l'utilisation de plusieurs matrices avec ACS utilisant le 3-OPT comme méthode de recherche locale permet d'obtenir statistiquement de meilleurs résultats qu'avec une seule matrice de visibilité incluant une procédure de

recherche locale. La figure 7.6 fournit une illustration de cette situation, avec un carnet de cinquante commandes. On observe que le retard moyen passe de 37,35 jours, avec une matrice de visibilité unique (ACS-MU-3OPT), à 20,47 jours, avec l'utilisation de matrices de visibilité multiples (ACS-MM-3OPT).

Il est donc permis de conclure que, peu importe la version de l'OCF, l'utilisation de matrices de visibilité multiples devient un avantage intéressant pour la performance. Elle permet, entre autres, de mieux adapter l'algorithme au problème à résoudre, dans le cas où plusieurs objectifs sont à optimiser. Même si les deux versions de l'OCF semblent donner des résultats identiques, selon M. Dorigo et L.M. Gambardella [DOR 97], ACS constitue aujourd'hui la version la plus évoluée et l'une des plus performantes pour résoudre des problèmes d'optimisation combinatoire. Nous avons également confirmé la meilleure performance de ACS en comparaison avec les algorithmes AS, MMAS, EAS et AS-Rank, pour le problème d'ordonnancement de voitures [GAG 08] présenté au chapitre 8.

Dans [GAG 02a], nous avons également proposé l'utilisation d'une autre forme de visibilité (*lookahead*) qui n'a pas permis, pour le problème de fabrication de barres d'aluminium, d'obtenir une amélioration significative de la qualité des résultats. Toutefois, dans [GAG 02b], cette forme de visibilité s'est avérée intéressante pour un autre problème d'ordonnancement. Dans le cas du problème industriel, la fonction d'évaluation des objectifs, pour une séquence donnée, demande une simulation complète de la production. L'utilisation du *lookahead* s'avère donc, dans ce cas, très désavantageuse en temps de calcul.

Nous allons maintenant présenter la suite de ces travaux où l'on cherche à fournir au décideur une information plus complète pour le choix de la séquence de production à mettre en œuvre.

7.2.3. *Solutions de compromis pour l'ordonnancement de la production de barres d'aluminium à l'usine Dubuc*

Le traitement lexicographique des objectifs est simple à mettre en œuvre d'un point de vue algorithmique, mais il est peu probable qu'en contexte industriel un planificateur recherche ce type de solutions. Etant donné que le planificateur de la production a défini trois objectifs dans la formulation du problème, il vise plutôt à trouver des solutions de compromis (solutions Pareto-optimales) entre les différents objectifs.

Dans les situations pratiques comme celle de la coulée de l'aluminium, les temps de calcul peuvent s'avérer importants si l'on cherche à générer entièrement l'ensemble Pareto-optimal. Nous avons abordé précédemment le fait que le temps de calcul est important pour réaliser l'évaluation d'une séquence de production dans le problème de fabrication des barres d'aluminium. Il est alors préférable de demander au décideur

ses préférences pour les différents objectifs et de produire exclusivement les solutions Pareto-optimales correspondant à celles-ci. C'est dans cette optique qu'une procédure générique, pouvant être adaptée à différentes métaheuristiques pour la recherche de solutions de compromis, a été proposée dans C. Gagné *et al.* [GAG 05]. Dans cet article, un exemple d'application de la procédure générique a été présenté à l'aide de la recherche avec tabous (*Tabu Search*) hybridée avec la recherche par voisinage variable (*Variable Neighborhood Search*) pour traiter un problème d'ordonnancement bi-objectif sur une machine unique, avec réglages dépendant de la séquence. Ces résultats ont été appliqués par la suite au problème de fabrication des barres d'aluminium de l'usine Dubuc [GAG 04].

La forme générale d'un problème d'optimisation avec objectifs multiples à minimiser peut être représentée par le programme suivant :

$$\text{« minimiser »} \qquad f(x) = \{f_1(x), f_2(x), \ldots, f_z(x)\} \qquad \text{sujet à } x \in E$$

où $|z| \geq 2$ représente le nombre d'objectifs à optimiser, $x = \{x_1, x_2, \ldots, x_N\}$ représente un vecteur de variables de décision de N dimensions, E est l'ensemble de solutions réalisables, et $f(x)$ est le vecteur des objectifs à optimiser. Etant donné que les objectifs sont habituellement conflictuels, il est généralement impossible de trouver une solution unique qui procure simultanément la solution optimale pour l'ensemble des objectifs. Par conséquent, la résolution du programme précédent consiste à rechercher un ensemble de solutions $E^* \subseteq E$ appelé ensemble Pareto-optimal ou ensemble efficace. S. Purao *et al.* [PUR 99] définissent qu'une solution réalisable $x \in E$ domine une solution réalisable $y \in E$ ssi $\exists u \in z | f_u(x) < f_u(y)$, et que $\forall v \in z, v \neq u$ on a $f_v(x) \leq f_v(y)$. De plus, une solution réalisable x est efficace ou non dominée (Pareto-optimale) si et seulement si il n'y a aucune autre solution réalisable qui la domine.

Supposons que l'optimisation de chaque objectif pris séparément ait permis de trouver les différentes solutions optimales. Le *point idéal* peut être exprimé comme suit : $F^* = \{F_1^*, F_2^*, \ldots, F_z^*\}$ où $F_v^* = min_{x \in E} f_v(x)$. Il représente, dans le cas d'objectifs conflictuels, le but à atteindre et correspond généralement à une solution non réalisable. A l'aide de la méthode du *Compromise Programming* (CP) proposée par M. Zeleny [ZEL 82], il est alors possible d'agréger les différents objectifs en une seule mesure, permettant de refléter la qualité d'une solution réalisable quelconque, en calculant sa distance au point idéal.

En utilisant cette idée dans l'optimisation multi-objectif, il suffit alors de rechercher, dans l'espace des solutions admissibles E, la solution minimisant la distance au point idéal F^*. L'importance relative accordée à chacun des objectifs par le décideur peut également être exprimée par un ensemble de pondérations $p_k \geq 0, k \in z | \Sigma p_k = 1$ et être prise en compte dans le calcul d'une distance pondérée au point idéal.

La figure 7.7 présente, pour le problème de cinquante commandes, une estimation de l'ensemble Pareto-optimal (ou ensemble de référence) obtenue à partir de nombreux essais numériques réalisés à l'aide de ACS-MM-3OPT dans diverses conditions, en considérant deux objectifs seulement. Le point indiqué par un cercle dans la partie inférieure gauche de la figure représente le point idéal pour ce problème. Une distance normalisée *dist* peut être calculée entre n'importe quel point de l'ensemble de référence et le point idéal. Pour effectuer ce calcul, les poids accordés à chacun des objectifs doivent être spécifiés pour pondérer la distance, selon l'importance de chacun des objectifs.

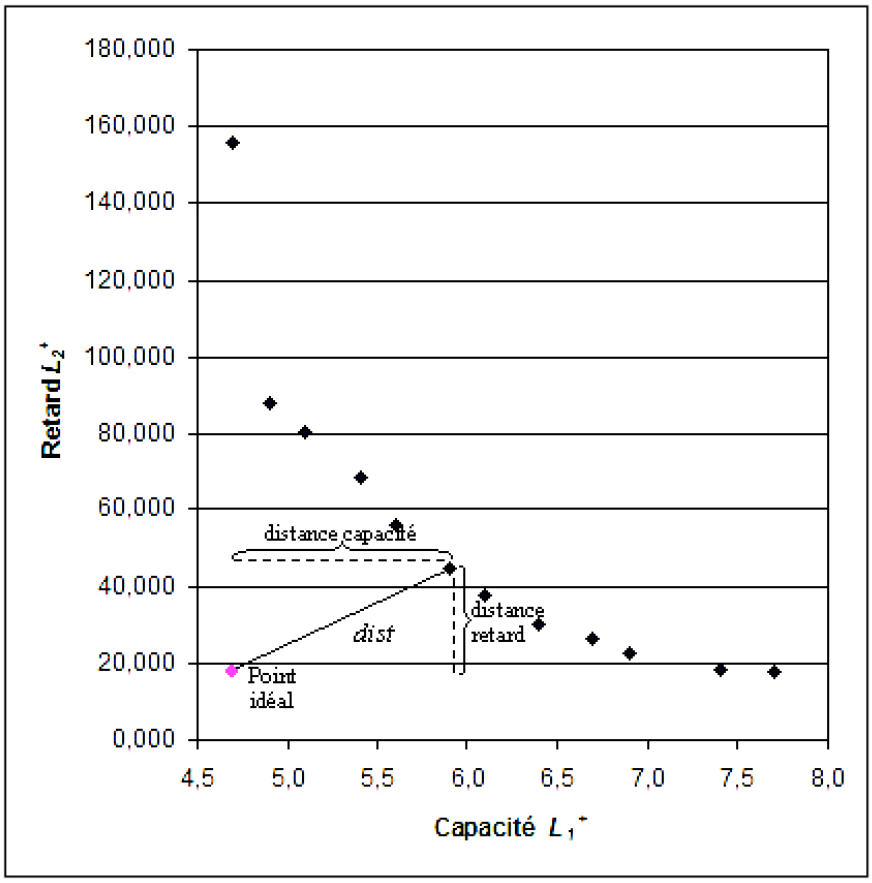


Figure 7.7. *Point idéal et ensemble de référence pour un problème de cinquante commandes avec l'optimisation de deux objectifs : minimisation de la perte de capacité de production (L_1^+) et du retard total (L_2^+)*

A l'aide de ACS-MM-3OPT présenté au paragraphe 7.2.2 et en priorisant les objectifs dans l'ordre L_1^+ - L_2^+, on obtient, sur la base de dix essais, les résultats moyens $L_1^+ = 4{,}75$ et $L_2^+ = 190{,}12$ tandis que, dans l'ordre inverse, on obtient $L_2^+ = 20{,}47$ et $L_1^+ = 7{,}54$. Ces solutions peuvent être considérées d'excellente qualité, selon l'estimation de l'ensemble Pareto-optimal présenté à la figure 7.7. On peut également comprendre, à l'aide de cette figure, que le décideur pourrait accepter facilement une perte de capacité supplémentaire d'une journée ($L_1^+ = 5{,}75$) pour diminuer le retard total d'environ 140 jours, en accordant un peu moins d'importance à l'objectif L_1^+. A l'inverse, il accepterait probablement une augmentation du retard total d'environ dix jours pour sauver une journée en perte de capacité. Donc, si le décideur pouvait accorder une pondération ($p_k \geq 0$) à chacun des objectifs, plutôt que de donner un ordre strict de priorité à ceux-ci, la qualité des décisions prises refléterait davantage ses attentes.

Phase 1 : RECHERCHE DU POINT IDEAL
(a) Calibrage des paramètres de la métaheuristique
(b) Optimisation des objectifs séparément pour déterminer le point idéal

Phase 2 : RECHERCHE DE SOLUTIONS DE COMPROMIS
(a) Enoncé des préférences du décideur (pondérations des objectifs)
(b) Processus itératif : (jusqu'à ce que les conditions d'arrêt soient respectées)
  (i) Construction ou génération de solutions par une métaheuristique
  (ii) Evaluation des solutions ainsi générées sur les différents objectifs
  (iii) Vérification des relations de dominance et enregistrement des solutions
  (iv) Calcul de la distance normalisée et pondérée par rapport au point idéal
(c) Application de perturbations sur la pondération des objectifs et retour à l'étape (b) de la phase 2

Phase 3 : PRESENTATION DES SOLUTIONS DE COMPROMIS
(a) Réduction du nombre de solutions à présenter au décideur
(b) Affichage des solutions de compromis

Figure 7.8. *Etapes d'une métaheuristique générique pour la recherche de solutions de compromis [GAG 04]*

La procédure présentée à la figure 7.8 a été proposée dans [GAG 04] pour trouver les solutions de compromis à l'aide de n'importe laquelle des métaheuristiques. Elle permet de générer la partie de l'ensemble Pareto-optimal correspondant aux préférences du décideur. Par exemple, en accordant une pondération de 0,8 à l'objectif L_1^+ et de 0,2 à l'objectif L_2^+, on peut fournir au décideur une estimation intéressante d'une partie de l'ensemble de solutions Pareto-optimal, tel que présenté à la figure 7.9a. Les

sept solutions proposées au décideur par ACS-MM-3OPT^{MO} en fonction des préférences exprimées sont indiquées à l'aide de triangles et sont superposées à l'ensemble de référence, qui est exprimé par des losanges. Les points encerclés et encadrés sur le graphique indiquent respectivement la solution de ACS-MM-3OPT^{MO} et celle de l'ensemble de référence, qui possèdent la distance normalisée et pondérée minimale au point idéal. En accordant maintenant une pondération de 0,2 à l'objectif L_1^+ et de 0,8 à l'objectif L_2^+, l'estimation de l'ensemble de solutions Pareto-optimal produit par ACS-MM-3OPT^{MO} est composée de six solutions qui se retrouvent maintenant plus à droite de la courbe présentée à la figure 7.9b.

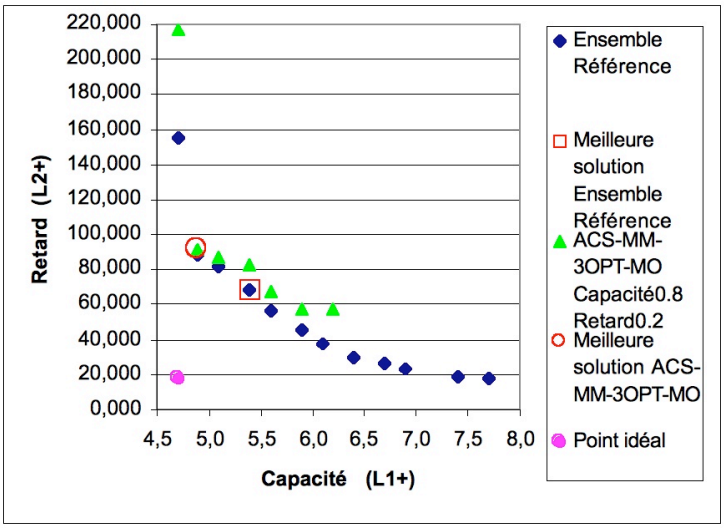
Pondérations		Solutions proposées en privilégiant un seul objectif	
L_1^+	L_2^+	L_1^+	L_2^+
1,0	0,0	4,75	180,15
		4,90	93,59
		5,11	83,93
0,0	1,0	7,43	18,49
		7,23	23,15
		6,91	26,99
		6,73	38,29
		6,42	42,30

Tableau 7.1. *Solutions de compromis obtenues en privilégiant un seul objectif*

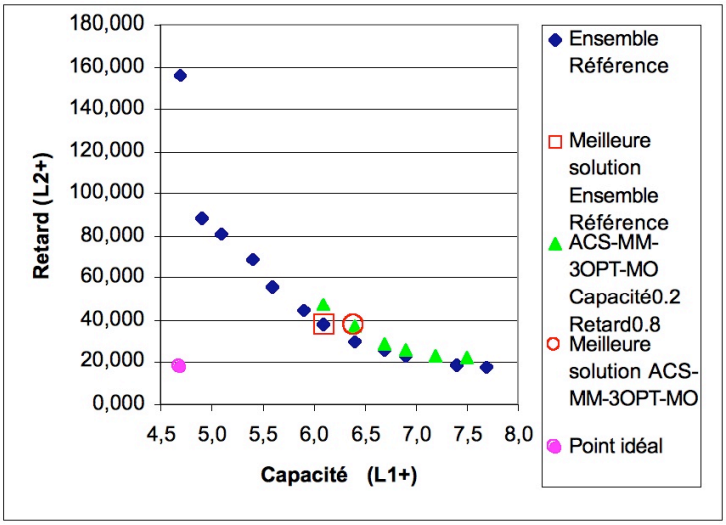
Même dans les cas où le décideur accorde une pondération de 1 à un des objectifs, ACS-MM-3OPT^{MO} propose alors un ensemble de solutions au décideur, plutôt qu'une seule solution, comme par une approche lexicographique. Par exemple, au tableau 7.1, dans le cas où le décideur a accordé une importance absolue à l'objectif L_1^+, il suggère un ensemble de trois solutions, tandis que l'approche lexicographique proposait uniquement la solution $L_1^+ = 4{,}75$ et $L_2^+ = 190{,}12$. Le décideur privilégiera sûrement une solution telle $L_1^+ = 4{,}90$ et $L_2^+ = 93{,}59$, pour économiser près de cent jours de retard, tout en induisant un temps improductif supplémentaire relativement minime dans l'usine. De même, dans le cas où le décideur a accordé une importance absolue à l'objectif L_2^+, ACS-MM-3OPT^{MO} suggère un ensemble de cinq solutions, tandis que l'approche lexicographique proposait la solution $L_2^+ = 20{,}47$ et $L_1^+ = 7{,}54$. Le décideur prend ainsi conscience qu'il peut économiser une demi-journée en perte de capacité (L_1^+), en acceptant un retard total supplémentaire de quelques jours.

L'approche proposée peut facilement s'appliquer avec plus de deux objectifs. Le lecteur peut se référer à [GAG 04] pour les résultats du problème de fabrication de barres d'aluminium avec trois objectifs. En raison de sa simplicité, de sa flexibilité et de sa performance, cette approche de résolution constitue un outil d'aide à la décision pertinent, dans un contexte d'application en industrie, où le compromis entre le temps de calcul et la qualité des solutions est important. De plus, l'outil d'aide à la décision,

dans sa forme actuelle, comporte toutes les composantes nécessaires pour ajouter une quatrième phase permettant d'interagir avec le décideur, et ainsi constituer un outil interactif.



(a) Pondération de 0,8 sur l'objectif L_1^+ et de 0,2 sur l'objectif L_2^+



(b) Pondération de 0,2 sur l'objectif L_1^+ et de 0,8 sur l'objectif L_2^+

Figure 7.9. *Comparaison de la performance de ACS-MM-3OPT^{MO} par rapport à l'ensemble de référence : problèmes de cinquante commandes*

7.3. Conclusion

Dans ce chapitre, nous avons présenté un cas réel d'application de la métaheuristique OCF en ordonnancement industriel dans le domaine de la fabrication de l'aluminium. Cet exemple illustre bien le fait que les métaheuristiques sont des stratégies de résolution qui doivent, en tout temps, être adaptées pour incorporer les éléments spécifiques du problème traité, pour obtenir une performance intéressante. En particulier, dans le cas de l'OCF, le ou les termes de visibilité sont très importants pour guider le processus de construction des solutions.

Dans cette application, nous avons introduit l'utilisation de matrices de visibilité multiples, plutôt que de faire une simple transposition vers le problème de VC. Ce changement au fonctionnement classique de l'OCF a permis d'améliorer la qualité des résultats produits par la métaheuristique. Nous avons également proposé une procédure efficace de traitement des objectifs, pour l'obtention de solutions de compromis, plutôt qu'un traitement lexicographique des objectifs. Ce type de solutions correspond davantage aux préoccupations d'un planificateur de production.

Il faut noter que nous avons également réalisé plusieurs autres implantations d'applications en ordonnancement dans différentes usines québécoises de la compagnie Rio Tinto Alcan. Celles-ci sont basées sur l'utilisation des algorithmes génétiques et de la recherche avec tabous. Nous estimons que chaque métaheuristique possède des forces et des faiblesses et que le travail réalisé lors de la conception, pour prendre en considération les spécifications du problème à résoudre, représente l'élément-clé, pour obtenir une performance intéressante avec chacune d'elles.

7.4. Bibliographie

[BUL 99] Bullnheimer B., Hartl R., Strauss C., « An Improved Ant system Algorithm for the Vehicle Routing Problem », *Annals of Operations Research*, vol. 89, p. 319–328, 1999.

[COL 91] Colorni A., Dorigo M., Maniezzo V., « Distributed optimization by ant-colonies », F. Varela, P. Bourgine (dir.), *Proceedings of the First European Conference on Artificial Life (ECAL'91)*, p. 134-142, Cambridge, Mass, Etats-Unis, MIT Press, 1991.

[DEN 83] Deneubourg J. L., Pasteels J. M., Verhaeghe J. C., « Probabilistic behaviour in ants : A strategy of errors ? », *Journal of Theorical Biology*, vol. 105, p. 259–271, 1983.

[DEN 89] Deneubourg J. L., Goss S., « Collective patterns and decision-making », *Ethology & Evolution*, vol. 1, p. 295–311, 1989.

[DOR 91] Dorigo M., Maniezzo V., Colorni A., Positive feedback as a search strategy, Rapport n° 91-016, Politecnico di Milano, Italie, 1991.

[DOR 92] Dorigo M., Optimization, learning and natural algorithms, PhD thesis, Politecnico di Milano, Italie, 1992.

[DOR 96] Dorigo M., Maniezzo V., Colorni A., « Ant system : optimization by a colony of cooperating agents », *IEEE Transactions on Systems, Man & Cybernetics*, vol. 26, n° 1, p. 29–41, 1996.

[DOR 97] Dorigo M., Gambardella L., « Ant Colony system : A Cooperative learning approach to the Traveling Salesman Problem », *IEEE Transactions on Evolutionary Computation*, vol. 1, n° 1, p. 53–66, 1997.

[DOR 04] Dorigo M., Stützle T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.

[GAG 01] Gagné C., L'ordonnancement industriel : stratégies de résolution métaheuristiques et objectifs multiples, PhD thesis, Université Laval, Québec, Canada, 2001.

[GAG 02a] Gagné C., Gravel M., Price W., « Algorithme d'optimisation par colonie de fourmis avec matrices de visibilité multiples pour la résolution d'un problème d'ordonnancement industriel », *Information Systems and Operational Research (INFOR)*, vol. 40, n° 2, p. 259–276, 2002.

[GAG 02b] Gagné C., Price W., Gravel M., « Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence dependent setup times », *Journal of the Operational Research Society*, vol. 53, n° 8, p. 895–906, 2002.

[GAG 04] Gagné C., Gravel M., Price W., « Optimisation multi-objectifs à l'aide d'un algorithme de colonie de fourmis », *Information Systems and Operational Research (INFOR)*, vol. 42, n° 1, p. 23–42, 2004.

[GAG 05] Gagné C., Gravel M., Price W., « Using metaheuristic compromise programming for the solution of multiple objective scheduling problems », *Journal of the Operational Research Society*, vol. 56, n° 5, p. 687–698, 2005.

[GAG 08] Gagné C., Gravel M., Morin S., Price W., « Impact of the pheromone trail on the performance of ACO algorithms for solving the car sequencing problem », *Journal of the Operational Research Society*, vol. 59, n° 8, p. 1077–1090, 2008.

[GOS 90] Goss S., Beckers R., Deneubourg J. L., Aron S., Pasteels J. M., « How trail laying and trail following can solve foraging problems for ant colonies », R. Hughes (dir.), *Behavioural Mechanisms of Food Selection*, vol. G20 de *NATO-ASI Series*, Springler-Verlag, Berlin, 1990.

[GRA 02] Gravel M., Price W., Gagné C., « Scheduling continuous casting of aluminium using a multiple-objective ant colony optimization metaheuristic », *European Journal of Operational Research*, vol. 143, n° 1, p. 218–229, 2002.

[JOH 97] Johnson D. S., McGeoch L. A., « The traveling salesman problem : a case study in local optimization », p. 215–310, Local Search in Combinatorial Optimization, 1997.

[PUR 99] Purao S., Jain H., Nazareth D., « Supporting decision making in combinatorially explosive multicriteria situations », *Decision Support Systems*, vol. 26, p. 225–247, 1999.

[STU 97] Stützle T., Hoos H., « The MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems : Towards Adaptive Tools for Global Optimization », S. Voss, S. Martello, I. H. Ossmann, C. Roucairol (dir.), *Meta-Heuristic, Advances and Trends in Local Search Paradigma for Optimization*, p. 313–329, Kluwer Academic, 1997.

[STU 00] Stützle T., Hoos H., « MAX-MIN ant system », *Future Generation Computer Systems Journal*, vol. 16, n° 8, p. 889–914, 2000.

[ZEL 82] Zeleny M., *Multiple criteria decision making*, McGraw-Hill, 1982.

Chapitre 8

Optimisation par colonie de fourmis pour l'ordonnancement d'une chaîne d'assemblage automobile

8.1. Introduction

Au chapitre précédent, nous avons montré que, quelle que soit la version de l'optimisation par colonie de fourmis (OCF) utilisée, diverses adaptations sont rendues nécessaires pour obtenir une performance intéressante en milieu industriel. L'application présentée dans le présent chapitre pour l'ordonnancement d'une chaîne d'assemblage automobile fait ressortir de nouveau ce même constat. Toutefois, pour cette deuxième application, la performance de l'algorithme développé à l'aide de l'OCF est comparée à plusieurs autres algorithmes élaborés spécifiquement pour traiter ce problème industriel proposé par Renault, dans le cadre du Challenge ROADEF 2005 [NGU 05, SOL 08]. Les résultats obtenus devraient convaincre le lecteur que cette métaheuristique est tout à fait appropriée pour traiter différents contextes industriels et démontrer, une fois de plus, qu'elle doit toujours être adaptée spécifiquement au contexte étudié, pour être performante.

Le problème de l'ordonnancement d'une chaîne d'assemblage automobile consiste à déterminer l'ordre dans lequel un ensemble de voitures sont fabriquées sur une ligne, composée de trois ateliers consécutifs, telle qu'illustrée à la figure 8.1. A la section 8.2, nous présentons les travaux réalisés avec l'OCF pour la version théorique de ce problème industriel, qui consiste à ne tenir compte que de certaines contraintes de l'atelier

Chapitre rédigé par Caroline Gagné et Marc Gravel.

de montage [DIN 88]. Nous montrons que les différentes versions de l'OCF développées ont constamment permis d'améliorer les meilleurs résultats de la littérature pour ce problème d'ordonnancement uni-objectif. Dans ce cas, les essais numériques ont été réalisés sur des instances de la littérature contenant jusqu'à 400 voitures. Par la suite, à la section 8.3, nous présentons les travaux réalisés pour le problème industriel, tel que formulé par le constructeur automobile français Renault dans le cadre du Challenge ROADEF 2005. Nous montrons alors que l'OCF est tout à fait compétitive pour ce problème d'ordonnancement multi-objectif par rapport à plusieurs autres algorithmes. Dans ce cas, la taille de certaines instances dépasse 1 000 voitures à ordonnancer.

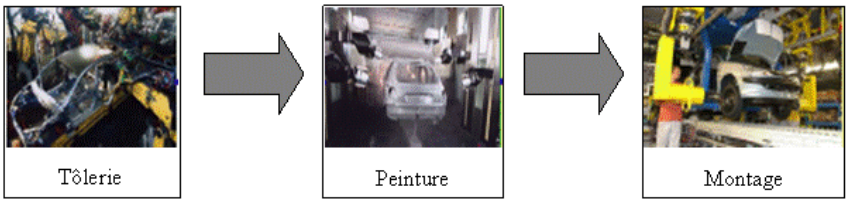


Figure 8.1. *Les trois ateliers d'une chaîne d'assemblage automobile [NGU 05]*

La gestion de la trace de phéromone représente un élément central dans les différents algorithmes d'OCF. Elle a fait l'objet de la majorité des modifications [DOR 04] qui ont mené aux différentes versions, telles *Ant System* (AS) [COL 91, DOR 91, DOR 92], *Elitist Ant System* (EAS) [DOR 91, DOR 92, DOR 96], *Rank-Based Ant System* (*AS-Rank*) [BUL 99], *MAX-MIN Ant System* (MMAS) [STU 97, STU 00] et *Ant Colony System* (ACS) [DOR 97]. Cependant, peu de travaux s'attardent à la modification de la structure interne de la trace de phéromone (encodage de la phéromone). En fait, cette dernière est généralement une transposition directe de celle utilisée pour le problème du voyageur de commerce (VC), même si la nature des problèmes traités est souvent très différente. A travers la démarche présentée dans ce chapitre, nous proposons l'utilisation d'une nouvelle structure de trace de phéromone spécialisée pour résoudre ce problème. Ceci met encore plus en évidence le fait que toutes les composantes de l'OCF doivent être adaptées au problème étudié pour obtenir une performance intéressante. Beaucoup d'applications utilisent des versions non adaptées de l'OCF, tout en intégrant la recherche locale comme élément principal pour l'amélioration de la qualité des solutions. Par la présentation de cette application et des travaux qui ont été réalisés par la suite, nous voulons sensibiliser les chercheurs à passer plus de temps au niveau de la conception, quelle que soit la métaheuristique utilisée, plutôt que d'utiliser très rapidement, dans leur développement, la recherche locale ou l'hybridation pour améliorer la performance.

8.2. Résolution du problème théorique d'ordonnancement d'une chaîne d'assemblage automobile à l'aide de l'OCF

Le problème théorique d'ordonnancement de voitures (POV) ne tient compte que des contraintes de l'atelier de montage. Dans cet atelier, chaque voiture est caractérisée par un ensemble d'options O (toit ouvrant, système ABS, etc.) où chacune d'elles est réalisée par un poste conçu pour traiter au plus un certain pourcentage de voitures passant dans l'atelier. Pour éviter que la capacité des postes ne soit dépassée, les voitures nécessitant les mêmes options doivent, par conséquent, être espacées afin de réguler le rythme de production. Ces contraintes peuvent s'exprimer sous la forme d'un ratio r_0/s_0 signifiant que pour toute sous-séquence consécutive de s voitures, au plus r voitures peuvent posséder l'option o. T. Kiss [KIS 04] a démontré que ce problème est NP-difficile.

Le POV a été décrit à l'origine par B. Parello *et al.* [PAR 86, PAR 88] avec l'objectif de déterminer un ordonnancement visant à minimiser, pour chaque option o, le nombre de sous-séquences de longueur s pour lesquelles il existe plus de r voitures possédant cette option. Lorsqu'une de ces *contraintes de capacité* est violée, on dit qu'il existe un *conflit*. Pour simplifier le problème, les voitures possédant exactement la même configuration d'options sont regroupées en classes. Pour chaque classe v, c_v représente alors le nombre de voitures à produire. Ces quantités définissent les *contraintes de production* du problème.

Le tableau 8.1 présente un exemple pour la production de 25 voitures (NC) possédant cinq options (O) et formant six classes (V). On note B la matrice des valeurs booléennes spécifiant si les voitures de la classe v possèdent l'option o. On a également le vecteur Y qui représente une séquence de production telle qu'illustrée au tableau 8.2.

			Classe #					
o	***r***	***s***	**1**	**2**	**3**	**4**	**5**	**6**
1	1	2	0	1	1	0	0	0
2	2	5	1	0	1	0	1	1
3	1	3	0	1	0	0	0	0
4	3	5	0	0	0	1	0	1
5	2	3	0	1	1	0	1	0
c_v			5	5	4	4	3	4

Tableau 8.1. *Exemple d'un POV*

Position	**1**	**2**	**3**	**4**	**5**	**6**	....	**21**	**22**	**23**	**24**	**25**
Y	3	5	5	4	6	4		3	1	4	5	1

Tableau 8.2. *Solution d'un POV*

Pour évaluer le nombre de conflits de la solution Y, on construit tout d'abord une matrice A de dimension $O \times NC$. On a alors $a_{ok} = 1$, dans le cas où la classe de voitures assignée à la position k de la solution requiert l'option o, et $a_{ok} = 0$, dans le cas contraire. La décomposition, pour les différentes options de la solution Y du tableau 8.2, permet ainsi de créer la matrice A présentée au tableau 8.3. L'évaluation de cette solution montre, par exemple, la présence de conflits aux positions 1 à 5, 2 à 6, 3 à 7, 20 à 24 et 21 à 25, pour l'option 2 ainsi qu'un conflit aux positions 1 à 3, pour l'option 5.

Position		**1**	**2**	**3**	**4**	**5**	**6**	....	**21**	**22**	**23**	**24**	**25**
Y		3	5	5	4	6	4		3	1	4	5	1
Option	1/2	1	0	0	0	0	0		1	0	0	0	0
	2/5	**1**	**1**	**1**	**0**	**1**	**0**		**1**	**1**	**0**	**1**	**1**
	1/3	0	0	0	0	0	0		0	0	0	0	0
	3/5	0	0	0	1	1	1		0	0	1	0	0
	2/3	**1**	**1**	**1**	0	0	0		1	0	0	1	0

Tableau 8.3. *Evaluation d'une solution*

Pour solutionner ce problème, nous avons proposé un ACS [GRA 05], noté ACS-2D dans la suite de ce chapitre, dont les principaux éléments sont présentés à la figure 8.2.

$t = 0$;
Initialiser la matrice de trace de phéromone $\tau(t)$ à une petite valeur τ_0 pour chaque paire de classes ij ;
POUR $t <$ *NbCycles* ET $L^{Bg} > 0$ conflit FAIRE
  Déterminer aléatoirement une classe de départ pour chaque fourmi ;
  POUR $pos = 2$ à NC FAIRE
    POUR $m = 1$ à *nb_ants* FAIRE
      Sélectionner la prochaine classe j, $j \notin tabou_m$, à être ajoutée à la séquence à la suite de la classe i parmi les cl classes candidates selon l'équation (8.1) ;
      Effectuer la mise à jour locale de la trace de phéromone pour (i, j) selon l'équation (8.5) ;
  POUR $m = 1$ à *nb_ants* FAIRE
    Evaluer le nombre de conflits L^m pour chaque fourmi m ;
  Effectuer la mise à jour globale de la trace de phéromone à l'aide de la meilleure solution du cycle Sol^{Bc} selon l'équation (8.6) ;
  Mettre à jour la meilleure solution connue Sol^{Bg} ;

Figure 8.2. *Pseudo-code de l'ACS-2D pour le POV*

La trace de phéromone est stockée dans une matrice τ symétrique à deux dimensions de taille $V \times V$, où V est le nombre de classes de l'instance. L'élément $\tau_{i,j}$ reflète l'intérêt de placer une voiture de classe i adjacente à une voiture de classe j. La diagonale de la matrice est utilisée, puisque deux voitures de même classe peuvent être

placées côte à côte. Les mises à jour de la trace se font toujours de manière symétrique. Au départ de l'algorithme, chaque élément de la matrice est initialisé à τ_0, qui représente une petite valeur positive.

L'algorithme se déroule pour *Nb_Cycles*, ou tant que le nombre de conflits (L^{Bg}) de la meilleure solution (Sol^{Bg}) demeure supérieur à zéro. Au début de chaque cycle, on détermine une classe de voitures de départ pour chacune des *nb_ants* fourmis. De la position 2 à NC, les classes de voitures j sont déterminées de manière séquentielle à l'aide d'une règle de transition (équation (8.1)). Un paramètre q_0 permet de déterminer la proportion des décisions prises de façon déterministe :

$$j = \begin{cases} \arg \max\limits_{l \notin tabou_m} \left\{ [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot [d_l]^{\delta} \right\} & \text{si } q \leq q_0 \\ J & \text{si } q > q_0 \end{cases} \tag{8.1}$$

où J est choisi selon la probabilité :

$$p_{ij}^{m} = \frac{[\tau_{ij}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_j} \right]^{\beta} \cdot [d_j]^{\delta}}{\sum\limits_{l \notin tabou_m} [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot [d_l]^{\delta}} \tag{8.2}$$

En plus de la trace de phéromone, la règle de transition proposée utilise deux termes de visibilité : le premier terme favorise le choix de classes de voitures qui engendrent le moins de nouveaux conflits (n_cf) alors que le deuxième terme favorise les classes de voitures les plus difficiles. Le concept de difficulté d'une classe (d_v) a été proposé par J. Gottlieb *et al.* [GOT 03], et il est basé sur la notion de taux d'utilisation d'une option définie par B. Smith [SMI 97]. On définit le taux d'utilisation (tu_o) d'une option o par le ratio entre le nombre de voitures possédant cette option (nb_o) et le nombre maximum de voitures pouvant posséder cette option pour satisfaire la contrainte de capacité. Mathématiquement, le taux d'utilisation de l'option o est calculé selon l'équation (8.3) :

$$tu_o = \frac{nb_o}{\lceil nc \cdot r_o / s_o \rceil} \tag{8.3}$$

La difficulté d_v d'une classe v est alors calculée à l'aide de l'équation (8.4) :

$$d_v = \sum_{o=1}^{O} b_{vo} \cdot tu_o \tag{8.4}$$

Les exposants permettant de donner l'importance relative à la trace de phéromone et aux deux termes de visibilité sont respectivement α, β, δ. On note également qu'une

liste de *cl* classes de voitures candidates est utilisée dans la règle de transition, afin de réduire le temps de calcul en limitant le choix de la classe j pouvant être sélectionnée par une fourmi. De plus, on s'assure que toute séquence produite par une fourmi respecte les contraintes de production. Pour ce faire, chaque fourmi m utilise une liste *tabou_m* contenant les classes pour lesquelles il ne reste plus de voitures à placer. Des détails supplémentaires au sujet de la règle de transition sont disponibles dans l'article original [GRA 05].

Suite au choix d'une classe j à la suite d'une classe i par une fourmi, une mise à jour locale de la trace de phéromone pour l'élément $\tau_{i,j}$ est réalisée de manière à diminuer légèrement la quantité de phéromone (équation (8.5)). Ceci permet d'éviter une trop grande répétition du même choix de classes de voitures et favorise ainsi une diversification dans le processus de recherche. Le paramètre ρ_ℓ représente le degré de persistance locale de la trace de phéromone.

$$\tau_{i,j} = \rho_\ell \cdot \tau_{i,j} + (1 - \rho_\ell) \cdot \tau_0 \tag{8.5}$$

A la fin d'un cycle, une mise à jour globale de la trace de phéromone est réalisée par une seule fourmi : la meilleure solution du cycle (Sol^{Bc}). Telle que présentée par l'équation (8.6), une évaporation est d'abord effectuée sur l'ensemble des éléments de la matrice τ, selon un degré ρ_g de persistance globale de la trace. Ensuite, la phéromone est ajoutée pour chaque paire de classes de voitures ij appartenant à Sol^{Bc}. L'augmentation est proportionnelle à la qualité de solution L^{Bc} et au nombre x de fois que la paire ij apparaît dans la solution. Contrairement au problème de VC, qui consiste à la formation d'un tour hamiltonien, le problème de POV avec le regroupement des voitures identiques en classes entraîne inévitablement la répétition des numéros de classes dans une solution réalisable. Ceci explique pourquoi la répétition des motifs est prise en compte dans la mise à jour globale de la trace de phéromone :

$$\tau_{i,j} = \rho_g \cdot \tau_{i,j} + (1 - \rho_g) \cdot \Delta_{i,j}^{Bc} \text{ où } \Delta_{i,j}^{Bc} = \begin{cases} x \cdot \frac{1}{L^{Bc}} & \text{si } ij \in Sol^{Bc} \\ 0 & \text{sinon} \end{cases} \tag{8.6}$$

En général, les valeurs attribuées aux différents paramètres de ACS-2D suivent les recommandations de M. Dorigo et L.M. Gambardella [DOR 97]. Dans les autres cas, les paramètres ont été fixés suite à des expérimentations numériques et à des analyses de sensibilité. Le lecteur peut consulter M. Gravel *et al.* [GRA 05] pour les résultats obtenus à l'aide de cet algorithme, pour les trois ensembles de problèmes tests disponibles sur Internet à l'adresse http://www.csplib.org/. Ces travaux ont mené au développement d'une version de ACS pour le problème industriel, tel que formulé par le constructeur automobile français Renault dans le cadre du Challenge ROADEF 2005. La présentation de ces travaux fait l'objet de la prochaine section.

Parallèlement à ces travaux, nous avons poursuivi notre réflexion pour tenter d'incorporer à l'algorithme ACS une plus grande information sur les spécificités de ce problème [GAG 08, MOR 09]. La trace de phéromone utilisée précédemment est structurée de manière à représenter l'intérêt de placer une voiture de classe i adjacente à une voiture de classe j. Cependant, en raison des contraintes de capacité r/s des options qui s'étendent sur plusieurs positions, le choix d'une classe de voitures est influencé, non seulement par la classe de voitures adjacente, mais aussi par les autres classes à proximité. En effet, une classe de voitures peut engendrer un conflit avec les s_{max} positions qui lui sont adjacentes, où s_{max} est l'étendue la plus longue parmi les O contraintes de capacité de type r/s. Pour le POV, il semble donc opportun d'évaluer le bénéfice de placer une classe de voitures en considérant son interaction, non seulement avec la classe qui lui est adjacente, mais aussi avec l'ensemble des classes situées à proximité.

Un autre élément à considérer concerne le nombre de positions séparant les classes de voitures. Toujours en raison des contraintes de capacité r/s des options qui composent les classes de voitures, il peut être favorable de placer deux classes à une certaine distance les séparant l'une de l'autre et totalement désavantageux de les placer à proximité. Pour cette raison, une dimension supplémentaire est ajoutée à la matrice de trace de phéromone τ, afin de différencier l'intérêt de placer deux classes de voitures selon la distance qui les sépare. Pour ce faire, la matrice de trace de phéromone τ' est de taille $V \times V \times s_{max}$, où $\tau'_{i,j,dist}$ représente l'intérêt de placer deux classes de voitures i et j à une distance de $dist$ positions l'une de l'autre. Cette nouvelle structure de trace de phéromone est appelée 3D, et les modifications apportées à l'ACS sont expliquées dans les paragraphes qui suivent. Elles touchent l'utilisation de τ' dans la règle de transition, la mise à jour locale et la mise à jour globale de la trace de phéromone.

L'utilisation de la trace de phéromone dans la règle de transition (équations (8.1) et (8.2)) est modifiée de façon à refléter l'intérêt de placer en position pos une classe candidate j à proximité des dernières classes de voitures de la séquence. Comme illustré par l'équation (8.7), on utilise maintenant une somme des traces de phéromone accumulées entre cette classe candidate j et chacune des classes i des s_{max} positions précédentes, en fonction de la distance qui les sépare. Cette somme mise à l'exposant α vient remplacer le terme $[\tau_{i,j}]$ dans la règle de transition présentée précédemment.

$$\left[\sum_{k=pos-s_{max}}^{pos-1} \tau'_{i,j,pos-k} \right]^{\alpha} \tag{8.7}$$

Une fois la classe de voitures choisie à l'aide de la règle de transition, une mise à jour locale est effectuée selon l'équation (8.8), entre la classe j placée en position pos et chacune des classes i des s_{max} positions précédentes. La modification implique

donc un total de s_{max} mises à jour locales de la trace :

$$\begin{array}{l}\text{POUR chaque classe } i \text{ en position } k \text{ entre } (pos - s_{max}) \text{ et } (pos\text{-}1) \text{ FAIRE} \\ \quad \tau'_{i,j,pos-k} = \rho_\ell \cdot \tau'_{i,j,pos-k} + (1 - \rho_\ell) \cdot \tau_0\end{array} \tag{8.8}$$

Comme dans l'algorithme ACS-2D présenté précédemment, la meilleure solution du cycle (Sol^{Bc}) participe à la mise à jour globale de la trace à la fin d'un cycle. De plus, la mise à jour globale de la trace de phéromone 3D se fait maintenant entre une classe de voitures i à une position y donnée et chacune des classes j se trouvant aux s_{max} positions suivantes. L'équation (8.9) présente la modification apportée à la mise à jour globale :

$$\begin{array}{l}\text{Evaporation de l'ensemble de la matrice } \tau' \text{ selon } \rho_g \\ \text{POUR chaque classe } i \text{ en position } k \text{ de la solution } S^{Bc} \text{ FAIRE} \\ \quad \text{POUR chaque classe } j \text{ en position } k' \text{ entre } (k{+}1) \text{ et } (k{+}s_{max}) \text{ FAIRE} \\ \quad\quad \tau'_{i,j,k'-k} = \tau'_{i,j,k'-k} + (1 - \rho_g) \cdot \frac{1}{L^{Bc}}\end{array} \tag{8.9}$$

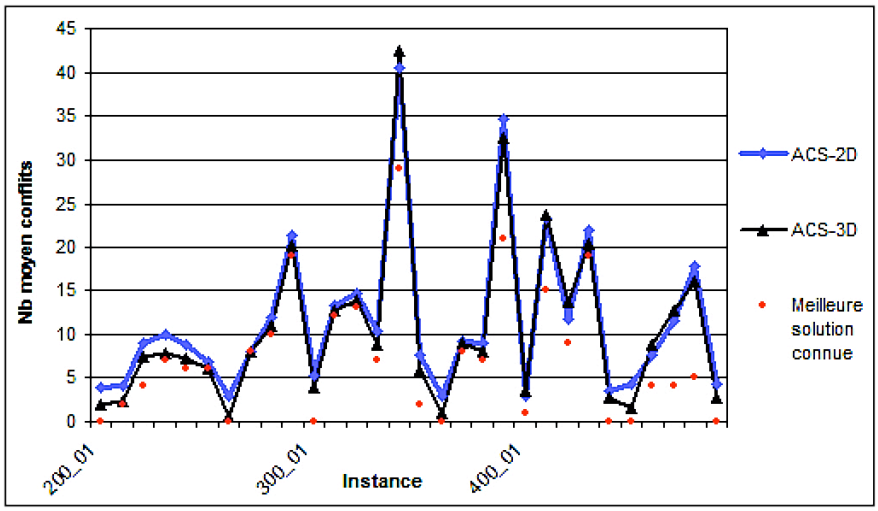


Figure 8.3. *Comparaison de la performance entre ACS-2D et ACS-3D pour le troisième ensemble de problèmes tests*

L'efficacité de cette nouvelle trace de phéromone spécialisée pour résoudre le POV a été démontrée par des essais numériques. En considérant le troisième ensemble de problèmes tests disponibles sur Internet qui regroupe les instances de plus grande taille, la figure 8.3 présente une comparaison de la performance entre ACS-2D et ACS avec la nouvelle structure de trace spécialisée (ACS-3D). Pour cette comparaison basée sur le nombre moyen de conflits, chaque instance de problèmes a été résolue à

cent reprises. Des tests statistiques ont confirmé la supériorité de ACS-3D par rapport à ACS-2D pour chaque groupe de problèmes (200 voitures, 300 voitures et 400 voitures). Au moment de la réalisation de ces travaux, ACS-3D a ainsi permis d'obtenir les meilleurs résultats publiés de la littérature. Afin de tirer avantage de cette trace de phéromone spécialisée, le paramètre α contrôlant son importance dans la règle de transition a toutefois dû être augmenté. Ceci va à l'encontre de la tendance observée dans la littérature, qui consiste à donner une valeur de 1 à α ou tout simplement à retirer ce paramètre de la règle de transition. Le lecteur peut consulter S. Morin *et al.* [MOR 09] et C. Gagné *et al.* [GAG 08] pour plus de détails.

8.3. Résolution du problème industriel d'ordonnancement d'une chaîne d'assemblage automobile à l'aide de l'OCF

La réalité de l'industrie automobile ne tient pas uniquement compte des contraintes de l'atelier de montage. En effet, dans la formulation du problème industriel proposée par le constructeur automobile Renault dans le cadre du Challenge ROADEF 2005 [NGU 05], les contraintes de l'atelier de peinture sont également prises en compte. Dans cet atelier, la minimisation de la consommation de solvant utilisé pour nettoyer les pistolets de peinture à chaque changement de couleur est un objectif important à considérer. De plus, une trop longue séquence de voitures de couleurs identiques rend difficile le contrôle visuel de la qualité. De ce fait, le nombre de voitures consécutives possédant une couleur identique ne peut dépasser un nombre maximal donné (raf_{max}). Dans cet atelier, l'objectif visé est donc de minimiser le nombre de changements de couleur (COULEUR).

En plus des contraintes de l'atelier de peinture, la formulation industrielle du POV subdivise les contraintes de capacité de l'atelier de montage en deux types : les contraintes de capacité pour les options prioritaires et les contraintes de capacité pour les options non prioritaires. On cherche ainsi à minimiser le nombre de conflits pour les options prioritaires (OP) et pour les options non prioritaires (ONP). Le problème industriel est donc un problème de nature multi-objectif où trois objectifs contradictoires sont à optimiser. Dans la présentation du Challenge ROADEF 2005, le constructeur automobile Renault propose de traiter les objectifs du problème selon les trois ordres de priorité suivants : OP-COULEUR-ONP, OP-ONP-COULEUR et COULEUR-OP-ONP. De plus, le traitement des objectifs est fait de manière lexicographique, selon l'équation (8.10), en accordant des pondérations $w1$, $w2$ et $w3$ à chacun d'eux. Ces pondérations sont respectivement de 1 000 000, 1 000 et 1.

$$F(Y) = w1 \cdot obj_1 + w2 \cdot obj_2 + w3 \cdot obj_3 \tag{8.10}$$

```
/* Initialisation */
Initialiser la matrice de trace de phéromone τ à une petite valeur τ0 pour chaque paire de classes ij ;
/* Boucle principale */
POUR temps < tMax FAIRE
    /* Construire une séquence pour chaque fourmi */
    POUR pos = 1 à NC FAIRE
        POUR m = 1 à nb_ants FAIRE
            CAS 1 : /* OP-COULEUR-ONP */
            Utiliser la règle de transition décrite à la procédure 1 (figure 8.5)
            CAS 2 : /* COULEUR-OP-ONP */
            Utiliser la règle de transition décrite à la procédure 2 (figure 8.6)
            CAS 3 : /* OP-ONP-COULEUR */
            Utiliser la règle de transition décrite à la procédure 3 (figure 8.7)
            Effectuer la mise à jour locale de la trace de phéromone pour (i, j) ;
    POUR m = 1 à nb_ants FAIRE
        Evaluer la solution de chaque fourmi m ;
    Appliquer des méthodes de recherche locale sur la meilleure solution du cycle Sol^Bc
    Réaliser la mise à jour globale de la trace de phéromone à l'aide de la meilleure solution du cycle Sol^Bc
    Mettre à jour la meilleure solution connue Sol^Bg
```

Figure 8.4. *Pseudo-code de ACS pour le POV+*

Une autre caractéristique du problème industriel d'ordonnancement de voitures (POV+) réside dans le fait qu'il faut faire le lien entre les différentes journées de production. Ainsi, il faut évaluer la solution en considérant les voitures déjà planifiées à la journée précédente et en extrapolant le nombre minimum de conflits générés avec la journée suivante. De même, on ajoute un changement de couleur si la première voiture de la journée courante est de couleur différente de la dernière voiture de la journée précédente. C'est dans ce contexte que nous avons proposé un ACS comme méthode de solution.

Pour le POV+, une séquence de production est notée :

$$Y = \{ClassesOP/ClassesONP/Couleurs\},$$

et les éléments situés à la position k de la séquence sont définis par :

$$Y(k) = ClassesOP(k)/ClassesONP(k)/Couleurs(k).$$

Le vecteur $ClassesOP$ permet l'évaluation de la solution sur l'objectif OP, le vecteur $ClassesONP$ sert à l'évaluation de la solution sur l'objectif ONP alors que le vecteur $Couleurs$ permet de déterminer la valeur de l'objectif COULEUR. Pour l'évaluation des objectifs OP et ONP, le constructeur automobile Renault a proposé une

méthode qui diffère du problème théorique présenté à la section précédente. En effet, le nombre de conflits dans une sous-séquence de longueur s représente l'écart entre le nombre de voitures possédant l'option et le nombre maximum r de voitures pouvant posséder cette option. Dans le problème théorique, le nombre de conflits était 1 si le nombre de voitures possédant l'option dépasse r et 0 autrement. Par ce changement au processus d'évaluation, le constructeur vise à disperser davantage les conflits dans la séquence.

Priorité des objectifs /* OP-COULEUR-ONP */

En cours de construction de la séquence, on peut se retrouver dans l'un des deux cas suivants :

- si la rafale maximale de couleur n'est pas atteinte, c'est-à-dire qu'il pourrait être possible d'ajouter une voiture de la même couleur que la dernière voiture placée, on doit chercher à favoriser les voitures de cette couleur en utilisant les étapes 1a), 1b), 1c) et 1d) de la procédure ;
- si la séquence oblige de faire une purge parce que l'on a atteint la rafale maximale, on utilise les étapes 1a), 1c) et 1e) de la procédure.

Etape 1 : sélection de $ClassesOP(k)$ et de $Couleurs(k)$

a) S'il existe des classes prioritaires pour lesquelles il reste des voitures à placer et qui ne génèrent pas de nouveaux conflits, alors seules celles-ci sont candidates. Sinon, toutes les classes prioritaires, pour lesquelles il reste des voitures à placer, sont candidates. Exclure les classes prioritaires qui possèdent seulement la couleur de la dernière voiture placée si l'on a atteint la rafale maximale avec la dernière voiture placée.

b) Attribuer un bonus aux classes prioritaires candidates possédant des voitures de la couleur permettant de continuer la rafale.

c) Faire le choix de $ClassesOP(k)$ par la règle de transition suivante :

$$j = \begin{cases} \arg\max_{l \notin tabou_m} \left\{ [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{OP} \right]^{\delta} \cdot b\_c \right\} & \text{si } q \leq q_0 \\ J & \text{si } q > q_0 \end{cases}$$

où J est choisie selon la probabilité :

$$p_{ij}^{m} = \frac{[\tau_{ij}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_j} \right]^{\beta} \cdot \left[d_j^{OP} \right]^{\delta} \cdot b\_c}{\sum_{l \notin tabou_m} [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{OP} \right]^{\delta} \cdot b\_c}$$

d) Si la classe prioritaire choisie possède des voitures de la couleur permettant de continuer la rafale en cours, on sélectionne automatiquement cette couleur dans $Couleurs(k)$. Sinon, aller à l'étape 1e).

e) Sélectionner une couleur aléatoirement parmi les couleurs disponibles dans la classe prioritaire choisie à l'exception de la couleur de la dernière voiture placée. Mettre cette couleur dans $Couleurs(k)$.

Etape 2 : sélection de $ClassesONP(k)$ selon la procédure 4

Figure 8.5. *Procédure 1*

L'algorithme ACS proposé pour solutionner le POV+ [GAG 06] est inspiré de ACS-2D décrit à la section précédente. Toutefois, la construction d'une solution Y à l'aide de la règle de transition dépend de l'ordre de priorité des objectifs. La figure 8.4

résume le fonctionnement général de ACS, qui incorpore des matrices de visibilité multiples dans le processus de recherche [GAG 02], ainsi que les améliorations proposées à la règle de transition et à la liste de candidats par Dorigo et Gambardella [DOR 97]. Il faut également noter qu'il existe une limite de temps (t_{Max}) imposée par le Challenge pour l'exécution de l'algorithme.

La procédure 1, présentée à la figure 8.5, est utilisée pour la construction de la séquence dans le cas où les objectifs sont priorisés dans l'ordre OP-COULEUR-ONP. La séquence de voitures est alors construite en cherchant principalement à minimiser le nombre de violations des contraintes de capacité liées aux options prioritaires (OP). Toutefois, au moment de la sélection de $ClassesOP(k)$, la couleur de la voiture ($Couleurs(k)$) est également déterminée par la règle de transition pour prendre en considération l'objectif de minimisation du nombre de changements de couleur. Pour le troisième objectif, la sélection de $ClassesONP(k)$ est considérée seulement lorsque la séquence complète est obtenue, et la procédure 4, présentée à la figure 8.8, est utilisée à cette fin.

La procédure 2, présentée à la figure 8.6, est utilisée pour la construction de la séquence dans le cas où les objectifs sont priorisés dans l'ordre COULEUR-OP-ONP. La séquence de voitures est construite en cherchant principalement à minimiser le nombre de changements de couleur. Cet objectif est toutefois facile à atteindre en alignant consécutivement des voitures de même couleur jusqu'à l'obtention de la rafale maximale (raf_{max}) ou jusqu'à l'épuisement des voitures d'une couleur donnée et en alternant les différentes couleurs. Pour cette raison, lors de la sélection de la couleur de la voiture ($Couleurs(k)$), la règle de transition détermine au même moment $ClassesOP(k)$, en considérant l'objectif de minimisation des violations des contraintes de capacité liées aux options prioritaires (OP). Pour le troisième objectif, la sélection de $ClassesONP(k)$ est considérée seulement lorsque la séquence complète est obtenue, et encore une fois, la procédure 4 est utilisée à cette fin.

La procédure 3, présentée à la figure 8.7, est utilisée pour la construction de la séquence dans le cas où les objectifs sont priorisés dans l'ordre OP-ONP-COULEUR. La séquence de voitures est alors construite en cherchant principalement à minimiser le nombre de violations des contraintes de capacité liées aux options prioritaires (OP). Une fois $ClassesOP(k)$ sélectionnée, $ClassesONP(k)$ est à son tour choisie selon des considérations similaires. Par la suite, une couleur ($Couleurs(k)$) est associée à la voiture dans l'optique d'une rafale la plus longue possible, tout en respectant la rafale maximale.

Les règles de transition utilisées dans les procédures 1, 2 et 3 peuvent être décrites comme suit : soit ij la trace accumulée entre les paires de classes prioritaires i et j, n_cf_j le nombre de nouveaux conflits générés en plaçant une voiture de classe j dans la prochaine position de la séquence, d_j un indicateur du niveau global de difficulté de la classe j (équation (8.4)), (b_c) un paramètre dont la valeur est égale à 1 et qui peut

être augmenté, dans certains cas, de façon à favoriser certaines couleurs (voir procédure 1, étape 1b), les paramètres α, β et δ exprimant l'influence relative de la trace et des différentes visibilités du problème. Alors, la règle de transition est utilisée par une fourmi m pour choisir la classe j qui suivra la classe i dans la séquence ($j \notin tabou_m$, c'est-à-dire j est une classe pour laquelle il reste au moins une voiture à placer). Notons également que q est un nombre aléatoire, et q_0 est un paramètre entre 0 et 1. Le paramètre q_0 détermine l'importance relative de l'exploitation de l'information existante et l'exploration vers de nouvelles solutions.

Priorité des objectifs /* COULEUR-OP-ONP */

En cours de construction de la séquence, on peut se retrouver dans l'un des deux cas suivants :

- si la rafale maximale de couleur n'est pas atteinte et qu'il reste des voitures de la couleur permettant de continuer la rafale en cours, on continue obligatoirement la rafale de couleur en cours en utilisant les étapes 1a) et 1c) de la procédure ;
- si la rafale ne peut être poursuivie, parce que la rafale maximale est atteinte ou parce qu'il ne reste plus aucune voiture de la couleur permettant de continuer la rafale en cours, on utilise les étapes 1b), 1c) et 1d) de la procédure.

Etape 1 : sélection de *Couleurs*(k) et *ClassesOP*(k)

a) S'il existe des classes prioritaires pour lesquelles il reste des voitures de la couleur permettant de continuer la rafale en cours et qui ne génèrent pas de nouveaux conflits, alors seules celles-ci sont candidates. Sinon, toutes les classes prioritaires, pour lesquelles il reste des voitures de la couleur permettant de continuer la rafale en cours, sont candidates.

b) S'il existe des classes prioritaires pour lesquelles il reste des voitures à placer et qui ne génèrent pas de nouveaux conflits, alors seules celles-ci sont candidates. Sinon, toutes les classes prioritaires, pour lesquelles il reste des voitures à placer, sont candidates. Exclure les classes prioritaires qui possèdent seulement la couleur de la dernière voiture placée si l'on a atteint la rafale maximale avec la dernière voiture placée.

c) Faire le choix de *ClassesOP*(k) par la règle de transition suivante :

$$j = \begin{cases} \arg\max_{l \notin tabou_m} \left\{ [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{OP} \right]^{\delta} \right\} & \text{si } q \le q_0 \\ J & \text{si } q > q_0 \end{cases}$$

où J est choisie selon la probabilité :

$$p_{ij}^{m} = \frac{[\tau_{ij}]^{\alpha} \left[\frac{1}{1+n\_cf_j} \right]^{\beta} \cdot \left[d_j^{OP} \right]^{\delta}}{\sum_{l \notin tabou_m} [\tau_{il}]^{\alpha} \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{OP} \right]^{\delta}}$$

d) Sélectionner une couleur aléatoirement parmi les couleurs disponibles dans la classe prioritaire choisie, à l'exception de la couleur de la dernière voiture placée. Mettre cette couleur dans *Couleurs*(k).

Etape 2 : sélection de *ClassesONP*(k) selon la procédure 4

Figure 8.6. *Procédure 2*

Priorité des objectifs /* OP-ONP-COULEUR */

En cours de construction de la séquence, on peut se retrouver dans l'un des deux cas suivants :

- si la rafale maximale de couleur n'est pas atteinte, c'est-à-dire qu'il pourrait être possible d'ajouter une voiture de même couleur que la dernière voiture placée, on doit chercher à favoriser les voitures de cette couleur en utilisant les étapes 1a), 1b), 1c), 1d), 1e) et 1f) de la procédure ;
- si la séquence oblige de faire une purge parce que l'on a atteint la rafale maximale,on utilise les étapes a), b), c), d) et f) de la procédure.

Etape 1 : sélection de $ClassesOP(k)$, $ClassesONP(k)$ et $Couleurs(k)$

a) S'il existe des classes prioritaires pour lesquelles il reste des voitures à placer et qui ne génèrent pas de nouveaux conflits, alors seules celles-ci sont candidates. Sinon, toutes les classes prioritaires, pour lesquelles il reste des voitures à placer, sont candidates.

b) Faire le choix de $ClassesOP(k)$ par la règle de transition suivante :

$$j = \begin{cases} \arg \max_{l \notin tabou_m} \left\{ [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{OP} \right]^{\delta} \right\} & \text{si } q \leq q_0 \\ J & \text{si } q > q_0 \end{cases}$$

où J est choisie selon la probabilité :

$$p_{ij}^m = \frac{[\tau_{ij}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_j} \right]^{\beta} \cdot \left[d_j^{OP} \right]^{\delta}}{\sum_{l \notin tabou_m} [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{OP} \right]^{\delta}}$$

c) Si la classe prioritaire choisie possède des voitures de classes non prioritaires ne générant pas de nouveaux conflits sur l'objectif ONP, celles-ci sont les classes candidates. Sinon, toutes les classes ONP, pour lesquelles il reste des voitures à placer, sont candidates.

d) Faire le choix de $ClassesONP(k)$ par la règle de transition suivante :

$$p_{ij}^m = \frac{[\tau_{ij}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_j} \right]^{\beta} \cdot \left[d_j^{ONP} \right]^{\delta}}{\sum_{l \notin tabou_m} [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{ONP} \right]^{\delta}}$$

e) Si les classes prioritaires et non prioritaires choisies possèdent des voitures de la couleur permettant de continuer la rafale en cours, on sélectionne automatiquement cette couleur dans $Couleurs(k)$. Sinon, aller à l'étape 1f).

f) Sélectionner une couleur aléatoirement parmi les couleurs disponibles dans les classes prioritaires et non prioritaires choisies, à l'exception de la couleur de la dernière voiture placée. Mettre cette couleur dans $Couleurs(k)$.

Figure 8.7. *Procédure 3*

La procédure 4 présentée à la figure 8.8 permet, quant à elle, d'attribuer une classe non prioritaire à chacune des voitures de la séquence dans le cas où les objectifs sont traités dans l'ordre OP-COULEUR-ONP et COULEUR-OP-ONP. Par économie de temps, cette étape est réalisée seulement sur la meilleure solution du cycle évaluée selon les deux premiers objectifs. Le choix de la classe ONP se fait de manière déterministe ($q \leq q_0$) ou probabiliste ($q > q_0$).

Sélection de $ClassesONP(k)$

En parcourant la séquence du début vers la fin, on sélectionne, pour chaque position k dont la classe prioritaire et la couleur sont à ce moment connues, la classe non prioritaire à affecter parmi les classes non prioritaires disponibles. Cette sélection considère alors le nombre de nouveaux conflits générés sur l'objectif ONP en plaçant une voiture de classe j dans cette position de la séquence et un indicateur du niveau global de difficulté de la classe non prioritaire j de la manière suivante :

$$j = \begin{cases} \arg\max_{l \notin tabou_m} \left\{ [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{ONP} \right]^{\delta} \right\} & \text{si } q \leq q_0 \\ J & \text{si } q > q_0 \end{cases}$$

où J est choisie selon la probabilité :

$$p_{ij}^m = \frac{[\tau_{ij}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_j} \right]^{\beta} \cdot \left[d_j^{ONP} \right]^{\delta}}{\sum_{l \notin tabou_m} [\tau_{il}]^{\alpha} \cdot \left[\frac{1}{1+n\_cf_l} \right]^{\beta} \cdot \left[d_l^{ONP} \right]^{\delta}}$$

Figure 8.8. *Procédure 4*

Une mise à jour locale, c'est-à-dire une diminution de la trace après le choix (ij), est réalisée selon l'équation (8.5), pour éviter que les fourmis répètent les mêmes choix et pour accroître la diversité dans le processus de recherche.

Les organisateurs du Challenge ROADEF 2005 ont fourni seize instances (ensemble A) à tous les participants pour aider au développement de leurs applications pendant la phase préliminaire. Celles-ci comportent entre 334 et 1 314 voitures à ordonnancer, possèdent entre six et 22 options (prioritaires et non prioritaires) et entre onze et 24 couleurs différentes. L'évaluation comparative de la performance des applications a également été établie par les organisateurs à partir de ces instances pour déterminer les 18 équipes qualifiées pour la phase suivante. Le tableau 8.4 résume les principaux résultats de cette phase du Challenge ROADEF 2005, qui démontrent que ACS s'est avérée une méthode particulièrement compétitive.

Dans la première colonne du tableau 8.4, on retrouve la liste des seize instances proposées dans cette phase. Celles-ci sont réparties selon l'ordre de priorité des objectifs et la difficulté estimée de l'instance par les organisateurs. La deuxième colonne présente le meilleur résultat obtenu par l'ensemble des équipes sur la fonction agrégative (équation (8.10)). Il s'agit de la valeur minimale obtenue parmi les cinq essais lors du test réalisé par les organisateurs pour l'ensemble des applications. La colonne 3 précise la valeur obtenue sur chacun des objectifs pour le meilleur résultat. La colonne 4 fournit le rang obtenu par ACS pour chacune des instances, ainsi que le rang pour chaque groupe d'instances. La colonne 5 précise le nombre d'équipes ayant réussi à trouver la valeur minimale sur l'objectif principal parmi les cinq essais, lors du test réalisé par les organisateurs, tandis que la colonne 6 précise, à l'aide d'un X, si ACS fait partie de celles-ci. Finalement, les deux dernières colonnes présentent les

meilleurs résultats obtenus lors de la phase finale pour l'ensemble A, ainsi que le rang obtenu par ACS.

Ensemble A	Phase de qualification					Phase finale	
	Meilleurs Résultats	Solution	Rang ACS	Nombre d'équipes	ACS	Meilleurs Résultats	Rang ACS
Instances faciles							
OP_COULEUR_ONP							
022_3_4	31000	(0,31,0)	8	24	X	31001	10
025_38_1	233624	(0,230,3624)	9	24	X	231452	6
064_38_2_ch1	112761	(0,112,761)	17	24	X	112759	7
064_38_2_ch2	34051	(0,34,51)	7	24	X	34051	9
OP_ONP_COULEUR							
025_38_1	129743	(0,121,794)	22	24	X	99720	12
			14				
Instances difficiles							
OP_COULEUR_ONP							
024_38_3	4266155	(4,266,155)	8	12	X	4249083	12
024_38_5	4289131	(4,289,131)	9	13	X	4280079	12
039_38_4_ch1	13137000	(13,137,0)	6	4	X	13129000	11
048_39_1	179642	(0,179,642)	9	16	X	175615	10
OP_ONP_COULEUR							
024_38_3	4009357	(4,357,9)	6	16	X	4000306	9
024_38_5	4041365	(4,41,365)	6	13	X	4034309	11
048_39_1	64351	(0,64,317)	7	11	X	61290	9
			2				
COULEUR_OP_ONP							
022_3_4	11039001	(11,39,1)	4	24	X	11039001	10
039_38_4_ch1	68156000	(68,156,0)	6	19	X	68161000	8
064_38_2_ch1	63423782	(63,423,782)	7	12	X	63423782	7
064_38_2_ch2	27367052	(27,367,52)	11	20	X	27367052	14
			2				
			8				7

Tableau 8.4. *Résultats de la phase de qualification du Challenge ROADEF 2005*

Au niveau des résultats obtenus lors de la phase de qualification, ACS obtient le 14e rang sur les instances faciles, dont l'objectif principal est OP, le 2e rang sur les instances difficiles, dont l'objectif principal est OP et le 2e rang pour les instances dont l'objectif principal est COULEUR. Globalement, ACS se classe au 8e rang sur les 24

équipes ayant remis une application à la date prévue pour la phase de qualification du Challenge. Au départ, 55 équipes étaient inscrites au Challenge.

On note particulièrement l'efficacité de ACS à optimiser le premier objectif. En effet, seulement trois équipes ont réussi à trouver, pour toutes les instances, la valeur minimale sur l'objectif principal parmi les cinq essais, lors du test réalisé par les organisateurs. Notre algorithme, basé sur ACS fait partie de ces trois équipes. Toutefois, pour obtenir de tels résultats, nous avons accordé une très grande importance aux éléments de visibilité par rapport à la trace de phéromone dans la règle de transition. Nous exploitons ainsi fortement la capacité de construction de la métaheuristique pour optimiser l'objectif principal. De ce fait, pour les instances faciles, on note alors une lacune pour explorer efficacement l'espace de recherche et obtenir de bons résultats sur les objectifs secondaires. Il manque alors un mécanisme de diversification permettant d'atteindre de nouvelles régions de l'espace de solutions.

Dans la phase finale, les équipes qualifiées pouvaient apporter des modifications à leurs algorithmes et disposaient également de 45 nouvelles instances (ensemble B) pour tester leurs applications. Il faut toutefois noter que toutes les équipes connaissaient l'espace de recherche pour les seize instances de l'ensemble A par la diffusion des solutions obtenues par toutes les équipes à la phase de qualification. Le contexte de conception et de mise au point d'une approche de solution modifiée est alors très différent de la phase de qualification. Dans notre cas, nous avons créé une métaheuristique hybride à relais en ajoutant un Tabou/VNS pour réaliser le mécanisme de diversification après l'exécution de ACS.

L'avant-dernière colonne du tableau 8.4 indique le meilleur résultat obtenu par les seize équipes qualifiées à l'aide de la version modifiée de leurs algorithmes pour les instances de l'ensemble A. On note alors une amélioration importante de la qualité des résultats pour plusieurs instances, qui démontre que les équipes ont tiré profit de la connaissance de l'espace de solutions dans la mise au point de leur nouvelle version d'algorithme. Pour sa part, la dernière colonne du tableau 8.4 précise le rang obtenu par notre algorithme hybride pour chacune de ces instances. On note que le Tabou/VNS a réalisé le mécanisme de diversification escompté, qui a permis l'obtention d'un meilleur classement sur les instances faciles. Toutefois, le rang global obtenu avec cette nouvelle version (7^{e} rang sur les 18 équipes finalistes) n'est pas très différent de celui de la phase de qualification.

Le classement final du Challenge ROADEF 2005 a été établi à partir des 19 instances de l'ensemble X et notre algorithme a finalement obtenu le 11^{e} rang.

8.4. Conclusion

Dans ce chapitre, nous avons présenté la démarche utilisée pour la conception d'un OCF afin de traiter le cas de l'ordonnancement industriel d'une chaîne d'assemblage.

Tout d'abord, nous avons montré à travers nos différents travaux que l'OCF est une métaheuristique très efficace pour solutionner le POV théorique. Les résultats obtenus sur les instances de problèmes disponibles sur Internet ont été considérés, jusqu'à tout récemment, comme les meilleurs résultats connus dans la littérature. On peut également constater, à travers les résultats obtenus à la phase de qualification du Challenge ROADEF, que l'OCF est une approche très compétitive pour solutionner le POV+. La plupart des autres équipes finalistes, dont les travaux ont été publiés, ont utilisé principalement des méthodes de recherche dans le voisinage, comme le recuit simulé, la recherche avec tabous, la recherche locale et la recherche avec voisinages variables [BEN 08, BRI 08, COR 08, EST 08, GAV 08, RIB 08]. On a pu constater la force de l'ACS pour la construction de très bonnes solutions sur l'objectif principal, pour des instances de grande taille, pouvant atteindre plus de 1 000 voitures. Toutefois, le fait de mettre l'accent sur une dimension particulière dans la construction des solutions fait en sorte que l'OCF produit de moins bons résultats, sur les instances faciles.

Dans la présentation de cette application, nous avons également voulu montrer que l'OCF est une métaheuristique qui doit, en tout temps, être adaptée pour incorporer les éléments spécifiques du problème traité, afin d'obtenir une performance intéressante. En particulier, l'utilisation de plusieurs termes de visibilité pour guider le processus de construction des solutions, ainsi que l'adaptation de la structure de trace de phéromone dans la règle de transition, permettent d'illustrer le fait qu'il ne faut pas s'en remettre aux versions de base publiées dans la littérature.

D'autre part, le traitement des objectifs de manière lexicographique, tel que proposé par Renault dans le cadre du Challenge, fait en sorte que plusieurs solutions *intéressantes* pour l'entreprise sont ignorées. En effet, le fait de relâcher l'importance accordée à l'objectif principal peut permettre de mettre en évidence des solutions beaucoup moins coûteuses pour l'entreprise. Nous estimons donc que l'obtention de solutions de compromis, telle que nous l'avons présentée au chapitre 7, serait tout à fait appropriée. Dans les travaux futurs, nous expérimenterons ce type d'approche pour trouver un ensemble de solutions de compromis qui pourront être comparées à la solution trouvée, en considérant les objectifs par ordre lexicographique. Il sera sûrement possible de mettre en évidence des solutions beaucoup plus intéressantes financièrement pour une entreprise manufacturière et qui correspondent davantage à la réalité industrielle.

8.5. Bibliographie

[BEN 08] Benoist T., « Soft car sequencing with colors : Lower bounds and optimality proofs », *European Journal of Operational Research*, vol. 191, n° 3, p. 957–971, 2008.

[BRI 08] Briant O., Naddef D., Mounié M., « Greedy approach and multi-criteria simulated annealing for the car sequencing problem », *European Journal of Operational Research*, vol. 191, n° 3, p. 993–1003, 2008.

[BUL 99] Bullnheimer B., Hartl R., Strauss C., « An Improved Ant system Algorithm for the Vehicle Routing Problem », *Annals of Operations Research*, vol. 89, p. 319–328, 1999.

[COL 91] Colorni A., Dorigo M., Maniezzo V., « Distributed optimization by ant-colonies », F. Varela, P. Bourgine (dir.), *Proceedings of the First European Conference on Artificial Life (ECAL'91)*, p. 134–142, MIT Press, Cambridge, MA, 1991.

[COR 08] Cordeau J., Laporte G., Pasin F., « Iterated tabu search for the car sequencing problem », *European Journal of Operational Research*, vol. 191, n° 3, p. 945–956, 2008.

[DIN 88] Dincbas M., Simonis H., Hentenryck P. V., « Solving the car sequencing problem in constraint logic programming », *Proceedings of the European Conference on Artificial Intelligence (ECAI-88)*, p. 290-295, Pitmann Publishing, Londres, 1988.

[DOR 91] Dorigo M., Maniezzo V., Colorni A., Positive feedback as a search strategy, Rapport n° 91-016, Politecnico di Milano, Italie, 1991.

[DOR 92] Dorigo M., Optimization, learning and natural algorithms, PhD thesis, Politecnico di Milano, Italie, 1992.

[DOR 96] Dorigo M., Maniezzo V., Colorni A., « Ant system : optimization by a colony of cooperating agents », *IEEE Transactions on Systems, Man & Cybernetics*, vol. 26, n° 1, p. 29–41, 1996.

[DOR 97] Dorigo M., Gambardella L., « Ant Colony system : A Cooperative learning approach to the Traveling Salesman Problem », *IEEE Transactions on Evolutionary Computation*, vol. 1, n° 1, p. 53–66, 1997.

[DOR 04] Dorigo M., Stützle T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, 2004.

[EST 08] Estellon B., Gardi F., Nouioua K., « Two local search approaches for solving real-life car sequencing problem », *European Journal of Operational Research*, vol. 191, n° 3, p. 912–927, 2008.

[GAG 02] Gagné C., Gravel M., Price W., « Algorithme d'optimisation par colonie de fourmis avec matrices de visibilité multiples pour la résolution d'un problème d'ordonnancement industriel », *Information Systems and Operational Research (INFOR)*, vol. 40, n° 2, p. 259–276, 2002.

[GAG 06] Gagné C., Gravel M., Price W., « Solving real car sequencing problems with Ant Colony Optimization », *European Journal of Operational Research*, vol. 174, n° 3, p. 1427–1448, 2006.

[GAG 08] Gagné C., Gravel M., Morin S., Price W., « Impact of the pheromone trail on the performance of ACO algorithms for solving the car sequencing problem », *Journal of the Operational Research Society*, vol. 59, p. 1077–1090, 2008.

[GAV 08] Gavranovic H., « Local search and suffix tree for car-sequencing problem with colors », *European Journal of Operational Research*, vol. 191, n° 3, p. 972–980, 2008.

[GOT 03] Gottlieb J., Puchta M., Solnon C., « A study of greedy, local search and ant colony optimization approaches for car sequencing problems », *Computers Science*, p. 246–257, 2003.

[GRA 05] Gravel M., Gagné C., Price W., « Review and comparison of three methods for the solution of the car-sequencing problem », *Journal of the Operational Research Society*, vol. 56, n° 11, p. 1287–1295, 2005.

[KIS 04] Kis T., « On the complexity of the car sequencing problem », *Operations Research Letters*, vol. 32, n° 4, p. 331–336, 2004.

[MOR 09] Morin S., Gagné C., Gravel M., Price W., « Ant Colony Optimization with a specialized pheromone trail for the car sequencing problem », *European Journal of Operational Research*, vol. 197, n° 3, p. 1185–1191, 2009.

[NGU 05] Nguyen A., Cung V., « Le problème du Car Sequencing Renault et le challenge ROADEF' 2005 », *Proceedings of Journées Francophones de Programmation par Contraintes*, p. 3–10, 2005.

[PAR 86] Parello B., Kabat W., Wos L., « Job-shop scheduling using automated reasoning : A case study of the car-sequencing problem », *Journal of Automated Reasoning*, vol. 2, p. 1-42, 1986.

[PAR 88] Parello B., « CAR WARS : The (almost) birth of an expert system », *AI Expert*, vol. 3, p. 60–64, 1988.

[RIB 08] Ribeiro C., Aloise D., Noronha T., Rochaand C., Urrutia S., « An efficient implementation of a VNS/ILS heuristic for a real-life car sequencing problem », *European Journal of Operational Research*, vol. 191, n° 3, p. 595–611, 2008.

[SMI 97] Smith B., « Succeed-first or Fail-first : A Case Study in Variable and Value Ordering Heuristics », *Proceedings of the third international Conference on the Practical Applications of Constraint Technology*, p. 321–330, Londres, Royaume-Uni, 1997.

[SOL 08] Solnon C., Cung V., Nguyen A., Artigues C., « The car sequencing problem : overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem », *European Journal of Operational Research*, vol. 191, n° 3, p. 912–927, 2008.

[STU 97] Stützle T., Hoos H., « The MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems : Towards Adaptive Tools for Global Optimization », S. Voss, S. Martello, I. H. Ossmann, C. Roucairol (dir.), *Meta-Heuristic, Advances and Trends in Local Search Paradigma for Optimization*, p. 313–329, Kluwer Academic, 1997.

[STU 00] Stützle T., Hoos H., « MAX-MIN ant system », *Future Generation Computer Systems Journal*, vol. 16, n° 8, p. 889–914, 2000.

Chapitre 9

Algorithme de fourmis pour mesurer et optimiser la capacité d'un réseau ferroviaire

9.1. Motivations et enjeux de la problématique traitée

Le développement et le succès commercial du concept « train grande vitesse », conjugué avec la saturation des infrastructures routières dans et autour des grandes villes, sont deux facteurs certains qui contribuent au regain d'intérêt accordé au transport ferroviaire. Parmi les systèmes de transports, le train possède également de nombreux atouts sur le plan du développement durable. Cet état de faits confère un avantage aux compagnies ferroviaires sur leurs concurrents aériens. Le train a détrôné l'avion ces dernières années sur bon nombre de liaisons courtes et moyennes distances. Conséquence directe du succès, le trafic ferroviaire affiche une augmentation constante en Europe ces dernières années. Mais pour maintenir et accentuer cette compétitivité, l'environnement économique impose aux compagnies ferroviaires de nouveaux défis, touchant notamment à l'amélioration de l'efficacité du système de transport ferroviaire, ou encore la qualité de service offert aux usagers. Vu du client, ces deux objectifs se traduisent par le souhait de voir la fréquence de trains augmenter et une meilleure ponctualité des trains par rapport à l'horaire communiqué.

9.1.1. *Analyse des stratégies de développement d'infrastructures ferroviaires*

La restructuration des chemins de fer européens, entreprise par la directive 91 / 440 / CEE du Conseil des Communautés européennes du 29 juillet 1991 relative au

Chapitre rédigé par Xavier Gandibleux, Julien Jorge, Xavier Delorme et Joaquín Rodriguez.

développement des chemins de fer communautaires, a conduit à la séparation des fonctions des gestionnaires de l'infrastructure et des entreprises ferroviaires. L'application de ces directives pour le réseau français a donné naissance à RFF (Réseau ferré de France) comme l'autorité en charge de la gestion du patrimoine lié à l'infrastructure, alors que la SNCF (Société nationale des chemins de fer français) devient un opérateur qui propose des circulations de trains voyageur et fret. Cette nouvelle organisation ouvre la possibilité de voir apparaître prochainement de nouveaux opérateurs ferroviaires. A l'image du transport aérien, les nouveaux opérateurs pourraient se positionner sur des liaisons les plus rentables et introduire des changements significatifs dans le transport de voyageurs. Le transport de fret est également concerné. La saturation des réseaux routiers et la prise en compte de l'impact environnemental de ce mode de transport devraient provoquer dans le futur un report d'une partie des flux sur le ferroviaire, qui se traduira par un accroissement du nombre de trains de fret en service.

Le système ferroviaire est comparable à un système de production. Un train qui circule est un consommateur de ressources, qui, dans le cas présent, sont des éléments de l'infrastructure ferroviaire (voies, jonctions, quais, etc.). La notion de *sillon* correspond à la mobilisation de l'ensemble des ressources d'une infrastructure ferroviaire qui permettront à un train d'effectuer sa mission. La mission d'un train peut être définie de manière simplifiée par les points origine/destination, les dates de départ/arrivée correspondantes, les différents points desservis, les durées d'arrêt correspondantes et les profils de vitesses entre les différents points. Le sillon sera aussi l'entité de transaction entre un gestionnaire d'infrastructure et un opérateur. L'opérateur qui souhaite faire passer un train voyageur ou fret sur une infrastructure est amené à acheter un sillon dans un catalogue de sillons fourni par le gestionnaire d'infrastructure. Dans ce contexte d'un marché composé d'opérateurs en concurrence, le gestionnaire d'infrastructure doit donc construire ce catalogue de sillons avec des prix qui offrent la meilleure perspective économique.

Une situation de saturation de l'infrastructure peut conduire à la construction ou la modification d'une partie de celle-ci. Une telle décision est prise au niveau stratégique de la gestion prévisionnelle de l'infrastructure ferroviaire. Elle engendre des investissements financiers colossaux et perturbe le fonctionnement du système existant avec des travaux pouvant nécessiter plusieurs mois ou années. Il convient donc d'être en mesure d'argumenter et d'évaluer quantitativement les demandes de modification d'infrastructure.

Pour ces deux besoins, il apparaît crucial de disposer d'outils opérationnels permettant d'analyser les stratégies de développement d'infrastructures. La question centrale préliminaire à l'utilisation optimale d'une infrastructure appelle une mesure de la capacité de l'infrastructure ferroviaire.

9.1.2. *La capacité de l'infrastructure ferroviaire*

En se réfèrant à la définition donnée par l'Union internationale des chemins de fer [UIC 04], la capacité ferroviaire est un concept à plusieurs dimensions. Une première dimension est bien sûr le *nombre de trains* qu'il est possible de faire circuler dans un intervalle de temps. Une autre dimension est la *vitesse moyenne* des trains ; si les vitesses augmentent, la distance d'arrêt qui doit séparer les trains augmente, ce qui peut conduire à diminuer le nombre de trains. Une troisième dimension est la *stabilité* des horaires, elle représente la capacité d'une grille horaire à absorber les retards dus aux aléas de l'exploitation. Pour améliorer la stabilité, des marges sont ajoutées aux temps alloués à chaque circulation, ce qui conduit aussi à réduire le nombre de trains. Enfin, la dernière dimension est l'*homogénéité*, plus les types de trains qui circulent ont des vitesses différentes, moins de trains pourront circuler dans un même intervalle de temps. Les services de trains ayant des vitesses différentes seront ainsi qualifiés de « mixtes », par opposition aux services de trains homogènes.

Selon la fiche UIC [UIC 04], la capacité d'une infrastructure peut être représentée par une constante correspondant au périmètre du polygone qui relie les valeurs des quatre paramètres fondamentaux précédents. Pour une infrastructure donnée, chaque modification d'un ou plusieurs paramètres se répercutera sur les autres de telle manière que le périmètre du polygone reste identique. La figure 9.1 illustre de manière théorique deux contextes d'utilisation de la capacité d'une même infrastructure. Le premier polygone (trait plein) correspond à un service de trains mixtes et rapides, et le second à un service de trains homogènes, avec une marge plus importante, malgré le nombre plus élevé de trains.

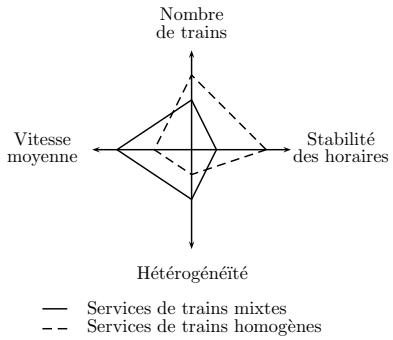


Figure 9.1. *Equilibre de capacité*

A ces paramètres fondamentaux, s'ajoutent selon les circonstances, des paramètres auxiliaires, qui influent aussi sur la notion de capacité. Parmi ceux-ci, on retrouve :

1) la structure des horaires (la mise en place de correspondances ou le choix d'horaires cadencés va « consommer » de la capacité) ;

2) la qualité de service (le niveau de qualité recherché est aussi très corrélé aux choix faits dans le nombre de trains, les structures d'horaires, la vitesse moyenne et les marges qui permettent de garantir la stabilité des horaires) ;

3) le matériel roulant (la longueur des trains et le nombre de niveaux du matériel roulant impactent l'offre réelle, et donc la capacité perçue par les clients).

Parmi les questions souvent soulevées lors de l'analyse de la capacité d'une infrastructure ferroviaire, celles qui reviennent le plus souvent sont :

1) la capacité de l'infrastructure est-elle *suffisante* pour faire face aux évolutions de l'offre ?

2) quels sont les investissements offrant le *meilleur débit* pour les offres à venir ?

A ces deux questions centrales correspondent deux types de problèmes d'optimisation dont la résolution offre un élément de réponse. Les problèmes associés à ces questions sont appelés respectivement « problème de faisabilité » et « problème de saturation ».

La pluralité de facettes de la capacité est une explication possible à la multitude de méthodes mises au point pour son évaluation. Les différentes méthodes présentées dans la littérature se regroupent en quatre catégories [ROD 07] : les formules analytiques, les formules probabilistes, les méthodes par construction d'horaires et les méthodes de simulation. Les travaux présentés dans ce chapitre s'inscrivent dans le cadre du projet RECIFE (REcherche sur la Capacité des Infrastructures FErroviaires) et relèvent de la catégorie des méthodes par construction d'horaires. Ainsi, la capacité est dérivée d'une grille horaire élaborée de façon la plus dense possible. Elle correspond à la situation saturée, donc à la limite de capacité.

9.1.3. *Le projet RECIFE*

Les contributions à l'évaluation de capacités d'infrastructures ferroviaires fondées sur des méthodes par construction d'horaires sont récentes ([HAC 97, SCH 08, VEL 05, ZWA 96] par exemple), et très peu de logiciels opérationnels sont disponibles. A notre connaissance, deux systèmes ont été utilisés pour mener des études sur le réseau français : CAPRES[1] et DEMIURGE[2].

1. http://www.fasta.ch/capres/capres_english.htm.

2. http://recherche.sncf.com/la_recherche_et_ses_domaines/exploitation/demiurge.html.

CAPRES est destiné à l'étude de la saturation de lignes. Le logiciel est issu de l'Ecole polytechnique fédérale de Lausanne. Il a été largement utilisé pour des études sur le réseau ferroviaire suisse, mais également sur le réseau français en région lyonnaise. DEMIURGE s'attache à la même problématique. Le logiciel a été commandé par la SNCF et a été utilisé sur le réseau ferroviaire français. Un point commun entre ces deux d'outils d'analyse de réseaux se situe au niveau de la modélisation de l'infrastructure. Ils sont destinés à des études au niveau d'une ligne et donc considèrent le réseau d'une manière macroscopique. Une gare ou une bifurcation sera considérée comme un élément complexe dans le modèle, mais sans toutefois offrir la possibilité d'une description très fine des événements qui s'y passent.

Notre proposition vient en complément à ces deux références, en considérant un niveau de modélisation de l'infrastructure davantage microscopique. Les études visées concernent une gare ou bifurcation. L'étendue géographique d'une infrastructure de ce type étant petite à l'échelle d'une ligne, les distances parcourues sont courtes et le trafic y est généralement très dense. La méthode proposée étant basée sur la construction d'une grille horaire des circulations, des approximations sur la durée des conflits risquent de masquer des incompatibilités, ou au contraire d'en générer des fausses. Dans un cas comme dans l'autre, cela change radicalement l'intérêt de la grille horaire obtenue.

Les attendus du projet RECIFE [ROD 07] étaient de développer des outils d'aide à la décision qui s'appuient sur les techniques issues de la recherche opérationnelle pour l'étude *spécifique des nœuds ferroviaires ou des gares*. Les objectifs principaux du projet étaient de :

1) proposer une *modélisation mathématique* de ces problèmes, permettant l'appréciation de solutions au regard de *différents objectifs* ;

2) intégrer les résultats précédents dans une *plate-forme logicielle* ouverte incluant des *modules d'analyse et de visualisation* ;

3) valider le modèle et la résolution sur des *instances de problèmes réels* à partir de jeux de données du *nœud de Pierrefitte-Gonesse* et de la *gare de Lille-Flandres*.

Les contributions effectives de ce projet rencontrent les attendus, avec notamment la production de modèles et méthodes :

1) de mesure et d'optimisation de la capacité [DEL 03, DEL 04, GAN 04] ;

2) de mesure de la stabilité d'horaire [DEL 09] ;

3) la définition et l'intégration de ces deux entités dans un système d'aide à la décision multi-objectif [GAN 08].

Composant algorithmique parmi ces contributions, ce chapitre met l'accent sur la méthode de résolution approchée, qui repose sur le paradigme des colonies de fourmis, et qui a été proposée pour la mesure et l'optimisation de la capacité des nœuds

ferroviaires ou des gares. Les expérimentations numériques présentées illustrent un ensemble de fonctionnalités faisant usage de cet algorithme en tant que solveur, d'une manière transparente pour l'utilisateur.

La suite du chapitre s'organise de la façon suivante. La section 9.2 introduit les définitions des éléments ferroviaires préliminaires à la formulation d'un modèle d'optimisation, lequel s'apparente au problème d'optimisation combinatoire classique, dit de *set packing*. Les méthodes de résolution investiguées sont évoquées dans la section 9.3, en développant expressément l'algorithme inspiré des colonies de fourmis. La section 9.4 illustre un ensemble de situations d'étude où l'algorithme de fourmis est utilisé comme solveur. Cette section se termine en soulignant tout l'intérêt de la présente solution opérationnelle, quand les cas d'étude se traduisent par des instances numériques de très grande taille à traiter. Enfin, une conclusion sur les enseignements tirés de cette contribution, ainsi que différentes questions ouvertes faisant l'objet de travaux de recherche en cours, terminent le chapitre.

9.2. Modélisation

En préliminaire à la description de la modélisation proposée au problème de capacité d'infrastructure ferroviaire, les composants qui interviennent dans le système ferroviaire, la cinématique qui anime le système ainsi que les interactions qui apparaissent entre ces composants sont introduits. Bien que complexe, l'ensemble du système à représenter s'identifie à un problème de disjonctions de ressources souvent unaires. Il est formalisé à l'aide d'un modèle d'optimisation combinatoire dit de *set packing*. L'instanciation de ce modèle sur la problématique ferroviaire est donnée dans cette section sous une forme simplifiée. C'est ce problème d'optimisation qui sera traité à l'aide du paradigme des colonies de fourmis, sans toutefois utiliser des caractéristiques qui habitent le problème ferroviaire.

9.2.1. *Description des parties du système réel à traiter*

L'infrastructure ferroviaire se compose de *voies* qui permettent la circulation de trains, et d'un système de *signalisations* qui garantit la *sécurité* des circulations. Une infrastructure peut présenter différentes typologies, allant d'une *simple voie* à un *réseau complet* en passant par des ensembles de *voies parallèles (tronçons) ou sécantes (jonctions ou nœuds)* et des *zones d'arrêt (quais)*. La sécurité des circulations repose principalement sur la notion de *canton*. Un canton est une partie de l'infrastructure pouvant accueillir un et un seul train à tout instant. Pour cela, en amont de chaque point d'entrée d'un canton figure un signal. Le signal précise si le canton est libre ou occupé par un train.

Pour déterminer l'aspect des signaux, il faut donc connaître la position précise des trains. L'entité atomique d'une infrastructure qui permet de détecter la présence d'un

train sur la voie est appelée « zone de détection ». Un canton sera composé d'une ou plusieurs zones de détection. La détection se fait le plus souvent à l'aide d'un dispositif électrique appelé « circuit de voie ».

Une *ligne* est une suite ordonnée de tronçons et de nœuds formant un chemin dans un réseau. Ce sont les lignes qui structurent la façon dont les ressources sont mobilisées dans une infrastructure ferroviaire. Elles traversent une succession de régions géographiques, comme par exemple le nœud de Pierrefitte-Gonesse qui est traversé par la ligne Paris-Nord/Bruxelles-Midi (figure 9.2). Au sein d'une région, plusieurs *parcours* peuvent appartenir à une même ligne. Cet ensemble de possibles représente les alternatives pour franchir la région. Un parcours peut combiner différents *itinéraires* en fonction de la signalisation. *In fine*, un parcours se décline par l'ensemble des cantons qu'il emprunte (figure 9.3).

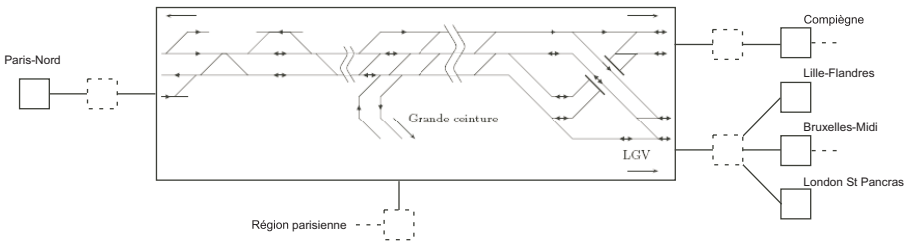


Figure 9.2. *Exemple de régions (matérialisé par les carrés/rectangles), telles que le nœud de Pierrefitte-Gonesse (rectangle central) qui se voit traversé par plusieurs lignes, telles que la ligne Paris-Nord/Lille-Flandres*

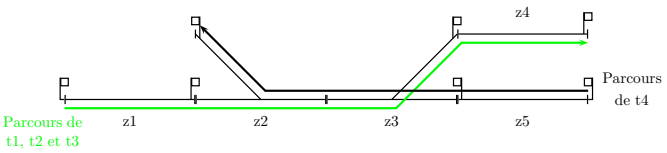


Figure 9.3. *Exemple de deux parcours sur une infrastructure de type bifurcation. Un parcours est emprunté par les trains t1 à t3, l'autre par le train t4. Les zones de détection sont numérotées de z1 à z5. Quatre cantons sont définis, avec les zones z2 et z3 appartenant à un même canton : c1={ z1 }, c2={ z2, z3 }, c3={ z4 } et c4={ z5 }.*

La *marche d'un train* sur une infrastructure est une circulation réelle d'un convoi, laquelle est caractérisée par son passage à un point donné, à une vitesse donnée et à une date donnée. Elle est matérialisée par un sillon dans un *graphique de circulation* espace-temps (figure 9.4). Le sillon représente la capacité d'infrastructure requise (parcours et horaire) pour faire circuler un train donné d'un point à un autre à un moment donné (directive 95/19/CE du Conseil de l'Union européenne).

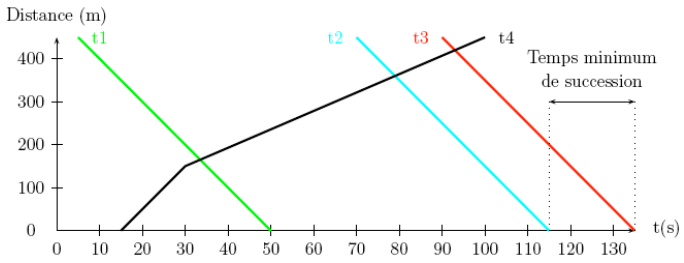


Figure 9.4. *Exemple d'un graphique de circulation pour l'infrastructure représentée par la figure 9.3. Il est constitué de quatre sillons correspondant aux quatre trains repérés par t1 à t4. Les trains t2 et t3 se succèdent en batterie au plus court, compte tenu du temps minimum de succession entre trains. Le train t4 circule dans l'autre sens et ralentit au cours de la période observée.*

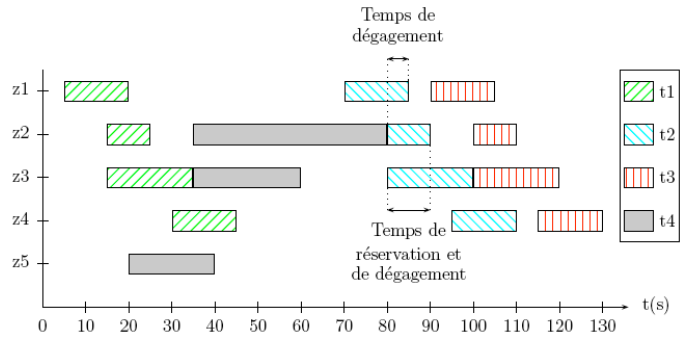


Figure 9.5. *Exemple de diagramme de Gantt. Les rectangles traduisent le temps d'occupation (et éventuellement de réservation et de dégagement) d'un train sur les zones de détection d'une infrastructure. Cette représentation véhicule la même information que la figure 9.4 mais d'une manière plus fine. A noter dans cet exemple la synchronisation des dates de début d'occupation des zones z2 et z3, puisqu'elles appartiennent à un même canton.*

La marche d'un train peut encore se représenter dans un diagramme de Gantt (figure 9.5). Encore d'usage dans le métier bien qu'ancienne, cette représentation permet d'apprécier plus finement la cinématique liée à l'évolution du train sur l'infrastructure. Cependant, la maîtrise de cette cinématique n'est pas indispensable pour la compréhension du modèle d'optimisation et de l'algorithme de résolution dans le contexte de ce chapitre. Elle n'est donc pas explicitée dans cet écrit, mais le lecteur intéressé trouvera ce niveau de détail dans [DEL 03].

Contrairement à CAPRES et DEMIURGE, c'est au niveau de la zone de détection que se situe la granularité de nos travaux. Elle traduit une ressource unaire consommée par un train au cours de sa circulation. C'est précisément à ce niveau que peuvent

apparaître des conflits de circulation. On appelle conflit de circulation la séquence des zones communes à deux parcours. Un conflit traduit une incompatibilité entre les choix de parcours et d'heures d'entrée des trains dans une infrastructure. La figure 9.6 illustre un tel conflit entre deux trains qui demandent simultanément la zone 3. Le lecteur intéressé trouvera dans [DEG 07] une typologie des situations pouvant conduire à un conflit.

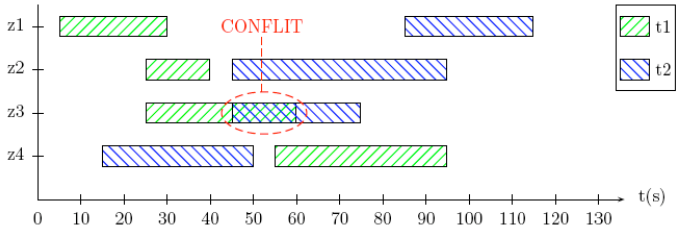


Figure 9.6. *Exemple d'une situation de conflit mettant en jeu deux trains t1 et t2 au passage sur quatre zones (z1, z2, z3 et z4). Ces deux trains sont en conflit puisqu'ils demandent la zone z3 au même moment*

Pour résumer ces éléments plus formellement, soit T et R représentant respectivement l'ensemble des trains à considérer et l'ensemble des parcours possibles, une circulation se décline en un train $t \in T$ sur un parcours $r \in R$ à une heure d'entrée h_{in} prise dans un intervalle défini par une fenêtre de temps, $\left[\underline{h}, \overline{h} \right]$. Il est raisonnable de considérer la fenêtre constituée de valeurs discrètes sur un pas unitaire de trente ou soixante secondes. Les circulations définissent une grille horaire, terme caractérisant le graphique d'une région et représentant les heures de passage prévues des trains sur chacune des voies de la zone considérée. La figure 9.13 est un exemple de grille horaire.

9.2.2. *Modélisation de la problématique ferroviaire traitée*

La modélisation proposée [DEL 03] est basée sur un graphe d'incompatibilité train-parcours et s'inspire des travaux de P.J. Zwaneveld *et al.* [ZWA 96, ZWA 97] réalisés dans le cadre du projet DONS [BER 94]. La formulation mathématique de ce modèle correspond à un problème classique d'optimisation combinatoire, connu pour être NP-difficile, et appelé *set packing problem* (SPP) [GAR 79, NEM 99].

Un SPP se définit de la façon suivante. Soit un ensemble fini d'éléments valués $I = \{1, \ldots, n\}$ et $\{T_j\}, j \in J = \{1, \ldots, m\}$ une collection de sous-ensembles de I, un couplage est un sous-ensemble $P \subseteq I$ d'éléments disjoints (c'est-à-dire tel que $\nexists (p_1, p_2) \in P^2, \exists j \in J, (p_1, p_2) \in T_j^2$). L'ensemble J peut aussi être vu comme un ensemble de contraintes entre les éléments de l'ensemble I. La figure 9.7 illustre un

exemple didactique de couplage P à partir d'un ensemble $I = \{1, \ldots, 6\}$ et des sous-ensembles $T_1 = \{2, 4\}$, $T_2 = \{2, 3, 5, 6\}$, $T_3 = \{1, 6\}$. Elle illustre une solution réalisable à ce problème, de valeur égale à s.

L'objectif du problème de *set packing* est de maximiser la valeur totale du couplage obtenu. Le nom de « problème de couplage généralisé » est ainsi parfois utilisé pour le SPP. Ce problème peut être formulé avec le modèle mathématique suivant :

$$\left[\begin{array}{ll} \max\ z(X) = \displaystyle\sum_{i \in I} c_i x_i & \\ s.c. & \displaystyle\sum_{i \in I} t_{i,j} x_i \leq 1, \forall j \in J \\ & x_i \in \{0, 1\} \quad , \forall i \in I \end{array}\right] \tag{9.1}$$

avec un vecteur $X = (x_i)$ tel que $x_i = 1$ si $i \in P$, 0 sinon ;

– un vecteur $C = (c_i)$ tel que $c_i =$ valeur de l'élément i ;

– une matrice $T = (t_{i,j})$ telle que $t_{i,j} = 1$ si $i \in T_j$, 0 sinon.

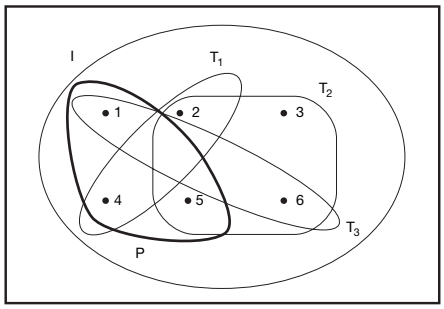


Figure 9.7. *Exemple didactique de couplage d'un problème SPP*

Le tableau 9.1 donne une forme matricielle correspondant à l'exemple didactique.

$C =$

1	2	3	4	5	6
1	1	1	1	1	1

$T =$

0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	0	1

Tableau 9.1. *L'instance numérique correspondant à l'exemple didactique illustré par la figure 9.7*

Avant de présenter une modélisation en termes ferroviaires, la définition des deux problèmes à représenter, correspondant à une étude de capacité, tels qu'ils ont été mentionnés dans le paragraphe 9.1.2, est précisée :

– *le problème de faisabilité* consiste à vérifier la faisabilité du passage d'une combinaison donnée de trains selon une grille horaire établie (points et dates d'entrée-sortie fixés) dans l'infrastructure considérée. La question est alors de savoir s'il existe un routage (affectation d'un parcours et éventuellement d'un quai) de ces trains permettant de les faire tous passer sans retard ;

– *le problème de saturation* consiste à introduire le maximum de circulations supplémentaires dans une grille horaire établie (éventuellement vide). Ce problème prolonge une solution d'un problème de faisabilité. Les trains ainsi ajoutés représentent la *marge de capacité* disponible de l'infrastructure par rapport à une offre. Les nouveaux trains sont choisis parmi une liste de trains appelée *liste saturante*. Les trains de la liste saturante ont un nombre fini de dates d'entrée et de sortie possibles et des niveaux de préférence peuvent éventuellement être définis. Le nombre fini de dates d'entrée et de sortie possibles permet notamment d'intégrer les contraintes horaires extérieures au nœud.

Ces deux problèmes peuvent être formulés par un SPP comportant deux fonctions objectifs z^1 et z^2 considérées lexicographiquement : le nombre de trains routés appartenant à la grille horaire établie (objectif z^1 de faisabilité) est d'abord maximisé. Il vient ensuite la maximisation du nombre de trains appartenant à la liste des trains supplémentaires possibles (objectif z^2 de saturation), sans dégrader la valeur sur z^1. Le modèle est construit sur les variables de décision $x_{t,r}$ binaires qui valent 1 lorsque le train t est routé sur le parcours r, 0 sinon. C'est un SPP bi-objectif où les deux objectifs sont traités en ordre lexicographique (lex) z^1, z^2. Il s'écrit de la façon suivante :

$$lex \quad \left(\max z^1 = \sum_{t\in T_{GH}, r\in R_t, \delta\in\Delta_t} x_{t,r,\delta} \quad ; \quad \max z^2 = \sum_{t\in T_{LS}, r\in R_t, \delta\in\Delta_t} x_{t,r,\delta}\right)$$

$$\text{s.c.} \quad \sum_{r\in R_t, \delta\in\Delta_t} x_{t,r,\delta} \leq 1 \qquad \forall t \in T \tag{9.2}$$

$$x_{t,r,\delta} + x_{t',r',\delta'} \leq 1 \qquad \forall\big((t,r,\delta),(t',r',\delta')\big) \in Inc \tag{9.3}$$

$$x_{t,r,\delta} \in \{0,1\} \qquad \forall t \in T,\ r \in R_t, \delta \in \Delta_t \tag{9.4}$$

avec :

R_t	: sous-ensemble des parcours admis pour un train t ($R_t \subseteq R$) ;
T_{GH}	: les trains de la grille horaire ;
T_{LS}	: les trains de la liste saturante ;
T	: l'ensemble des trains considérés (c'est-à-dire $T = T_{GH} \cup T_{LS}$) ;
Δ_t	: l'ensemble des heures d'entrée h_{in} admises pour un train t ;
Inc	: l'ensemble des incompatibilités entre tous les trains, pour toutes les combinaisons de parcours et d'heures admises.

Ce modèle comporte trois types de contraintes. La contrainte (9.2) impose l'unicité de parcours pour un même train. La contrainte (9.3) traduit les incompatibilités de routages. Enfin, la contrainte (9.4) définit le caractère binaire des variables de décision.

Le modèle présenté ici est une version simplifiée. Le modèle complet [DEL 03] permet notamment de considérer des contraintes supplémentaires pour pouvoir l'appliquer au cas d'une gare. Ces contraintes concernent les ordres d'arrivée et de départ des trains à un même quai, la capacité des quais et enfin les « équilibres » entre paires de trains (le quai d'arrivée d'un premier train doit être le même que celui de départ d'un second train). En outre, le modèle complet offre également la possibilité de véhiculer une notion de préférence du décideur. Dans le modèle ci-dessus, les fonctions objectif sont à coût unitaire. En agissant sur le levier que constituent les coefficients de coût de la fonction économique, il est aisé de favoriser des parcours ou des classes de trains. La section 9.4 illustre cet usage.

9.3. Algorithme de résolution

La résolution exacte des problèmes de faisabilité et de saturation présentés dans la section précédente a été tentée dans un premier temps. Par résolution exacte, on entend la mise en œuvre d'un algorithme permettant de trouver au moins une solution optimale et de démontrer son optimalité. Même si la littérature sur la résolution exacte du SPP est relativement abondante, notamment concernant son étude polyhédrale et la recherche d'inégalités valides (voir par exemple [BOR 98, COR 01]), aucune méthode exacte *ad hoc* efficace ne permet de traiter des problèmes de grande taille. Nos travaux se sont donc orientés vers une méthode générique qui s'appuie sur l'utilisation du solveur MILP de CPLEX[3].

Les expérimentations [DEL 03] ont montré que cette méthode éprouvait beaucoup de difficultés à fournir une solution, même après un temps de traitement conséquent (plus de douze heures). Une des causes est la très mauvaise qualité de la relaxation linéaire (jusqu'à plus de 1 000 % de la solution optimale). De plus, la qualité de la solution entière s'est avérée souvent mauvaise, au regard des solutions entières obtenues ultérieurement par d'autres moyens. Pour améliorer les performances de résolution, quatre types de prétraitement des instances ont été considérés. Ils reposent sur la recherche de cliques et des tests de dominance. Les cliques obtenues correspondent à des inégalités valides du problème. Elles ont permis d'améliorer significativement la valeur de la borne supérieure obtenue par relaxation linéaire et par conséquent les performances de la méthode de résolution. De même, les tests de dominance ont permis de réduire très fortement la taille des problèmes considérés.

3. ILOG CPLEX : http://www.ilog.com/products/cplex/.

Toutefois, ces deux améliorations ne se sont pas avérées suffisantes, compte tenu de la taille des instances ferroviaires à traiter. Or les études de capacité sont appelées à être menées avec des ressources (temps de calcul et/ou mémoire) standards et limitées. De plus, une telle étude relève la plupart du temps d'un processus itératif, avec un décideur, et ne supporte donc pas des temps de réponse trop importants. La recherche d'un *optimum* apparaît généralement comme n'étant pas envisageable, et il est raisonnable de s'orienter vers la recherche d'une solution de bonne qualité, compte tenu des ressources disponibles, à l'aide d'une méthode approchée. Les métaheuristiques [PIR 02] font partie de ces méthodes de résolution, où les algorithmes de fourmis représentent une réponse potentielle, aux côtés d'autres paradigmes comme les algorithmes génétiques, la recherche tabou, le recuit simulé ou encore la méthode GRASP.

A notre connaissance et d'une manière surprenante, aucune métaheuristique n'a été proposée pour le SPP avant 2004. Nous avons développé deux méthodes approchées dérivées de paradigmes différents. La première [DEL 04] est inspirée de la métaheuristique GRASP (*Greedy Randomized Adaptive Search Procedure* - [FEO 89]). La seconde [GAN 04, GAN 05] s'inspire des colonies de fourmis [DOR 99] et fait l'objet de la suite de cette section (le lecteur est considéré empreint des principes généraux de cette métaheuristique ; dans la négative, il trouvera les prérequis indispensables dans les premiers chapitres de cet ouvrage).

L'algorithme de colonie de fourmis (ACF) développé épouse le schéma conventionnel de cette classe de méthodes, tout en présentant différentes caractéristiques originales :

```
initialiseParametres( )
tantque not( estFini ?( ) ) faire
  construireSolutions( )
  gestionSolutions( )
  gestionParametres( )
fin tantque
présenterMeilleureSolution( )
```

Les caractéristiques et originalités sont présentées dans les sous-sections suivantes. A noter qu'aucun des paramètres présentés dans l'algorithme ne doit faire l'objet d'un réglage préliminaire de la part d'un utilisateur. Quelle que soit l'instance numérique de *set packing* à traiter, il s'agit soit de constantes aux valeurs recommandées résultant d'expérimentations numériques, soit d'un réglage auto-adaptatif. Ces paramètres sont indépendants de l'instance à traiter. Le réglage auto-adaptatif met en jeu une *probabilité* $\mathcal{P}$ qui sert à piloter la convergence de l'algorithme. La valeur instantanée de $\mathcal{P}$ est utilisée à différents niveaux stratégiques de l'algorithme (catégorie de fourmis, mode d'exploitation des phéromones, perturbation des phéromones). L'algorithme utilise uniquement l'aspect apprentissage déduit du comportement des fourmis.

A l'exception des recherches locales, il n'utilise pas les caractéristiques du SPP, ni les spécificités du problème ferroviaire.

9.3.1. *Variables, solutions et phéromones*

Pour un problème de *set packing* comportant *n variables* binaires x_i, un vecteur de *phéromone* de dimension *n* à valeurs réelles sera manipulé par l'algorithme. A chaque variable x_i est associé un niveau de phéromone ϕ_i. Toutes les solutions manipulées par l'algorithme sont dites *réalisables saturées*. Une telle solution consiste en une collection de variables binaires fixées à un (les autres étant fixées à zéro, par dépendance avec les variables à un), telle que toutes les contraintes soient respectées et qu'il soit impossible de fixer une variable supplémentaire, sans violer au moins une contrainte.

Illustration : sur l'exemple didactique du paragraphe 9.2.2, le problème comporte six variables, x_1 à x_6. Initialement ces six variables sont libres, c'est-à-dire qu'aucune n'a reçu une valeur 0 ou 1. Si $x_2 = 1$ alors les contraintes 1 et 2 sont saturées. Conséquence directe de ce choix, les variables x_3, x_4, x_5 et x_6 sont nécessairement fixées à zéro (car si l'une d'entre elles venait à être fixée à 1, cela provoquerait une violation de la contrainte 1 ou 2), x_1 est libre. Avec $x_2 = 1$, il est encore possible de choisir $x_1 = 1$, ce qui conduit à une solution réalisable saturée, de valeur $z(x) = 2$.

9.3.2. *Construction de solutions*

Cette phase, représentée par la procédure `construireSolutions` se décline pour un cycle (itération) de la manière suivante. Une fourmi construit une solution réalisable saturée en sélectionnant itérativement, à partir des valeurs de phéromone, les variables binaires fixées à 1. Deux catégories de fourmis sont considérées, la fourmi *fonceuse* et la fourmi *curieuse*. La façon de sélectionner une variable dépend de la catégorie dont est issue la fourmi :

– la fourmi fonceuse utilise intensivement l'information mise à sa disposition pour construire des solutions. Dans les faits, une fourmi fonceuse sélectionne toutes les variables pour constituer une solution réalisable saturée, par l'application d'un mécanisme de choix glouton sur les valeurs de phéromone (fonction `bestValue`) ;

– la fourmi curieuse relativise le comportement fonceur de la première en lui accordant une certaine aptitude à explorer l'espace de solutions. Une fourmi curieuse sélectionne chaque variable indépendamment, par application d'un mécanisme de choix mixte : soit une roulette biaisée sur les valeurs de phéromone (fonction `rouletteWheel`), soit un choix glouton sur les valeurs de phéromone (fonction `bestValue`).

$I_t \leftarrow I$; $x_i \leftarrow 0, \forall i \in I_t$
tantque ($I_t \neq \emptyset$) **faire**
  si (curieuse) **and** (`randomValue(0,1)` $\geq \mathcal{P}$) **alors**
    $i^* \leftarrow$ `rouletteWheel`$(\phi_i, i \in I_t)$
  sinon
    $i^* \leftarrow$ `bestValue`$(\phi_i, i \in I_t)$
  fin si
  $x_{i^*} \leftarrow 1$; $I_t \leftarrow I_t \setminus \{i^*\}$; $I_t \leftarrow I_t \setminus \{i : \exists j \in J, t_{i,j} + t_{i^*,j} > 1\}$
fin tantque
`AppliquerRechercheLocale()`

Ce principe s'inspire du mode exploration et exploitation présenté dans [STU 98]. Une fourmi possède un caractère de fonceuse ou de curieuse, compte tenu de la *probabilité* $\mathcal{P}$, le degré de curiosité d'une fourmi ($\mathcal{P}$= 0 : totalement curieuse ; $\mathcal{P}$= 1 : totalement fonceuse). Sur toutes les solutions réalisables saturées est appliquée *a posteriori* une *recherche locale*, qui a pour mission d'améliorer la qualité de la solution élaborée par une fourmi. L'algorithme ACF utilise une seule colonie de `maxAnt`=15 fourmis. La colonie réalise un cycle, qui se traduit par la sortie de toutes les fourmis membres de la colonie sur le territoire à visiter. Au terme d'un cycle, `maxAnt` solutions sont construites.

Seul le parcours réalisé par une fourmi sera retenu à l'issue de chaque cycle (procédure `gestionSolutions`). Il correspond au parcours présentant la meilleure performance au regard de la fonction à optimiser. C'est donc la fourmi qui a rapporté cette solution qui laissera des traces de phéromone. Ces cycles vont se succéder jusqu'à ce que le travail réalisé par la colonie de fourmis ait consommé la nourriture disponible sur leur territoire (prédicat `estFini ?`). La meilleure des meilleures solutions identifiées sera présentée en conclusion de la phase de recherche de l'algorithme (procédure `présenterMeilleureSolution`).

9.3.3. *Gestion des traces de phéromone*

Le modèle de phéromone utilisé est reconnu pour être un point central dans un algorithme de colonies de fourmis. Il représente le modèle d'échantillonnage probabiliste de l'espace de recherche. Dans l'algorithme développé, les valeurs des phéromones évoluent sur l'intervalle [0,0; 1,0]. La procédure `initialiseParametres` initialise tous les niveaux de phéromone à la valeur maximale (`phiInit` = 1,0). Les coefficients d'évaporation et de dépôt des phéromones sont fixes et sont établis *a priori*. Le coefficient d'évaporation des phéromones `rhoE` est réglé à la valeur 0,9, et le coefficient de dépôt `rhoD` est établi à la valeur 0,1. La mise à jour des phéromones est réalisée par la procédure `gestionParametres`. Toutes les phéromones sont concernées par l'application du mécanisme d'évaporation. La règle retenue est multiplicative et est donnée par :

$$\phi_i \leftarrow \phi_i \times \texttt{rhoE},\ \forall i \in I$$

Seules les phéromones correspondant aux variables fixées à 1 dans la meilleure solution $\tilde{x}^t$ issue du cycle t sont concernées par l'application du mécanisme de dépôt. La règle retenue est additive et est donnée par :

$$\phi_i \leftarrow \phi_i + \texttt{rhoD},\ \forall i \in I \mid \tilde{x}_i^t = 1$$

9.3.4. *Perturbation des phéromones*

Les hypothèses prises pour la gestion des traces laissées par les meilleures fourmis au terme de chaque cycle font qu'au bout d'un certain temps, des variables peuvent être ignorées par l'algorithme, du fait d'un niveau de phéromone proche de zéro. A l'opposé, des variables qui reviennent très souvent dans les bonnes solutions risquent d'occulter le principe de sélection de variables. Un mécanisme de perturbation du vecteur de phéromone [GAN 04] a été développé dans l'idée de perturber le territoire des fourmis. Son objectif est de permettre à l'algorithme de redémarrer avec un historique partiel de l'évolution suivie par l'algorithme au cours des étapes précédentes. Les conditions de déclenchement de ce mécanisme pour un cycle t reposent sur le passé de la convergence de l'algorithme. Lorsque deux conditions sont rassemblées, alors le déclenchement s'effectue :

1) la première condition détecte des niveaux bas de phéromone. Soit $\phi_{nul} = 0{,}001$ le seuil pour considérer un niveau de phéromone comme nul ; il suffit qu'au moins une variable ait son niveau de phéromone à « zéro » pour valider la condition :

$$\exists i \in I \mid \phi_i^t \leq \phi_{nul}$$

2) la seconde condition regarde la progression de la convergence de l'algorithme :

- pour les instances à coûts pondérés, il suffit que deux cycles successifs (`iterStagnant` = 2) donnent la même qualité de solution :

$$z(\tilde{x}^t) = z(\tilde{x}^{t+1})$$

- pour les instances de *set packing* dites à coûts unitaires, elle est vérifiée quand deux cycles successifs (`iterStagnant` = 2) donnent *stricto sensu* la même solution :

$$\tilde{x}^t = \tilde{x}^{t+1}$$

La procédure de perturbation réalise trois actions :

1) la première action atténue tous les niveaux de phéromone, d'un coefficient compris entre 85 % et 100 %, selon que l'on soit plus proche du début ou de la fin de l'exécution de l'algorithme. Soit `attenuation`ϕ fixé à 0,85, alors :

$$\phi_i \leftarrow \phi_i \times (\texttt{attenuation}\phi + (1 - \texttt{attenuation}\phi) \times \mathcal{P}), \forall i \in I$$

2) la seconde action vise à brouiller partiellement les traces de phéromone. Soit `prorata`ϕ`Aleatoires`, fixé à 0,10, le pourcentage de variables du problème qui vont voir leur niveau de phéromone fixé aléatoirement. Soit `set`ϕ`ResetLow` et `set`ϕ`ResetHigh`, respectivement de valeurs 0,05 et 0,45. La nouvelle valeur est d'autant plus faible que l'on est proche de la fin de l'exécution :

$$I_1 \leftarrow \lfloor \texttt{prorata}\phi\texttt{Aleatoires} \times n \rfloor \text{ valeurs prises aléatoirement dans } I$$

$$\phi_i \leftarrow \texttt{set}\phi\texttt{ResetLow} + \text{random}(0, \texttt{set}\phi\texttt{ResetHigh} \times (1 - \mathcal{P})), \forall i \in I_1$$

3) la dernière action a pour mission de redonner un rôle dans l'algorithme aux variables qui présentent un niveau de phéromone « bas ». Soit `set`ϕ`Low`, `set`ϕ`High` et `seuil`ϕ`Bas`, respectivement de valeurs 0,05, 0,20 et 0,1. Les niveaux de phéromone les plus bas vont être augmentés à une valeur comprise entre `set`ϕ`Low` et `set`ϕ`High` :

$$I_2 \leftarrow \{i \in I \mid \phi_i < \texttt{seuil}\phi\texttt{Bas}\}$$

$$\phi_i \leftarrow \phi_i + \text{random}(\texttt{set}\phi\texttt{Low}, \texttt{set}\phi\texttt{High}), \forall i \in I_2$$

La perturbation des phéromones est réalisée par la procédure `gestionParametres`. Les figures 9.8 à 9.11 illustrent le mécanisme de perturbation des phéromones, depuis la détection des conditions d'application de la perturbation, jusqu'à l'arrêt de l'algorithme.

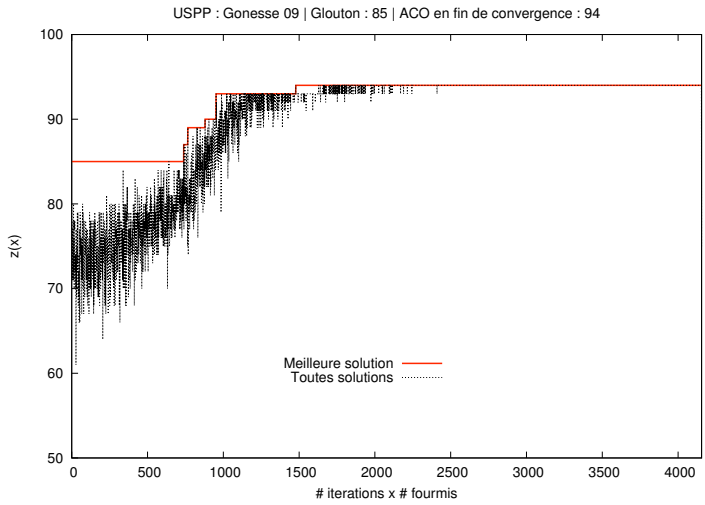


Figure 9.8. *Convergence de l'algorithme amenant au déclenchement de la procédure de perturbation*

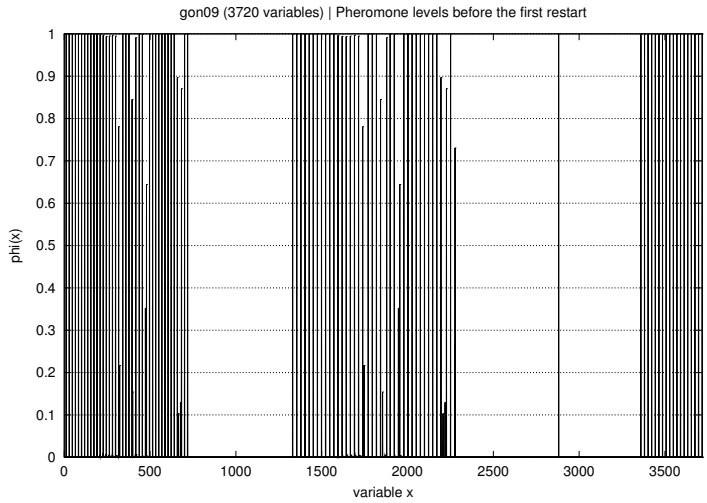


Figure 9.9. *Etat du vecteur de phéromone avant application de la procédure de perturbation*

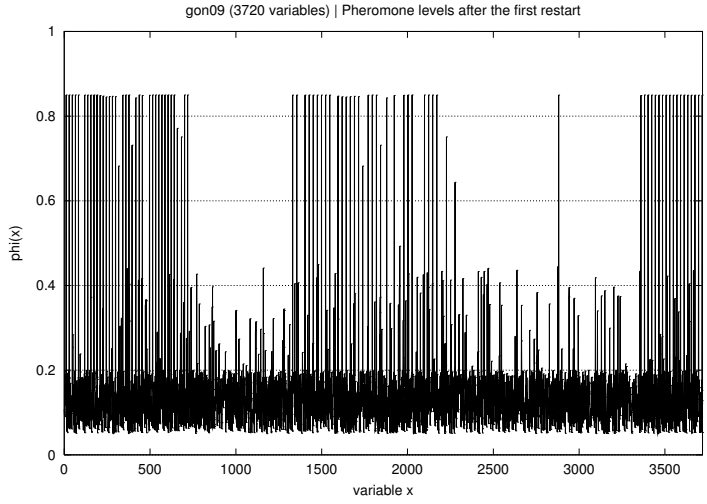


Figure 9.10. *Etat du vecteur de phéromone après application de la procédure de perturbation*

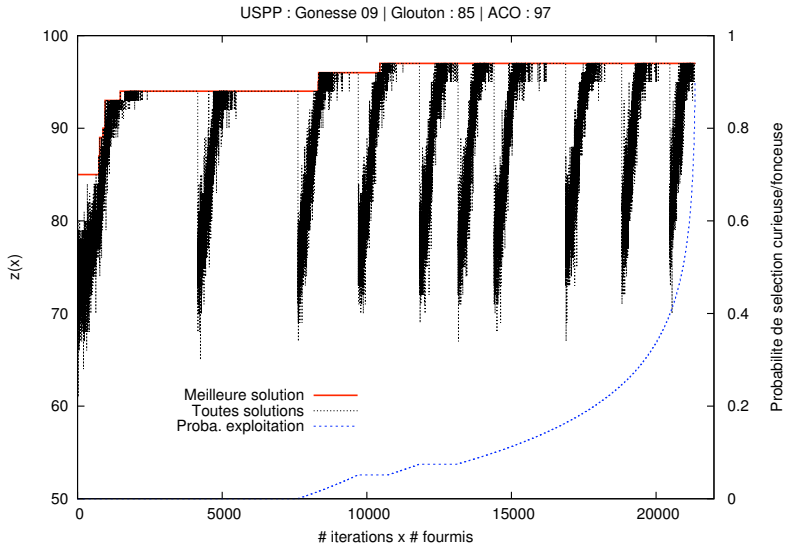


Figure 9.11. *Evolution de la convergence de l'algorithme suite à l'application de la perturbation*

9.3.5. *Stratégie auto-adaptative du pilotage de l'algorithme*

L'algorithme gère automatiquement le nombre de cycles à réaliser sur la base d'une condition d'arrêt dynamique [JOR 06]. La condition d'arrêt est fondée sur l'intérêt de la recherche et la diversité des solutions. On appelle phase de convergence de l'algorithme, l'ensemble des cycles qui précèdent le déclenchement de la procédure de perturbation des traces de phéromone. A l'issue de deux convergences (`numberRestart` = 2), le nombre total de cycles effectués (`NbInitialIterFaites`) est noté et pris en référence. Le nombre total de cycles à réaliser est ensuite établi comme suit :

$$\texttt{NbTotalIterArealiser} \leftarrow 2 \times \texttt{NbInitialIterFaites}$$

De plus, ce nombre est augmenté à chaque fois qu'une convergence améliore la meilleure solution connue. Soit `NbIterMoyen`, le nombre moyen de cycles pour une convergence, alors :

$$\texttt{NbTotalIterArealiser} \leftarrow \texttt{NbTotalIterArealiser} + \texttt{NbIterMoyen}$$

La mise à jour du nombre de cycles est réalisée par la procédure `gestionSolutions` suite à l'identification de l'amélioration de la meilleure solution connue.

En outre, la probabilité $\mathcal{P}$, qui impacte directement les stratégies de l'algorithme, va évoluer avec les cycles. L'algorithme va uniquement évoluer avec des fourmis curieuses durant les deux premières convergences. En d'autres termes, la probabilité $\mathcal{P}$ est nulle. Ensuite la probabilité $\mathcal{P}$ augmente itérativement, les fourmis deviennent de moins en moins curieuses. Ainsi, au cycle t, la probabilité s'obtient par :

$$\mathcal{P} = \max\left(\mathcal{P}, \frac{\log(\mathrm{t} - \texttt{NbInitialIterFaites})}{\log(\texttt{NbTotalIterArealiser} - \texttt{NbInitialIterFaites})}\right)$$

9.3.6. *Recherches locales*

Les recherches locales sont fondées sur un voisinage de type $k - p$ *exchange* [FEO 95]. Ce mouvement traduit que k variables fixées à 1 vont être échangées contre p variables fixées à 0 dans la solution courante, tout en garantissant la réalisabilité de la solution obtenue au terme de l'échange. Ces recherches sont des méthodes de descente, et visent à améliorer les solutions construites par les fourmis, pour un coût calculatoire qui demeure raisonnable. Elles sont au nombre de trois, notées 1-1W, 1-2U, 1-2W, les lettres U et W désignent l'application sur des instances respectivement à coûts unitaires (*unicost* – USPP) ou pondérées (*weighted* – WSPP). Ces recherches locales sont indépendantes de l'algorithme ACF. Pour ces raisons, le lecteur est renvoyé à [DEL 03] pour les détails concernant les recherches locales.

9.4. Expérimentations numériques

9.4.1. *Le nœud ferroviaire de Pierrefitte-Gonesse*

Le nœud ferroviaire de Pierrefitte-Gonesse est géographiquement situé au nord de Paris, dans une zone fortement urbanisée et à proximité de l'aéroport parisien de Roissy-Charles de Gaulle. Il s'agit d'une bifurcation complexe articulée autour de deux jonctions entre la ligne banalisée Paris-province et deux lignes dédiées, l'une aux trains grande vitesse, l'autre aux trains de fret.

Ce nœud voit passer un trafic voyageurs important partant de la gare de Paris-Nord. C'est un trafic de type « grande vitesse » vers le nord (Lille, Londres, Bruxelles, etc.), de type « grandes lignes » vers les villes comme Saint-Quentin, Maubeuge, Amiens et de type « régional » vers Compiègne, ou encore Saint-Just-en-Chaussée. Le matériel roulant est divers et varié, qu'il s'agisse des trains à grande vitesse (type TGV, Thalys, Eurostar, abrégé par HST dans la suite), ou des trains classiques (type grandes lignes ou régionaux, abrégé par IC/IR). Cette variété s'observe de par la composition du train en nombre de voitures ou rames, la longueur du convoi, les performances de la motrice, les équipements spécifiques embarqués, etc. Le nœud voit aussi passer un trafic fret en transit entre le nord et le sud de la région parisienne. Ces trains empruntent « la grande ceinture », la ligne qui contourne Paris et qui est dédiée à ce trafic. Le matériel

roulant est fortement hétérogène de par la fonction et l'origine des wagons qui composent un train. Pour être exhaustif, le nœud voit également passer un trafic voyageur « local » (type RER) et abrite plusieurs gares RER. Cependant, ce trafic circule sur des voies qui lui sont réservées et par conséquent il ne se mélange pas aux autres circulations en situation d'exploitation nominale. Il n'est donc pas repris dans la suite de la présente étude.

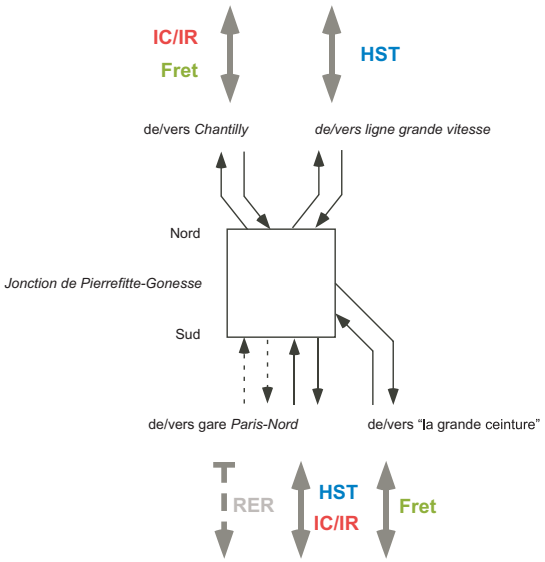


Figure 9.12. *Description synthétique du nœud ferroviaire de Pierrefitte-Gonesse et des principales circulations qui le fréquentent*

La figure 9.12 décrit la topologie du nœud et les circulations usuelles qui le fréquentent. Le trafic est qualifié dans le nœud selon sa circulation ; il est dans le sens pair quand le train circule de la province vers Paris, et impair depuis Paris vers la province. S'agissant d'un nœud de passage pour le trafic considéré, un train sur cette infrastructure n'a *a priori* aucune raison de marquer l'arrêt en marche normale. Dans la suite, les trains seront considérés avec une marche dans le meilleur des cas, c'est-à-dire à vitesse maximale autorisée et tous les feux au vert sur leur parcours. Pour traverser le nœud, un HST dispose de cinq parcours possibles dans le sens pair, contre six dans le sens impair. Un IC/IR dispose de cinq et huit parcours possibles pour respectivement les sens pair et impair. La durée de traversée du nœud diffère selon le parcours suivi par un train. Ces données précises sont élaborées pour les besoins de l'étude soit *via* des logiciels de simulation ferroviaire, reproduisant très fidèlement toute la dynamique du système (performances du train, signalisation, géométrie des voies, etc.), soit *via* des bases de données enregistrant en temps réel les informations de circulation prélevées au fil de l'infrastructure.

Contrairement à l'organisation du RER, on assiste à une mixité des circulations dans le nœud entre des trains de fret et les autres types de trains de voyageurs. Cette mixité se traduit par des parties de l'infrastructure communes à différents parcours. En première approche, une circulation de voyageurs constitue un train « court et rapide » alors qu'une circulation de fret est un train « long et lent ». Du fait que les parcours de ces circulations se cisaillent dans la jonction, des gènes entre trains peuvent apparaître, avec des répercussions directes sur la capacité du nœud en nombre de trains.

9.4.2. *Horaire de base considéré pour l'étude*

L'instance est construite sur la base de la circulation réelle relevée le 7 février 2008, pour la fenêtre de temps allant de 18h00 à 19h00. La période étudiée reprend tous les trains au départ et à l'arrivée de la gare de Paris-Nord qui transitent par le nœud de Pierrefitte-Gonesse dans cette fenêtre de temps. Cette circulation est composée de trains HST et de trains IC/IR.

La figure 9.13 représente la grille horaire de base de ce trafic au passage entre 18h00 et 19h00 sur le nœud (pour une question de facilité de représentation graphique, les accélérations des trains en dehors du nœud ont été négligées, en prenant l'hypothèse d'une évolution linéaire du train). Il s'agit d'une grille horaire « idéale », du fait que tous les trains sont considérés *a priori* comme empruntant le parcours de durée minimale. Le tableau 9.2 précise les 26 trains qui traversent le nœud au cours de la fenêtre de temps considérée. Ils se ventilent en seize trains dans le sens pair contre dix trains dans le sens impair.

Trois études sont décrites dans la suite. Les tailles d'instances traitées dans ces deux premières études sont très petites, les temps de résolution sont de l'ordre de la seconde. La troisième étude illustre l'aptitude qu'a l'algorithme à traiter des instances de grandes tailles.

9.4.3. *Etude 1. Faisabilité de l'horaire de base*

La première étude traite la question de la faisabilité de l'horaire de base. Partant de l'horaire idéal, sans décalage autorisé sur les heures d'entrées et sans préférence de parcours, est-il possible de planifier tous les trains ? Le problème à optimiser résultant de la situation est un USPP (*Unicost Set Packing Problem*) qui comporte 162 variables binaires et 228 contraintes de *set packing*. Sur cette taille d'instance, aucune difficulté n'est attendue. L'algorithme ACF planifie les 26 trains en quelques secondes et procure une liste de cent solutions (grilles horaires) équivalentes du point de vue du nombre de trains planifiés. Une solution de cette liste indique que seize parcours sont utilisés pour faire passer les 26 trains prévus.

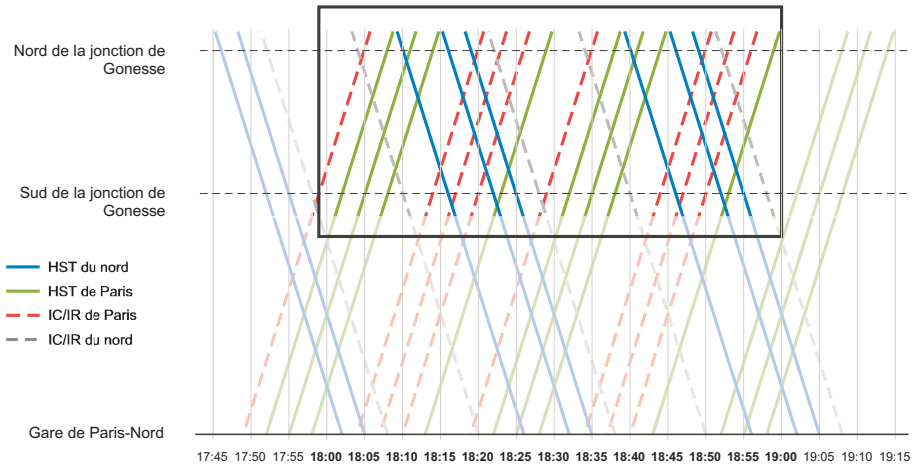


Figure 9.13. *Grille horaire pour une circulation sur le nœud de Pierrefitte-Gonesse aux heures de pointe*

Heure	Type	Identifiant	Destination
17 :59	IC/IR	RE 48579	Amiens
18 :02	HST	TGV 7343	St-Omer
18 :05	HST	THA 9453	Köln Hbf
18 :08	HST	TGV 7277	Tourcoing
18 :14	IC/IR	12337	St-Quentin(Aisne)
18 :17	IC/IR	12037	Amiens
18 :20	IC/IR	RE 48535	St-Just-en-Chaussée
18 :23	HST	EST 9051	London St. Pancras
18 :29	IC/IR	RE 47853	Compiègne
18 :32	HST	TGV 7145	Valenciennes
18 :35	HST	THA 9355	Amsterdam / Oostende
18 :38	HST	TGV 7281	Dunkerque
18 :44	IC/IR	12039	Amiens
18 :47	IC/IR	12339	Maubeuge
18 :50	IC/IR	RE 47857	Compiègne
18 :53	HST	EST 9053	London St. Pancras

Heure	Type	Identifiant	Provenance
18 :04	IC/IR	12036	Amiens
18 :10	HST	TGV 7144	Valenciennes
18 :16	HST	TGV 7072	Lille Flandres
18 :19	HST	THA 9346	Bruxelles-Midi
18 :22	IC/IR	RE 47850	Compiègne
18 :34	IC/IR	12334	St-Quentin(Aisne)
18 :40	HST	EST 9036	London St. Pancras
18 :46	HST	TGV 7074	Lille Flandres
18 :49	HST	THA 9448	Köln Hbf
18 :52	IC/IR	RE 48536	St-Just-en-Chaussée

Tableau 9.2. *Liste de trains en circulation dans le nœud (heure d'entrée h_{in} ; type de train ; identifiant du train ; destination/provenance)*

A défaut d'un critère additionnel, rien ne permet de discerner les solutions équivalentes entre elles. En revanche, une notion de stabilité des grilles horaires, comme proposée dans [DEL 09], réduit souvent considérablement le nombre de solutions. Une alternative à un critère de stabilité serait de tenter de planifier les trains autant que possible sur les parcours les plus directs. Par exemple, pour la solution mentionnée ci-dessus, on observe des utilisations de l'infrastructure telles que celle illustrée par la figure 9.14 pour le TGV 7343 en provenance de Paris. Une telle mobilisation de l'infrastructure limite la possibilité de croisement avec un autre train en direction de

Paris, du fait de son passage sur le bas de l'infrastructure. Or, si celui-ci peut être planifié sur le parcours le plus direct, il réduit la gène engendrée sur les autres parcours et reste moins de temps sur l'infrastructure. Afin de prendre en compte ce souhait, un deuxième critère traduisant le temps de parcours utilisé est associé à chaque variable du SPP comme suit :

$$c_i^2 = 1 + \max\{\text{temps de parcours pour le type de train et le sens considéré}\} - \text{temps de parcours correspondant à la variable } x_i$$

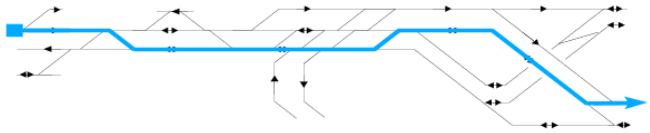


Figure 9.14. *Le TGV 7343 en provenance de Paris-Nord et à destination de Saint-Omer entre dans le noeud de Pierrefitte-Gonesse à 18h02 et emprunte un parcours indirect pour rejoindre la ligne grande vitesse*

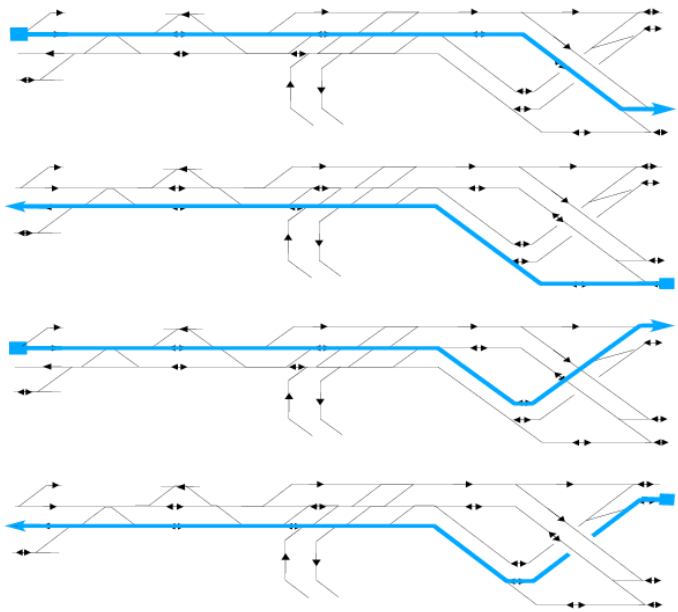


Figure 9.15. *Quatre parcours directs pour trains voyageur retenus dans la solution produite par l'algorithme ACF. De haut en bas, les parcours directs de Paris-Nord de/vers la ligne grande vitesse et de/vers la ligne classique*

Ce problème lexicographique (avec le premier critère correspondant à la faisabilité de la grille horaire initiale), peut être résolu par l'algorithme ACF en considérant la fonction objectif suivante :

$$c_i = M + c_i^2$$

avec M une constante égale à $\sum_{i \in I} c_i^2$. Le problème à optimiser résultant de la situation est un WSPP (*Weighted Set Packing Problem*) de taille identique à la situation précédente. L'algorithme ACF planifie les 26 trains en quelques secondes et procure une solution utilisant seulement quatre parcours. L'examen de ces parcours montre qu'il s'agit bien des parcours voyageur les plus directs sur ce nœud ferroviaire (figure 9.15). Il enchaîne cependant les trains sur ces parcours directs en utilisant parfois complètement la ressource disponible (aucune marge entre certains trains). Bien que séduisante du point de vue du routage des trains, une telle solution peut s'avérer instable, car le retard d'un train peut la rendre caduque.

L'étude suivante examine la faculté qu'a le nœud d'absorber des trains supplémentaires pour cette grille horaire.

9.4.4. *Etude 2. Intégration de trains de fret dans l'horaire de base*

Le modèle d'optimisation offre la possibilité d'envisager une déviation δ de valeur positive ou négative sur l'heure prévue d'entrée d'un train dans le nœud, h_{in}. Cette flexibilité va être exploitée dans cette seconde étude, qui aborde une première forme de saturation d'un horaire de base existant, en essayant d'ajouter des trains de fret, sur une grille horaire donnée. La question à traiter s'énonce dans les termes suivants ; partant de l'horaire idéal, sans préférence de parcours, sans décalage autorisé sur les heures d'entrées des trains de l'horaire de base, mais avec un décalage autorisé des heures d'entrées des trains fret, est-il possible de planifier tous les trains ?

Dans le cas présent, étant donné l'horaire particulièrement serré des trains circulant à ce moment de la journée, il n'est pas réaliste d'envisager de grandes fenêtres de temps en vue d'injecter des trains fret. De telles fenêtres sont donc définies pour chaque train fret en considérant un ensemble de valeurs discrètes (minutes). En pratique, pour une circulation de train t donnée (c'est-à-dire un train, une heure d'entrée, un sens de circulation, un point d'entrée et un point de sortie dans la bifurcation), les déviations $\delta \in \Delta_t$ généreront l'ensemble des variantes possibles pour construire l'instance de *set packing* représentative de l'horaire de base.

L'ajout de cinq trains fret va être tenté, deux dans le sens pair, trois dans le sens impair dans les fenêtres de temps données dans le tableau 9.3. Le problème à optimiser résultant de la situation est un SPP à coûts unitaires qui comporte 311 variables binaires et 489 contraintes de *set packing*. Sur cette taille d'instance, aucune difficulté de résolution n'est attendue. L'algorithme ACF planifie les 31 trains en quelques secondes et procure une liste de cent solutions (grilles horaires) équivalentes du point

de vue du nombre de trains planifiés. Une solution de cette liste indique que 18 parcours sont utilisés (treize et cinq respectivement pour les trains voyageurs et fret) pour faire passer les 31 trains prévus. Le tableau 9.3 indique les heures d'entrée planifiées à l'issue de l'exécution de l'algorithme de fourmis.

sens	train	entrée [au plus tôt ; au plus tard]	entrée planifiée
pair	$fret_1$	[18 :23 ; 18 :29]	18 :28
pair	$fret_2$	[18 :38 ; 18 :44]	18 :40
impair	$fret_3$	[18 :00 ; 18 :04]	18 :01
impair	$fret_4$	[18 :22 ; 18 :34]	18 :25
impair	$fret_5$	[18 :22 ; 18 :34]	18 :29

Tableau 9.3. *Liste des trains fret à intégrer à la grille horaire de base et heure d'entrée planifiée à l'issue de l'exécution de l'algorithme de fourmis*

L'insertion de trains fret conduit à un horaire réalisable faisant passer tous les trains sur le nœud de Pierrefitte-Gonesse sans dépasser 19h, ni modifier l'horaire de base correspondant aux trains voyageurs : la capacité résiduelle permet une telle insertion. Une variante pourrait s'attacher à mesurer le nombre maximum de trains fret pouvant être injectés dans le nœud pour mesurer cette capacité résiduelle. L'étude suivante s'attache à la problématique de saturation sans horaire de base disponible, correspondant à la mesure de la capacité absolue.

9.4.5. *Etude 3. Mesure de la capacité absolue du nœud ferroviaire*

Les deux premières études illustrent l'aptitude de l'ACF à traiter indifféremment une variété de questions ferroviaires lui conférant une certaine flexibilité. Toutefois, l'intérêt premier de cette technique de résolution est d'être en mesure de traiter efficacement des instances de grandes tailles (ce qui n'est pas illustré dans les deux premières études). Les grandes instances se trouvent principalement dans les études de saturation. Cette troisième étude illustre cette faculté de l'algorithme à la lumière de seize instances de saturation. Chaque instance représente une configuration particulière du trafic (exemple TGV et fret) sans horaire de base.

Le tableau 9.4 synthétise les résultats (valeurs moyennes) collectés sur seize exécutions de l'algorithme ACF (pour un calculateur type Apple PowerBook équipé d'un processeur PowerPC G4 1GHz, de mémoire RAM 512Mb et du système d'exploitation Mac OS X 10.3.9), chaque exécution correspondant à une graine différente au niveau du générateur de nombres aléatoires. Le nombre de trains présents dans la liste saturante ($|T_{LS}|$), ainsi que le nombre de variables et de contraintes du SPP résultant, y sont aussi indiqués. A noter que plusieurs de ces instances n'ont pas pu être résolues à l'*optimum* avec un logiciel comme Cplex (version 8.2) dans un temps raisonnable

(ordre de la journée de temps de calcul, ce qui est déjà beaucoup trop pour une utilisation pratique). Les résultats sont donc donnés par rapport à la meilleure solution connue (obtenue soit avec Cplex, soit avec GRASP, soit avec ACF) pour l'instance.

Instances				ACF	
nom	# trains	# variables	# contraintes	Qualité (%) moy	CPUt (s) moy
Gon01	90	465	8 661	100,00	10,38
Gon02	120	620	11 676	100,00	25,50
Gon03	240	1 240	23 736	100,00	175,00
Gon04	240	1 240	50 705	100,00	80,62
Gon07	380	1 620	79 514	99,44	187,69
Gon05	360	1 860	55 938	99,34	485,44
Gon06	360	1 860	119 009	98,55	258,44
Gon14	150	2 160	140 632	99,63	360,33
Gon10	150	2 400	186 861	100,00	407,67
Gon15	130	2 503	185 523	98,39	468,67
Gon11	125	2 683	228 972	99,62	606,00
Gon12	200	2 880	311 228	98,12	635,00
Gon13	157	3 210	380 292	99,26	734,67
Gon08	720	3 720	155 985	100,00	2 256,00
Gon09	720	3 720	482 887	97,62	689,00

Tableau 9.4. *Résultats de saturation*

En ce qui concerne la qualité des solutions, ACF est en moyenne à moins de 2,5% de la meilleure solution connue pour toutes les instances, et a produit sur plusieurs instances les meilleures valeurs connues. Pour rappel, l'algorithme ACF ne demande aucun réglage de ses constantes. En ce qui concerne les temps de calcul, et vu qu'il s'agit d'un problème d'optimisation de niveau stratégique, les valeurs obtenues sont tout à fait acceptables. Ces expérimentations numériques réalisées montrent que l'algorithme monte en charge d'une manière très satisfaisante pour des situations conséquentes.

9.5. Conclusions et perspectives

L'algorithme ACF mis au point dans le cadre de cette étude appliquée apparaît comme un algorithme simple et autoparamétré. Il traite un ensemble de problèmes d'optimisation rencontrés en exploitation ferroviaire, se formulant comme un *set packing*. Il comporte trois caractéristiques : une stratégie de perturbation des phéromones, une stratégie d'autoparamétrage, qui gère la convergence de l'algorithme, et un double mode de construction de solutions. Il fait appel à un ensemble de recherches locales *ad hoc*, au regard du problème de *set packing* à traiter.

Le modèle d'optimisation considéré par l'algorithme ACF permet de traiter avec une grande flexibilité les heures d'entrée des trains, ainsi que les parcours autorisés

dans l'infrastructure. Il permet aussi de manipuler des préférences sur les parcours empruntés, différentes catégories de trains saturants et de traiter les contraintes particulières rencontrées dans les gares (arrêt aux quais de capacité suffisante par exemple). Les réponses fournies concernent la mesure de la capacité (faisabilité, saturation résiduelle/absolue) mais aussi permettent d'analyser l'utilisation de l'infrastructure (zones peu/très fréquentées). L'algorithme ACF traite parfaitement les instances représentatives de ces différentes situations, avec des performances tout à fait remarquables, même pour des problèmes de très grande taille.

L'algorithme ACF a ainsi pu être intégré en tant que solveur dans un logiciel possédant d'autres fonctionnalités non décrites dans le présent document : un module de calcul de stabilité, un processus d'aide multicritère à la décision, des indicateurs statistiques supports au système d'aide à la décision.

Parmi les perspectives possibles, l'exploitation de la structure particulière du problème ferroviaire présent dans le *set packing*, pour espérer encore gagner en efficacité de résolution, est une piste envisageable. Sur le plan ferroviaire, l'ensemble est appelé à être appliqué pour mener des études de capacité sur la gare de Lille-Flandres.

9.6. Remerciements

Ces travaux ont bénéficié du soutien financier du Conseil régional Nord-Pas-de-Calais et des fonds FEDER de l'Union européenne. Les auteurs remercient la Direction de la recherche et de la technologie de la SNCF pour les données fournies, qui ont permis de réaliser les expérimentations présentées. Les travaux ont été menés en collaboration avec la SNCF (direction régionale des infrastructures, Lille), l'INRETS (unité ESTAS, Villeneuve d'Ascq), l'Université de Nantes, l'Ecole nationale supérieure des Mines de Saint-Etienne et l'Université de Valenciennes et du Hainaut Cambrésis.

9.7. Bibliographie

[BER 94] Van den Berg J., Odijk M., « DONS : computer aided design of regular service timetables », *Computer in Railways (COMPRAIL)*, Madrid, 1994.

[BOR 98] Borndörfer R., Aspects of set packing, partitioning, and covering, PhD thesis, Fachbereich 3 Mathematik der Technischen Universität Berlin, Berlin, 1998.

[COR 01] Cornuéjols G., *Combinatorial optimization : packing and covering*, CBMS-NSF regional conference series in applied mathematics, Philadelphia : Society for Industrial and Applied Mathematics, 2001.

[DEG 07] Degoutin F., Modélisation par contraintes et heuristiques pour l'évaluation de la capacité d'infrastructures ferroviaires, thèse de Doctorat, Université de Valenciennes et du Hainaut Cambrésis, Valenciennes, 2007.

[DEL 03] Delorme X., Modélisation et résolution de problèmes liés à l'exploitation d'infrastructures ferroviaires, thèse de Doctorat, Université de Valenciennes et du Hainaut Cambrésis, Valenciennes, 2003.

[DEL 04] Delorme X., Gandibleux X., Rodriguez J., « GRASP for Set Packing Problems », *European Journal of Operational Research*, vol. 153, n° 3, p. 564–580, 2004.

[DEL 09] Delorme X., Gandibleux X., Rodriguez J., « Stability evaluation of a railway timetable at station level », *European Journal of Operational Research*, vol. 195, n° 3, p. 780–790, 2009.

[DOR 99] Dorigo M., Di Caro G., Gambardella L. M., « Ant algorithms for discrete optimization », *Artificial Life*, vol. 5, n° 3, p. 137–172, 1999.

[FEO 89] Féo T. A., Resende M., « A probabilistic heuristic for a computationally difficult set covering problem », *Operations Research Letters*, vol. 8, p. 67–71, 1989.

[FEO 95] Féo T. A., Resende M. G., « Greedy Randomized Adaptative Search Procedures », *Journal of Global Optimization*, vol. 6, p. 109–133, 1995.

[GAN 04] Gandibleux X., Delorme X., T'Kindt V., « An Ant Colony Algorithm for the Set Packing Problem », M. Dorigo, M. Birattari, C. Blum, L. Gambardella, F. Mondada, T. Stutzle (dir.), *Ant Colony Optimization and Swarm Intelligence*, vol. 3172 de *Lecture Notes in Computer Sciences*, p. 49–60, Springer, 2004.

[GAN 05] Gandibleux X., Jorge J., Angibaud S., Delorme X., Rodriguez J., « An ant colony optimization inspired algorithm for the Set Packing Problem with application to railway infrastructure », *MIC2005 : The Sixth Metaheuristics International Conference*, Vienne, Autriche, août 2005.

[GAN 08] Gandibleux X., Riteau P., Delorme X., « RECIFE : a MCDSS for railway capacity », *MCDM 2008*, Auckland, Nouvelle-Zélande, 2008.

[GAR 79] Garey M. R., Johnson D. S., *Computers and intractability : a guide to the theory of NP-Completeness*, V.H. Freeman and Company, San Francisco, 1979.

[HAC 97] Hachemane P., Evaluation de la capacité de réseaux ferroviaires, thèse, Ecole Polytechnique Fédérale de Lausanne, Lausanne, 1997.

[JOR 06] Jorge J., Gandibleux X., « Self-Adaptive Stopping Condition for an Ant Colony Optimization Inspired Algorithm Designed for Set Packing Problems », *7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics*, 2006.

[NEM 99] Nemhauser G. L., Wolsey L. A., *Integer and combinatorial optimization*, Wiley-Interscience, New York, 1999.

[PIR 02] Pirlot M., « Métaheuristiques pour l'optimisation combinatoire : un aperçu général », J. Teghem, M. Pirlot (dir.), *Optimisation approchée en recherche opérationnelle - Recherches locales, réseaux neuronaux et satisfaction de contraintes*, p. 25–55, Hermès Science Publications, Paris, 2002.

[ROD 07] Rodriguez J., Delorme X., Gandibleux X., Marlière G., Bartusiak R., Degoutin F., Sobieraj S., « RECIFE : modèles et outils pour l'analyse de la capacité ferroviaire », *Recherche Transports Sécurité*, vol. 95, p. 19–36, 2007.

[SCH 08] Schlechte T., Borndörfer R., « A suitable Model for a bicriteria Optimization Approach to Railway Track Allocation », *MCDM 2008*, Auckland, Nouvelle-Zélande, 2008.

[STU 98] Stützle T., « An ant approach for the flow shop problem », *Proceedings of EUFIT'98*, p. 1560–1564, Aix-la-Chapelle, Allemagne, 1998.

[UIC 04] UIC, Fiche 406/R–La capacité, 2004.

[VEL 05] Velásquez R., Ehrgott M., Ryan D. M., Schöbel A., « A Set-packing Approach to Routing Trains through Railway Stations », *ORSNZ 2005*, Wellington, Nouvelle-Zélande, 2005.

[ZWA 96] Zwaneveld P. J., Kroon L. G., Romeijn H. E., Salomon M., Dauzère-Pérès S., van Hoesel S. P., Ambergen H. W., « Routing trains through railway stations : Model formulation and algorithms », *Transportation Science*, vol. 30, n° 3, p. 181–194, 1996.

[ZWA 97] Zwaneveld P. J., Railway planning - routing of trains and allocation of passenger lines, PhD thesis, Rotterdam school of management, TRAIL research school, Rotterdam, 1997.

Chapitre 10

Les fourmis pour la segmentation d'images médicales par résonance magnétique

Dans ce chapitre, nous présentons une nouvelle méthode de segmentation d'images par résonance magnétique (IRM), basée sur le critère de variance intraclasse modifiée (MVar). Le critère $MVar$ prend en compte non seulement l'information sur les niveaux de gris, mais aussi la distribution spatiale des niveaux de gris, en exploitant les surfaces des régions segmentées. La complexité algorithmique qui en résulte constitue une difficulté, notamment dans les applications temps réel. Afin de pallier ce problème, nous utilisons une métaheuristique basée sur un algorithme de colonie de fourmis. Les résultats obtenus sur des images IRM montrent que la méthode proposée permet d'avoir une segmentation satisfaisante, moyennant un faible temps d'exécution.

10.1. Introduction

La segmentation d'image occupe une place importante dans les systèmes de traitement d'image. Toute erreur faite au stade de la segmentation, située en début de chaîne, peut entacher de façon irrémédiable toute la suite des traitements : extraction de primitives et codage, classification, reconnaissance d'objets, analyse et interprétation de scènes. Pour améliorer les performances d'un système de traitement d'image, l'optimisation de la segmentation est donc une étape cruciale. Pour preuve, la quantité de travaux dédiés à la segmentation est aujourd'hui considérable. Pour les méthodes les plus répandues, on peut trouver un état de l'art dans [SEZ 04]. Parmi ces méthodes, le seuillage d'image par analyse d'histogramme est sans doute l'approche la plus facile

Chapitre rédigé par Amir Nakib, Raphaël Blanc, Hamouche Oulhadj et Patrick Siarry.

à mettre en œuvre et à implémenter [COC 95]. Cette approche repose sur l'hypothèse que les différentes classes d'objets à segmenter peuvent être distinguées par leur niveau de gris. Comme les relations spatiales entre les pixels de l'image sont ignorées, la seule information exploitée est la distribution des densités de niveaux de gris. Faisant abstraction de l'information locale de voisinage, qui nécessite de visiter chaque parcelle de l'image, l'approche induit un gain en vitesse de convergence important, ce qui explique son efficacité, notamment dans les environnements temps réel, où le délai de traitement représente une donnée à optimiser.

Dans le cas d'un histogramme en N classes, la segmentation d'image par seuillage consiste à déterminer $N-1$ seuils de façon à ce que chaque classe soit associée à un intervalle distinct de niveaux de gris. En notant G l'ensemble des niveaux de gris de l'image, $[T_0T_1], ..., [T_{k-1}T_k], ..., [T_{N-1}T_N]$, l'ensemble des intervalles de niveaux de gris correspondant aux différents seuils de segmentation considérés, cette opération peut se résumer par l'application :

$$X \rightarrow T(X)/X = k \quad si \quad X \in \left[\begin{array}{cc} T_k & T_{k+1} \end{array} \right] \tag{10.1}$$

où $X \in G$, $T(X) \in \{0, ..., N\}$ et N désigne le nombre de classes.

Dans le cas d'un seuillage simple, appelé aussi binarisation, les nouvelles valeurs attribuées aux pixels de l'image appartiennent à l'ensemble $[0, 1]$. Dans la littérature, plusieurs classifications des méthodes de seuillage ont été proposées [CHE 98, HOU 06, KAP 85, OTS 79, PAL 91, SEZ 04]. Cependant, la plupart des auteurs classent ces méthodes en deux catégories : les méthodes paramétriques et les méthodes non paramétriques. Les méthodes paramétriques sont basées sur l'hypothèse que les densités de probabilité des niveaux de gris des différentes classes suivent des modèles gaussiens. En partant d'une approximation de l'histogramme par une combinaison de gaussiennes, dont les paramètres sont bien déterminés, les seuils optimaux de segmentation sont à l'intersection des gaussiennes. Ce type de méthode n'est efficace que lorsque l'histogramme est multimodal, c'est-à-dire comportant plusieurs pics significatifs, représentant des points d'ancrage pour les différentes gaussiennes. Cependant, dans la pratique, ce cas de figure ne se présente pas fréquemment, ce qui rend la segmentation problématique, chaque fois que l'histogramme de l'image est unimodal. En revanche, les méthodes non paramétriques ne supposent aucun *a priori* sur le profil de l'histogramme. Généralement, elles se réduisent à minimiser un critère statistique et fonctionnent indifféremment sur des histogrammes multi ou unimodal. Dans son article original [OTS 79], N. Otsu décrit trois critères discriminants possibles : la variance intraclasse, la variance interclasse et la variance totale. Les trois critères sont équivalents et, suivant la situation, l'un des trois peut être utilisé.

Pour minimiser un critère de segmentation, une solution classique consiste à faire appel à l'algorithme de la descente du gradient. On converge vers le minimum de la

fonction, en suivant la ligne de la plus grande pente du gradient. Toutefois, cet algorithme n'est efficace que dans le cas d'une binarisation. Lorsque le nombre de classes est plus important, la fonction à minimiser peut présenter plusieurs minimums locaux et rien ne garantit que le minimum trouvé corresponde au minimum global. En effet, l'algorithme s'arrête dès que la pente du gradient ne diminue plus de façon sensible, ce qui se produit dès que l'on rencontre une vallée ou un plateau. De plus, dans le cas d'une segmentation en plusieurs classes, l'espace des solutions peut s'avérer trop vaste pour que l'on puisse l'explorer de façon exhaustive dans des délais raisonnables. Pour pallier tous ces problèmes, le recours à une métaheuristique devient incontournable. Là où les méthodes classiques peuvent faillir, ce type d'algorithme est capable de procurer en principe une solution sous optimale acceptable, à l'issue d'une série d'itérations n'explorant qu'un sous-espace réduit de l'espace total des solutions. D'où l'intérêt des métaheuristiques pour résoudre le problème de la segmentation d'image en plusieurs classes, quel que soit l'histogramme de l'image, multi ou unimodal. Nous avons étudié et adapté au problème de la segmentation d'image plusieurs métaheuristiques [NAK 07a, NAK 07b]. Nous décrivons dans ce chapitre les résultats obtenus à l'aide d'un algorithme de colonie de fourmis.

Nous nous sommes intéressés à la segmentation d'images biomédicales et plus particulièrement des images obtenues *via* l'imagerie par résonance magnétique (IRM).

Cette application présente un intérêt important en pratique. En effet, le clinicien est confronté au quotidien à des problèmes d'analyse d'images, qui peuvent lui prendre un temps précieux. Sa priorité est d'établir des diagnostics fiables, dans des délais raisonnables. C'est en parvenant à automatiser le processus de segmentation et d'analyse d'image que l'on fournira au clinicien des outils de diagnostic ergonomiques et fiables, qui pourront l'aider à choisir les thérapeutiques appropriées aux maladies traitées.

La suite de ce chapitre est organisée de la façon suivante. Dans la section 10.2, nous rappelons les principes de l'imagerie par résonance magnétique. Dans la section 10.3, nous présentons la problématique des images biomédicales traitées et l'intérêt pratique de nos résultats en milieu clinique. La section 10.4 est consacrée à la description du critère de segmentation développé. Nous détaillons dans cette partie les idées qui nous ont servi de trame pour construire ce critère, puis définir la fonction objectif à minimiser *via* l'algorithme de colonie de fourmis. La section 10.5 est focalisée sur la présentation de la méthode de segmentation proposée. Nous indiquons les modifications que nous avons apportées pour l'adapter au problème de la segmentation d'image. Dans la section 10.6, nous présentons des résultats de segmentation, obtenus sur une série d'images biomédicales bien représentatives. Cette partie est complétée par une étude critique, dans laquelle nous mettons en valeur les points forts de notre méthode de segmentation et insistons sur les points qui peuvent faire l'objet d'une amélioration. Enfin, ce chapitre se termine par une conclusion, dans laquelle nous récapitulons les apports essentiels de notre étude et proposons des perspectives de développement susceptibles d'améliorer la méthode de segmentation d'image proposée.

10.2. Principe de l'imagerie par résonance magnétique

L'imagerie par résonance magnétique nucléaire (IRM ou RMN) est une technique d'imagerie médicale d'apparition récente, qui consiste à étudier les modifications d'aimantation des protons d'hydrogène d'une substance, sous l'action d'un champ magnétique. Elle repose sur les propriétés magnétiques des noyaux, que l'on étudie grâce à l'application d'un champ magnétique et d'une excitation par une onde de radiofréquence.

Dès la fin de l'excitation, un retour à l'état d'équilibre se produit. La localisation spatiale des atomes est obtenue en ajoutant un gradient directionnel au champ magnétique de base. La relaxation des protons est alors modifiée par la variation du champ magnétique. Les techniques de traitement du signal utilisant des algorithmes de transformées de Fourier rapides permettent alors de localiser l'événement. En modifiant les paramètres d'acquisition IRM, notamment le temps de répétition entre deux excitations et le temps d'écho (temps entre le signal d'excitation et la réception de l'écho), l'utilisateur peut différencier les tissus de l'organisme : on sait en effet que des tissus différents ont des caractéristiques IRM différentes.

L'IRM a l'avantage d'apporter une bonne visualisation de l'eau, donc de l'œdème et de l'inflammation, avec une bonne résolution et un bon contraste. Les tissus graisseux sont également bien explorés par cette technique. *A contrario*, cette imagerie est moins adaptée à l'étude des tissus pauvres en protons, comme les tissus osseux.

10.3. Importance de la segmentation en IRM

Dans l'évolution actuelle de l'imagerie médicale diagnostique, et notamment de la technique d'IRM, on peut distinguer deux grands types d'imageries : une imagerie descriptive, anatomique (IRM morphologique), et une imagerie de la fonction de l'organe étudié (IRM fonctionnelle).

En routine clinique, l'imagerie morphologique est la plus utilisée, mais elle correspond de plus en plus à un socle minimum d'interprétation, qui est complété, parfois dans un second temps (pour des raisons de durée d'examen), par des techniques plus avancées, comme la spectroscopie, l'IRM fonctionnelle, ou l'imagerie de perfusion. Que ce soit ces différentes techniques ou l'imagerie morphologique classique, ces approches requièrent la segmentation des images sources, pour permettre une quantification précise des données acquises et de leur évaluation.

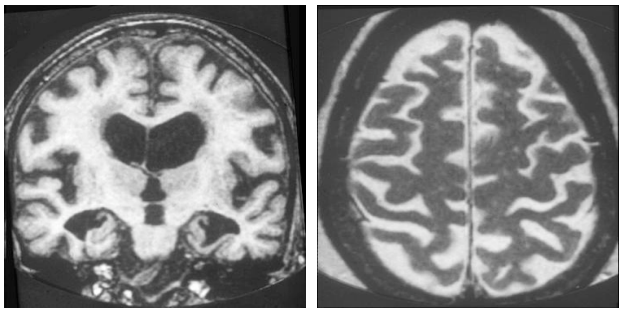
10.3.1. *Exemples de pathologies étudiées en IRM*

Il s'agit d'abord des pathologies dégénératives :

– les démences (maladie d'Alzheimer : atrophie corticale prédominant sur le cortex parétio-occipito-temporal) : intérêt de la segmentation, pour mettre en évidence la diminution du volume cérébral, parfois en orientant l'examen sur certaines régions spécifiques (au niveau du lobe temporal, l'hippocampe) ;

– la dégénérescence des noyaux gris centraux (maladies de type Parkinson) : de même, l'IRM et la volumétrie des noyaux sous-thalamiques sont utilisées pour juger de l'état de gravité et d'évolution de ces maladies.

L'IRM permet d'étudier aussi les pathologies des espaces liquidiens cérébraux : autour (espaces sous-arachnoïdiens) et à l'intérieur (système ventriculaire) du cerveau, se trouve un liquide, le liquide cérébro-spinal. La quantité, la circulation et la répartition de ce liquide peuvent être affectées dans le cadre de nombreuses pathologies : notamment, le système ventriculaire peut être touché par des pathologies comme l'hydrocéphalie (communicante ou non) ou, plus récemment reconnue, l'hypotension intracrânienne. Là encore, la segmentation de ces différents espaces permet une étude précise de la volumétrie et des mouvements, ou de la circulation, au sein de ces différents compartiments. Deux exemples de pathologies sont présentés sur la figure 10.1. Dans la figure 10.1a, le patient présente une atrophie cortico sous-corticale. Le patient de la figure 10.1b présente une atrophie corticale. Ces deux pathologies sont des signes de la maladie d'Alzheimer.



(a) (b)

Figure 10.1. *Exemples de pathologies : (a) atrophie cortico sous-corticale, (b) atrophie corticale*

Les segmentations sont également intéressantes pour évaluer des volumes de charge lésionnelle dans des pathologies inflammatoires, comme la sclérose en plaque, ou des pathologies tumorales : elles permettent alors l'appréciation de l'extension de la zone tumorale, par rapport à la présence d'une réaction œdémateuse en périphérie.

En dehors de l'encéphale, l'IRM permet l'étude de la dégénérescence des tissus cutanés faciaux (lipodystrophie).

10.4. Critère de segmentation proposé : la variance intraclasse biaisée

Les modifications apportées à la variance intraclasse, utilisée dans la méthode de N. Otsu [OTS 79], consistent en l'introduction de paramètres de correction, qui caractérisent les propriétés d'une bonne segmentation [NAK 07a]. D'où l'utilisation d'un nouveau critère de segmentation, appelé variance intraclasse biaisée. Mathématiquement, ce critère s'écrit :

$$MVar\,(t_1, ..., t_{N-1}) = \alpha . \sum_{j=1}^{N} \left(\beta_j^{-1} . \sigma_{int}^2(j) + \gamma_j\right) \tag{10.2}$$

où N représente le nombre de classes à segmenter (supposé connu *a priori*), $\sigma_{int}^2(j)$ la variance intraclasse de la classe j, t_1 à t_{N-1} les seuils de segmentation sélectionnés, et β_j et γ_j les paramètres de correction de la variance intraclasse utilisée dans la méthode Otsu.

α est égal à $(1000 \times M)^{-1} \sqrt{NR}$, où M est le nombre total de pixels dans l'image et NR le nombre de régions, c'est-à-dire le nombre de composantes connexes dans l'image segmentée. Ce terme est utilisé pour normaliser et pénaliser les images sursegmentées.

Le terme $\beta_j^{-1} = (1 + logN_j)$ est choisi de telle sorte que le terme $\beta_j^{-1}\sigma_{int}^2(j)$ soit faible pour les grandes classes. N_j est le nombre de pixels dans la classe j.

Le terme $\gamma_j = (C(N_j)/N_j)$ est élevé lorsque l'image segmentée comporte beaucoup de régions de même taille, surtout si elles sont petites. $C(N_j)$ est le nombre de régions dont le cardinal est égal à N_j. Pour les grandes régions, $C(N_j)$ est, dans la plupart des cas, égal à 1, alors que, pour les régions de petites tailles, il devient supérieur à 1.

Dans tous les cas, le paramètre N_j force γ_j à être proche de 0 pour les grandes régions. Tous ces paramètres sont inspirés du critère de M. Borsotti [BOR 98], utilisé pour l'évaluation de la qualité de la segmentation des images en couleur. Le problème de la segmentation d'une image revient à trouver les seuils optimaux maximisant le critère défini par l'expression (10.2) :

$$\begin{array}{l} MVar\left(\left(t_1^*, ..., t_{N-1}^*\right)\right) = Max\,\{MVar\,(t_1, ..., t_{N-1})\} \\ s.c. : 1 < t_1 < t_2 <, ..., < 255 \end{array} \tag{10.3}$$

10.5. Adaptation de ACO au problème de la segmentation d'image

Pour adapter l'algorithme ACO de M. Dorigo [BLU 05, DOR 96] au problème de la segmentation d'image, nous avons introduit une contrainte et défini des critères d'arrêt. La contrainte porte sur les intervalles de variation des variables : si une fourmi sort de son intervalle, sa densité de phéromone est remise à zéro.

Les critères d'arrêt sont au nombre de deux :

– le nombre maximum d'itérations : $Nbimax_1 = 100(N-1)$, où N est le nombre de classes ;

– le nombre maximum d'itérations n'induisant pas d'amélioration du dernier maximum enregistré : $Nbimax_2 = 200$.

La règle de déplacement, appelée probabilité de transition [BLU 05], est définie par :

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}(t)^\alpha . \eta_{ij}^\beta}{\sum_{l \in J_i^k} \tau_{il}(t)^\alpha . \eta_{il}^\beta} & si \quad j \in J_i^k \\ 0 & si \quad j \notin J_i^k \end{cases} \tag{10.4}$$

Une fois la fonction objectif évaluée ($MVar$), une fourmi k dépose une quantité $\Delta\tau_{ij}^k(t)$ de phéromone :

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{L^k(t)} & si\ (i,j) \in T^k(t) \\ 0 & si \quad (i,j) \notin T^k(t) \end{cases} \tag{10.5}$$

où $T^k(t)$ est la solution fournie par la fourmi k à l'itération t, $L^k(t)$ est la somme des seuils de segmentation contenus dans la solution.

Afin d'éviter d'être piégé par des solutions locales, on évapore de $\rho\tau_{ij}(t)$, à la fin de chaque itération, les phéromones déjà déposées et on procède à une mise à jour :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{10.6}$$

où m est le nombre de fourmis utilisées à l'itération t et ρ un paramètre de réglage. Au départ, la phéromone est initialisée par une petite quantité $\tau_0 \geq 0$.

La procédure présentée dans l'algorithme 10.1 résume les différentes étapes de l'algorithme de segmentation (MVAR-ACO), basé sur ACO et la maximisation de la variance intraclasse biaisée.

Le réglage des paramètres de l'algorithme a été fait de manière empirique. Les valeurs utilisées sont résumées dans le tableau 10.1.

Algorithme 10.1 Algorithme de segmentation MVAR-ACO

```
Tirer m fourmis aléatoirement.
tantque la condition d'arrêt n'est pas satisfaite faire
    pour chaque fourmi k = 1, ..., m faire
        Evaluer la fonction objectif donnée par la formule 10.2.
        Attribuer une probabilité de transition selon l'expression 10.4.
        Evaporer les pistes selon l'expression 10.6.
    fin pour
fin tantque
Afficher le meilleur état rencontré au cours de la recherche.
```

Paramètre	Valeur
Nombre de fourmis m	$50 \times N$
Cœfficient d'évaporation	$0,1$
α	1
β	5

Tableau 10.1. *Valeurs des paramètres de ACO*

10.6. Résultats et interprétation

La discussion porte sur la sélection des seuils optimaux, au sens de la maximisation du critère $MVar$, et la présentation de quelques images IRM. Nous ne présentons ici que les cas de quatre et cinq classes de segmentation. Les résultats procurés par MVAR-ACO sont comparés à ceux obtenus avec l'entropie 2D de C. Shannon (2DSE).

Une étude comparative des performances de MVAR-ACO avec celles d'autres méthodes tirées de la littérature est ensuite présentée, pour la segmentation d'images synthétiques.

10.6.1. *Résultats sur des images IRM*

Afin d'illustrer les résultats procurés par notre algorithme, deux images IRM du cerveau, avec leur classification multiniveau quand $N = 4$ et 5, sont présentées sur la figure 10.2. Les résultats dans le cas d'un sujet sain sont montrés sur les figures 10.2c et 10.2e ; ceux dans le cas d'une atrophie sont sur les figures 10.2d et f.

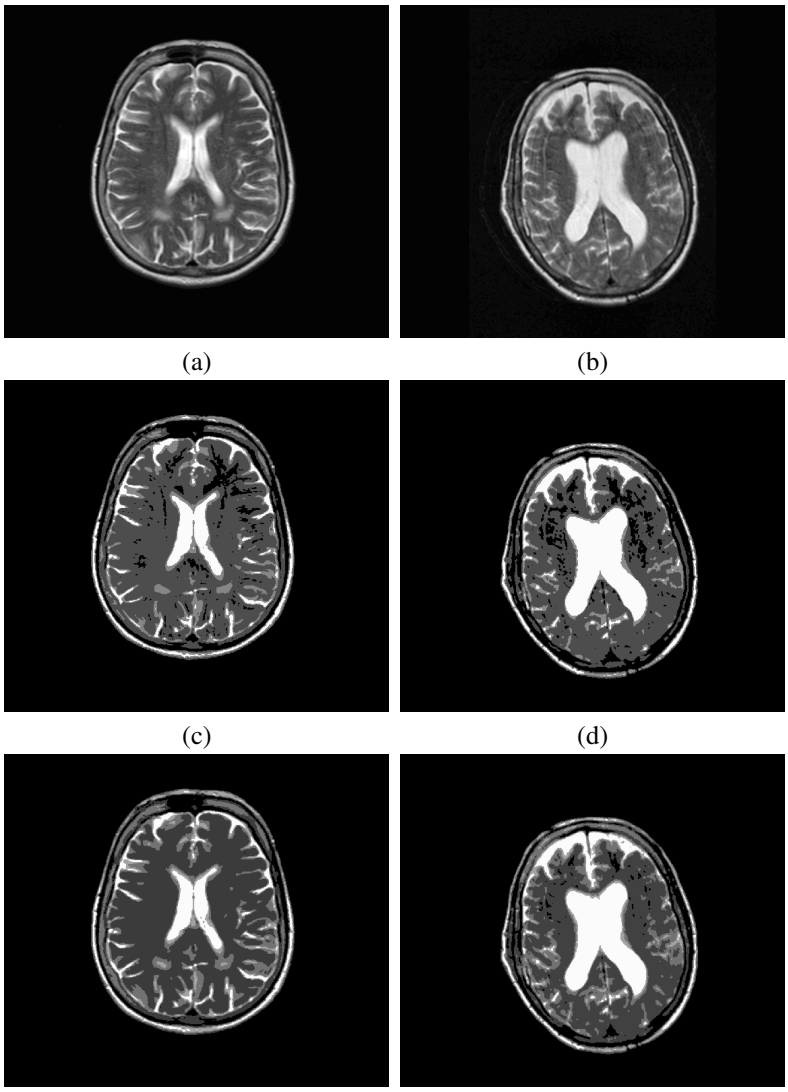


Figure 10.2. *Segmentation d'IRM sain et pathologique : (a) image originale dans le cas sain, (b) image originale dans le cas pathologique, (c) segmentation en quatre classes* $T = (68, 128, 192)$ *du cas sain, (d) segmentation en quatre classes* $T = (70, 129, 191)$ *du cas pathologique, (e) segmentation en cinq classes* $T = (57, 105, 155, 202)$ *du cas sain, (f) segmentation en cinq classes* $T = (58, 104, 154, 200)$ *du cas pathologique, où* T *est le vecteur de seuils.*

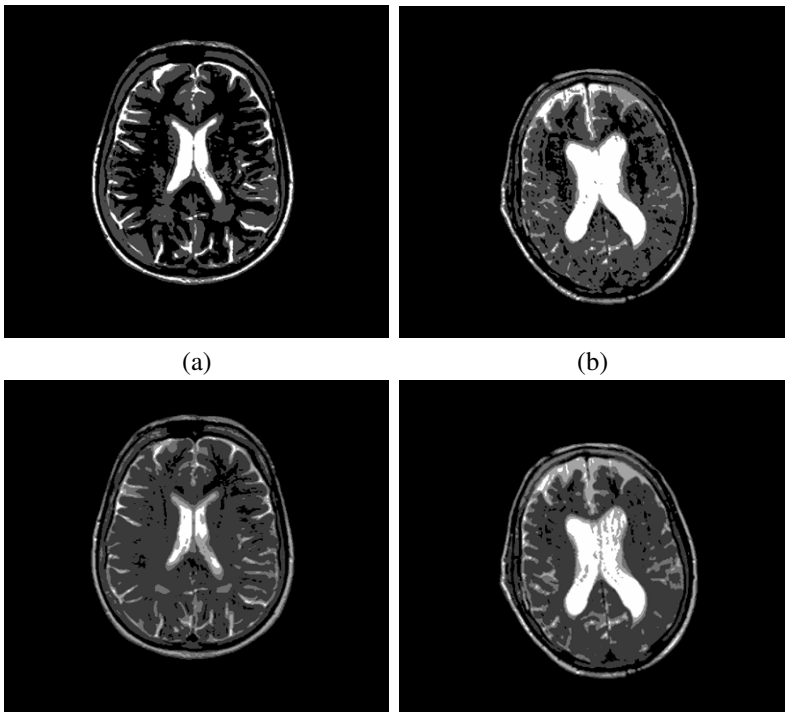


Figure 10.3. *Résultats de segmentation avec 2DSE : (a) segmentation en quatre classes* $T = (64, 128, 191)$ *du cas sain, (b) segmentation en quatre classes* $T = (65, 131, 192)$ *du cas pathologique, (c) segmentation en cinq classes* $T = (52, 102, 152, 203)$ *du cas sain, (d) segmentation en cinq classes* $T = (52, 103, 155, 205)$ *du cas pathologique, où* T *est le vecteur de seuils.*

Notre but est de détecter rapidement les différents espaces et matières blanches entourant l'espace des ventricules. Afin d'estimer la complexité de l'algorithme d'optimisation, nous avons calculé son temps d'exécution pour différentes classes. Le tableau 10.2 rassemble les résultats expérimentaux qui ont été obtenus sur l'image de la figure 10.2a. Le tableau montre l'efficacité de l'algorithme de colonie de fourmis, il confirme que notre méthode est rapide, en comparaison de celles décrites dans [COC 03, DRA 05, KAM 00, MUR 06, QIA 07, SON 06, WAR 00] : par exemple, dans [COC 03], les résultats sont obtenus après 120 *s*.

La figure 10.3 montre les résultats obtenus *via* l'application de la méthode 2DSE. Nous pouvons remarquer que les résultats procurés par notre méthode sont plus homogènes que ceux procurés par 2DSE. On le voit nettement, par exemple, en comparant les matières blanches détectées, dans la figure 10.2f et dans la figure 10.3d.

Nombre de classes	*Temps* (s)
3	3, 9
4	4, 6
5	6, 0

Tableau 10.2. *Résultats expérimentaux dans le cas sain de la figure 10.2a*

10.6.2. *Comparaison des performances*

Nous allons comparer les méthodes développées, du point de vue de la qualité de la segmentation obtenue. A cet effet, nous utilisons des images synthétiques, dont le résultat optimal de segmentation est connu *a priori* (image *vérité terrain* connue). Pour mesurer les performances de la segmentation, nous avons utilisé le critère de l'erreur de classification (ME). ME est défini en terme de corrélation des images avec l'observation humaine. Il correspond au rapport du nombre de pixels de l'arrière-plan faussement classés au premier plan, et *vice versa*. L'expression analytique de ME est donnée par :

$$ME = \left(1 - \left(\frac{|B_O \cap B_T| + |F_O \cap F_T|}{(|B_O| + |F_O|)}\right)\right) \cdot 100 \qquad (10.7)$$

où B_O et F_O sont, respectivement, l'arrière-plan et le premier plan de l'image originale et B_T et F_T sont, respectivement, l'arrière-plan et le premier plan de l'image test segmentée.

Les performances de MVAR-ACO sont comparées à celles de six autres méthodes : la méthode classique de N. Otsu, la méthode de Kapur, la méthode EM, la méthode VE (*valley emphasis*), la méthode basée sur l'entropie 2D de Tsallis (TE) et la méthode K-means basée sur la distance euclidienne. Les principes de base de toutes ces méthodes sont présentés dans [SEZ 04].

Pour notre expérience, l'image synthétique de la figure 10.4a est utilisée. Dans un premier temps, cette image est dégradée avec un bruit multiplicatif de plus en plus fort (figures 10.4b à d). Ensuite, un bruit blanc gaussien (BBG) additif est ajouté aux images déjà bruitées. On présente ces images sur la figure 10.4.

Les résultats de segmentation obtenus *via* les méthodes comparées sont présentés sur la figure 10.5. Pour chaque méthode, les résultats sont présentés sur une colonne de la figure. La quantification de ces résultats en fonction des valeurs du critère ME est résumée dans le tableau 10.3. En examinant le tableau 10.3, nous pouvons constater que la méthode de segmentation MVAR-ACO permet d'avoir de faibles valeurs du critère ME, c'est-à-dire une bonne qualité de segmentation, dans le cas d'un bruit multiplicatif (figures 10.4b à d). Lorsque le BBG est ajouté, la méthode n'est plus aussi performante. Cependant, ses performances dépassent celles des méthodes reposant sur le même principe (Otsu, Kapur, K-means, VE et TE).

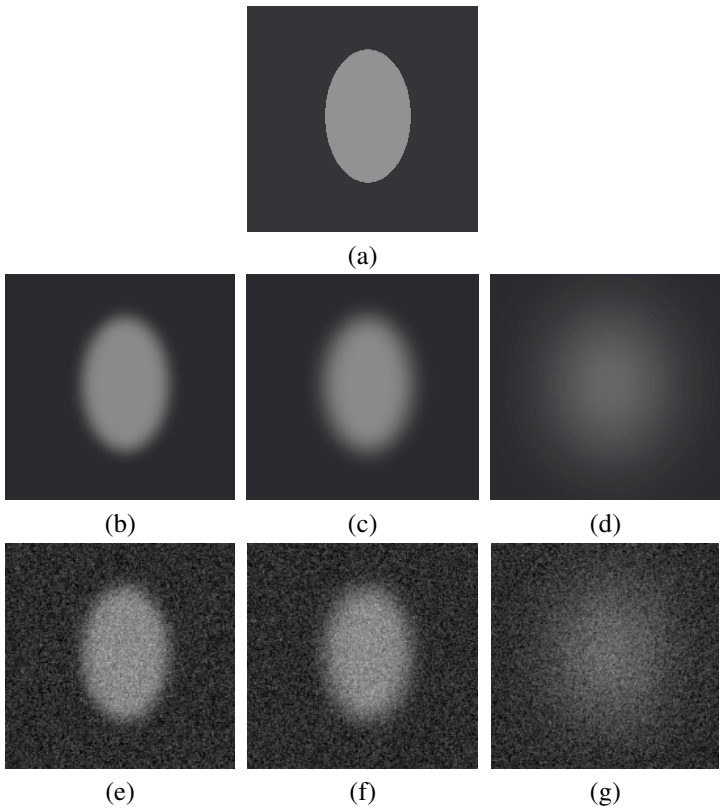


Figure 10.4. *Image originale (a) et images bruitées synthétiques : (b) image bruitée ($RSB = 17,77dB$), (c) image bruitée ($RSB = 15,44dB$), (d) image bruitée ($RSB = 5,84dB$), (e) image bruitée ($RSB = 15,91dB$), (f) image bruitée ($RSB = 13,91dB$), (g) image bruitée ($RSB = 5,44dB$), où RSB est le rapport signal sur bruit. En plus du bruit multiplicatif un BBG a été ajouté aux images (e), (f) et (g).*

Par rapport aux autres méthodes, l'algorithme MVAR-ACO est le plus performant, sauf dans le cas des images (e), (f) et (g), où la méthode EM le dépasse de peu. Ces résultats s'expliquent par le fait que, dans le cas des images (e), (f) et (g), la modélisation de l'histogramme avec deux gaussiennes était encore possible, malgré la densité du bruit, ce qui a permis d'avoir une bonne détection. Malheureusement, dans le dernier cas (image (g)), le bruit est si important que l'histogramme est fortement dégradé. En d'autres termes, il devient unimodal, ce qui rend la méthode EM peu performante.

D'un point de vue visuel, les résultats présentés sur la figure 10.5 montrent la supériorité de l'algorithme MVAR-ACO par rapport aux autres. En ce qui concerne les

six autres méthodes concurrentes, la méthode K-means permet d'avoir un résultat optimal, dans le cas des images faiblement bruitées. Malheureusement, l'augmentation du bruit a conduit à une baisse considérable de ses performances. Les résultats de la méthode de Kapur ne sont pas loin des résultats des autres méthodes, dans le cas des images (e) à (g), mais ne sont jamais les meilleurs. Quant à la méthode Otsu, ses performances sont médiocres dans le cas des images fortement bruitées. Les méthodes VE et TE permettent d'avoir des résultats comparables à ceux des autres méthodes, sans atteindre le résultat optimal.

Cette étude comparative nous a permis d'avoir une évaluation précise des performances du critère proposé ($MVar$).

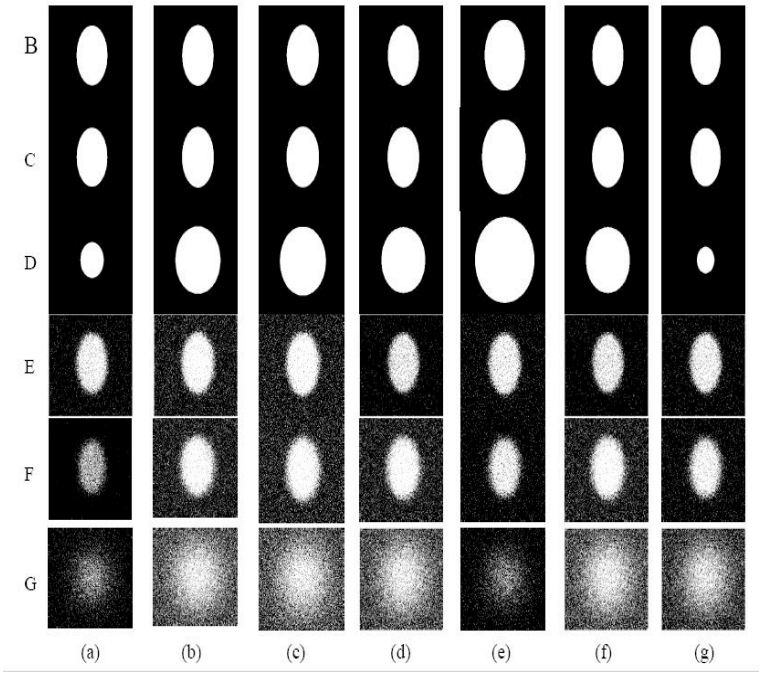


Figure 10.5. *Performances des différentes méthodes : (colonne a) images segmentées par MVAR-ACO, (colonne b) images segmentées par K-means, (colonne c) images segmentées par Otsu, (colonne d) images segmentées par Kapur, (colonne e) images segmentées par la méthode EM, (colonne f) images segmentées par la méthode VE, (colonne g) images segmentées par la méthode TE.*

Images test de la figure 10.4	Méthodes de segmentation						
	MVAR-ACO	K-means	Otsu	Kapur	EM	VE	TE
Image B	**0,17**	**0,17**	0,45	5,61	8,68	0,34	0,64
Image C	**0,60**	0,61	0,88	4,50	12,50	0,63	1,12
Image D	**4,53**	10,86	12,22	4,97	28,87	11,59	12,90
Image E	**5,75**	11,84	11,92	5,77	**5,74**	5,90	5,92
Image F	10,13	14,92	13,82	6,74	**6,60**	13,22	6,68
Image G	**15,64**	36,45	36,63	16,66	**16,63**	35,56	29,98

Tableau 10.3. *Evaluation des performances des différentes méthodes de segmentation. Les faibles valeurs de ME sont en gras*

10.7. Conclusion

Dans ce chapitre, nous avons présenté une méthode rapide pour trouver les seuils optimaux, basée sur un critère de segmentation $MVar$ et l'optimisation par colonie de fourmis. Les résultats expérimentaux montrent que la méthode proposée est plus efficace que la méthode classique 2DSE. En perspective à ce travail, nous travaillons sur la segmentation de séquences d'images IRM, en appliquant une optimisation dynamique par colonie de fourmis.

10.8. Bibliographie

[BLU 05] Blum C., « Ant colony optimization : Introduction and recent trends », *Physics of Life Reviews*, vol. 2, p. 353–373, 2005.

[BOR 98] Borsotti M., Campadelli P., Schettini R., « Quantitative evaluation of color image segmentation results », *Pattern Recognition Letters*, vol. 19, p. 741–747, 1998.

[CHE 98] Cheriet M., Said J., Suen C. Y., « A recursive thresholding technique for image segmentation », *IEEE Trans. on Image Processing*, vol. 7, n° 6, p. 918–921, 1998.

[COC 95] Cocquerez J. P., Philipp S., *Analyse d'images : filtrage et segmentation*, Masson, 1995.

[COC 03] Cocosco C. A., Evans A. C., « A fully automatic and robust brain MRI tissue classification », *Medical Image Analysis*, vol. 7, n° 4, p. 513–527, 2003.

[DOR 96] Dorigo M., Maniezzo V., Colorni A., « Ant System : optimization by a colony of cooperating agents », *IEEE Trans. on Man. Cyber. Part B*, vol. 26, n° 1, p. 29–41, 1996.

[DRA 05] Drapaca C. S., Cardenas S., « Segmentation of tissue boundary evolution from brain MR image sequences using multi-phase level sets », *Computer Vision and Image Understanding*, vol. 100, p. 312–329, 2005.

[HOU 06] Hou Z., Hu Q., Nowinski W. L., « On minimum variance thresholding », *Pattern Recognition Letters*, vol. 27, p. 1732–1743, 2006.

[KAM 00] Kamber M., Shinghal R., Collins D. L., Francis G. S., Evans A. C., « Model-based segmentation of multiple sclerosis lesions in magnetic resonance brain images », *IEEE Trans. on Med. Imaging*, vol. 14, n° 3, p. 442–453, 2000.

[KAP 85] Kapur J. N., Sahoo P. K., Wong A. C. K., « A new method for gray-level picture thresholding using the entropy of the histogram », *Computer Vision, Graphics and Image Processing*, vol. 29, p. 273–285, 1985.

[MUR 06] Murgasova M., Dyet L. nad Edwards D., Rutherford M., Hajnal J. V., Rueckert D., « Segmentation of brain MRI in young children », *9th Int. Conf. on Medical Image Comput. and Computer-Assisted Intervention (MICCAI)*, p. 687–694, Copenhague, Danemark, octobre 2006.

[NAK 07a] Nakib A., Conception de métaheuristiques d'optimisation pour la segmentation d'images. Application à des images biomédicales, Thèse, Université Paris 12 Val de Marne, décembre 2007.

[NAK 07b] Nakib A., Roman S., Oulhadj H., Siarry P., « Fast MRI segmentation based on two dimensional survival exponential entropy and particle swarm optimization. », *Proceedings of the 29th Annual International Conference on the IEEE EMBS*, p. 5563–5565, Lyon, 22-26 août 2007.

[OTS 79] Otsu N., « A Threshold Selection Method from Gray-Level Histograms », *IEEE Trans. on Syst., Man and Cyb.*, vol. 9, n° 1, p. 62–66, 1979.

[PAL 91] Pal N. K., Pal S. K., « Entropy : A new definition and its applications », *IEEE Trans. Syst. Man. Cybern.*, vol. 21, p. 1260–1270, 1991.

[QIA 07] Qiao Y., Hu Q., Qian G., Luo S., Nowinski W., « Thresholding based on variance and intensity contrast », *Pattern Recognition Letters*, vol. 40, p. 596–608, 2007.

[SEZ 04] Sezgin M., Sankur B., « Survey over image thresholding techniques and quantitative performance evaluation », *Electronic Imaging*, vol. 13, n° 1, p. 146–165, 2004.

[SON 06] Song Z., Tustison N., Avants B., Gee J. C., « Integrated Graph Cuts for Brain MRI Segmentation », *9th Int. Conf. on Medical Image Comput. and Computer-Assisted Intervention (MICCAI)*, p. 831–838, Copenhague, Danemark, octobre 2006.

[WAR 00] Warfield S. K., Kaus M., Jolesz F. A., Kikinis R., « Adaptive template moderate spatially varying statistical classification », *Medical Image Analysis*, vol. 4, n° 1, p. 43–55, 2000.

Chapitre 11

La coloration des sommets d'un graphe par colonies de fourmis

11.1. Introduction

En abordant ce chapitre, le lecteur n'a probablement plus besoin d'être convaincu que les fourmis artificielles sont capables de résoudre des problèmes d'optimisation combinatoire complexes. Vers le milieu des années 1990, la situation était cependant bien différente, car il n'existait alors que peu de problèmes résolus à l'aide de colonies de fourmis. C'est en 1997 que D. Costa et A. Hertz [COS 97] ont déclaré pour la première fois que les fourmis sont capables de colorer les sommets d'un graphe. Pourtant, en y regardant de plus près, l'article de D. Costa et A. Hertz contenait bel et bien des algorithmes de fourmis pour la coloration des sommets d'un graphe, mais ceux-ci, bien que plus performants que de simples algorithmes constructifs, produisaient des résultats qui ne pouvaient pas rivaliser avec ceux obtenus à l'aide d'autres métaheuristiques, telles que la recherche tabou [GLO 97] ou le recuit simulé [KIR 83]. Depuis 1997, plusieurs chercheurs se sont penchés sur l'utilisation de fourmis artificielles pour la coloration des sommets d'un graphe, et force est de constater que les algorithmes de fourmis sont désormais compétitifs avec les meilleurs algorithmes de coloration connus à ce jour. Le but de ce chapitre est de faire un historique de l'évolution de ces algorithmes de fourmis. Nous commencerons par une définition précise du problème à résoudre et présenterons ensuite trois approches de résolution, qui diffèrent par le rôle attribué à chaque fourmi. Quelques résultats numériques seront présentés, afin de permettre une comparaison des différentes approches, et nous terminerons ce chapitre par quelques commentaires et interrogations.

Chapitre rédigé par Alain HERTZ et Nicolas ZUFFEREY.

11.2. Le problème de la coloration des sommets d'un graphe (PCSG)

Soit $G = (V, E)$ un graphe, où V est un ensemble de sommets et E un ensemble d'arêtes reliant certaines paires de sommets. Etant donné un entier positif k, une *k-coloration* de G est une fonction $c : V \longrightarrow \{1, \ldots, k\}$ qui attribue une valeur $c(v)$, appelée *couleur* de v, à chaque sommet $v \in V$. Les sommets de même couleur définissent une *classe* de couleur. Si deux sommets v et w sont adjacents (c'est-à-dire reliés par une arête) et ont la même couleur, on dit qu'ils sont *en conflit* et que l'arête qui relie v à w est une arête *conflictuelle*. Une k-coloration sans conflit est dite *légale*. Le plus petit entier k tel qu'il existe une k-coloration légale de G est appelé *nombre chromatique* de G et est noté $\chi(G)$. Le problème de la détermination du nombre chromatique d'un graphe, que nous noterons PCSG (pour problème de la coloration des sommets d'un graphe), est NP-dur [GAR 79]. Pour un entier positif k fixé, le problème de la k-coloration des sommets d'un graphe (k-PCSG) consiste à déterminer s'il existe une k-coloration légale de G. Une heuristique pour le k-PCSG peut permettre d'améliorer n'importe quelle borne supérieure B sur $\chi(G)$. En effet, étant donnée une borne $B \geq \chi(G)$ (par exemple $B = |V|$), on peut résoudre une suite de k-PCSGs avec des valeurs décroissantes de k (en partant de $k = B - 1$) jusqu'à ce qu'aucune k-coloration légale ne soit obtenue.

Pour illustrer toute cette terminologie, nous avons représenté à gauche de la figure 11.1 une 3-coloration non légale d'un graphe. Les lettres correspondent aux noms des sommets, alors que les nombres représentent les couleurs. Les arêtes reliant b avec e et c avec d sont conflictuelles et sont représentées avec des traits plus épais. Les sommets b, c, d et e sont en conflit. Le nombre chromatique de ce graphe est trois puisque, tel qu'illustré à droite de la figure 11.1, il existe une 3-coloration légale du même graphe, alors que le triangle constitué des sommets b, d et e (ou c, d et e, ou d, e et f) prouve qu'il est impossible d'utiliser moins de couleurs dans une coloration légale.

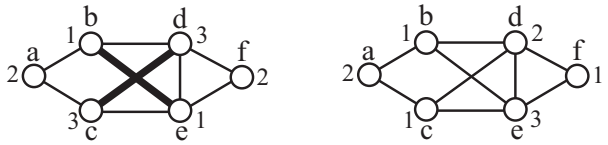


Figure 11.1. *Illustration des définitions de base*

Le PCSG a de nombreuses applications pratiques, principalement dans les domaines de la confection d'horaires, de l'ordonnancement et de l'affectation de fréquences dans les réseaux radio-mobiles [GAM 92, LEI 79, STE 85]. Il n'est par conséquent pas étonnant que de nombreux chercheurs se soient penchés sur ce problème. De nos jours, les meilleurs algorithmes exacts pour le PCSG ne sont pas en mesure de traiter des graphes de plus de cent sommets [HER 02, MEH 96]. Pour les plus grandes

instances, l'usage d'heuristiques est donc inévitable pour obtenir de bonnes bornes supérieures sur le nombre chromatique.

Un très grand nombre d'heuristiques ont été proposées pour résoudre le PCSG ou le *k*-PCSG. Les algorithmes *constructifs* parcourent les sommets séquentiellement en attribuant à chacun la plus petite couleur possible, c'est-à-dire le plus petit entier positif qui n'a pas encore été attribué à un sommet adjacent. Ce type d'heuristique est simple à implémenter et permet d'obtenir rapidement une borne supérieure sur le nombre chromatique. La qualité de la solution obtenue dépend cependant fortement de l'ordre dans lequel les sommets sont colorés. Les algorithmes constructifs les plus utilisés en pratique sont DSATUR [BRE 79] et RLF [LEI 79] qui utilisent un ordre des sommets construit de manière dynamique. Les métaheuristiques permettent d'obtenir de meilleures bornes sur le nombre chromatique, mais en des temps de calcul plus importants. Les types de métaheuristiques utilisés pour résoudre le PCSG sont variés puisque, outre les algorithmes de fourmis que nous décrirons par la suite, on y trouve des techniques de recherche locale, telles que le recuit simulé [CHA 87, JOH 91], la recherche tabou [BLO 08, HER 87], la recherche à voisinages variables [AVA 03], la recherche à espaces de solutions variables [HER 08], ainsi que des méthodes hybrides, qui combinent la recherche locale avec des techniques évolutives qui gèrent des populations de solutions [FLE 96, GAL 99, GAL 08, MAL 05, MOR 96]. Pour plus de détails sur les algorithmes constructifs ou pour une revue récente sur les métaheuristiques proposées pour le PCSG, le lecteur peut se référer à [HER 00] ou [GAL 06].

11.3. Trois approches par colonies de fourmis pour le PCSG

Les premiers algorithmes de fourmis pour le PCSG ont vu le jour en 1997 et de nouvelles publications dans le domaine paraissent depuis lors régulièrement. On peut regrouper ces différentes publications en trois classes, selon le rôle que les fourmis jouent dans l'algorithme. Dans les premiers algorithmes proposés, chaque fourmi est un algorithme constructif qui laisse une trace sur chaque paire de sommets non adjacents, pour indiquer si ces sommets ont reçu la même couleur. La deuxième catégorie comprend les algorithmes dans lesquels des fourmis se promènent sur le graphe et tentent collectivement de modifier la couleur des sommets qu'elles visitent, l'objectif étant de diminuer le nombre d'arêtes conflictuelles dans une *k*-coloration non légale. Finalement, dans la troisième catégorie, les fourmis sont devenues des algorithmes de recherche locale qui laissent des traces sur l'exploration qu'elles ont faites de l'espace de recherche. Alors que les premiers algorithmes de fourmis n'étaient pas compétitifs avec d'autres métaheuristiques pour la coloration des sommets d'un graphe, les algorithmes les plus récents (ceux de la troisième catégorie) rivalisent positivement avec les meilleurs algorithmes connus à ce jour. Dans les trois sous-sections qui suivent, nous décrivons ces trois approches de manière plus détaillée.

11.3.1. *Chaque fourmi est un algorithme constructif*

La première approche par colonie de fourmis pour le PCSG a été proposée par D. Costa et A. Hertz dans [COS 97]. Comme il s'agit d'extensions des algorithmes DSATUR [BRE 79] et RLF [LEI 79], nous donnons tout d'abord une description sommaire de ces algorithmes.

Etant donnée une coloration partielle sans arête conflictuelle d'un graphe G (c'est-à-dire que seuls quelques sommets sont colorés), nous noterons U l'ensemble des sommets qu'il reste à colorer. Le *degré de saturation* d'un sommet $v \in U$ est défini comme le nombre de couleurs différentes apparaissant sur des sommets adjacents à v. Par exemple, dans le graphe de la figure 11.2 ci-dessous, on a trois sommets déjà colorés, ce qui donne $U = \{b, d, f, g\}$. Le degré de saturation de b et d est 2, alors qu'il vaut 1 pour les sommets f et g.

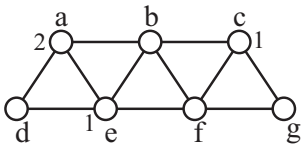


Figure 11.2. *Exemple de coloration partielle*

L'algorithme DSATUR colore les sommets séquentiellement en choisissant, à chaque étape, le sommet de U ayant le plus grand degré de saturation. Si plusieurs sommets maximisent cette valeur, l'algorithme en choisit un ayant un nombre maximum de sommets adjacents non colorés. La couleur attribuée au sommet choisi est la plus petite possible, c'est-à-dire le plus petit entier positif n'apparaissant pas sur un sommet adjacent. Dans l'exemple de la figure 11.2, les sommets b et d ont le plus grand degré de saturation, et l'algorithme choisira b car il est adjacent à un sommet non coloré (le sommet f), ce qui n'est pas le cas pour le sommet d. Comme les couleurs 1 et 2 apparaissent sur des sommets adjacents à b, l'algorithme lui attribuera la couleur 3.

L'algorithme RLF construit les classes de couleur les unes après les autres. Pour chaque classe, le premier sommet à en faire partie est choisi au hasard parmi ceux ayant un nombre maximum de sommets adjacents non colorés. La classe est alors complétée de la manière suivante. Soit $W \subseteq U$ l'ensemble des sommets non colorés qui ne peuvent plus faire partie de la classe en construction (car ils sont adjacents à au moins un sommet de cette classe). Le prochain sommet introduit dans la classe est un sommet faisant partie de U mais pas de W, et ayant un nombre maximum de sommets adjacents dans W. Lorsque U=W, on passe à la classe de couleur suivante. Dans l'exemple de la figure 11.2, supposons que l'on soit en train de construire la classe des sommets de couleur 2. L'ensemble U des sommets non colorés est $\{b, d, f, g\}$, alors que $W = \{b, d\}$. Comme le sommet f est adjacent au sommet $b \in W$, alors que le

sommet g n'est adjacent à aucun sommet de W, l'algorithme RLF choisira f comme prochain sommet de couleur 2. On aura alors $U = W = \{b, d, g\}$ et l'algorithme passera donc à la troisième classe de couleur.

Tel que déjà mentionné, un algorithme constructif complète pas à pas une solution partielle jusqu'à ce que tous les sommets soient colorés. A chaque étape, un sommet non coloré est choisi et une couleur lui est attribuée. Dans les algorithmes DSATUR et RLF, il n'y a aucun choix à faire quant à la couleur à attribuer au sommet choisi. En effet, une fois que DSATUR a choisi le prochain sommet à colorer, la couleur donnée à ce sommet est le plus petit entier n'apparaissant pas sur un sommet qui lui est adjacent. Pour l'algorithme RLF, la situation est un peu différente puisque la couleur est choisie d'office (en l'occurrence la couleur de la classe qui est complétée), et l'algorithme doit donc simplement effectuer un choix quant au sommet à qui l'on attribuera cette couleur. Dans les algorithmes de fourmis F_{DSATUR} et F_{RLF}, basés respectivement sur DSATUR et RLF, et proposés dans [COS 97], les fourmis laissent des traces qui influencent le choix du prochain sommet à colorer, mais pas la couleur à attribuer à ce sommet. Plus précisément, à chaque étape de l'algorithme constructif, chaque fourmi doit décider quel sera le prochain sommet à colorer, et ce choix dépend de deux facteurs :

– *l'attrait* : chaque fourmi a tendance à effectuer le choix le plus attrayant. Ainsi, dans F_{DSATUR}, cet attrait $A(v)$ pour un sommet v non coloré est défini comme le degré de saturation de v, alors que dans F_{RLF}, l'attrait $A(v)$ d'un sommet faisant partie de U, mais pas de W, est égal au nombre de sommets adjacents à v dans W ;

– *la trace* : les fourmis qui ont fait de bons choix dans le passé ont laissé une trace pour tenter d'influencer les choix futurs des fourmis. Dans notre cas, étant donnés deux sommets non adjacents v et w, soit $t(v, w)$ une trace laissée par les fourmis indiquant s'il est judicieux de donner la même couleur à ces deux sommets. Tel que mentionné ci-dessus, le choix d'un sommet v conduit automatiquement à l'attribution d'une couleur c à v, cette couleur étant soit la plus petite qui ne crée pas de conflit (pour F_{DSATUR}), soit la couleur de la classe en cours de complétion (pour F_{RLF}). En notant V_c l'ensemble des sommets de couleur c dans la solution partielle considérée, la trace $T(v)$ associée au choix du sommet v est alors définie comme $T(v) = \sum_{w \in V_c} t(v, w)$.

En notant C l'ensemble des sommets pouvant être choisis à une étape donnée (c'est-à-dire $C = U$ dans F_{DSATUR} et $C = U - W$ dans F_{RLF}), le choix du prochain sommet à colorer est effectué de manière standard, c'est-à-dire en choisissant aléatoirement dans C, avec la probabilité $P(v)$ de choisir v définie comme suit :

$$P(v) = \frac{A(v)^{\alpha} T(v)^{\beta}}{\sum_{w \in C} A(w)^{\alpha} T(w)^{\beta}}$$

où α et β sont deux paramètres qui donnent plus ou moins d'importance à chacun des deux facteurs influençant les choix des fourmis.

Au début de l'algorithme, toutes les valeurs $t(v, w)$ sont initialisées à zéro. Puis, à chaque fois qu'une fourmi a coloré tous les sommets du graphe, elle considère le nombre k de couleurs dans la solution s ainsi obtenue et augmente de $\frac{1}{k}$ les valeurs $t(v, w)$ associées aux sommets v et w de même couleur dans s. De plus, comme dans tout algorithme de fourmis, les valeurs $t(v, w)$ diminuent régulièrement grâce à un *facteur d'évaporation* ρ.

Plus de détails sur ces deux algorithmes de fourmis sont donnés dans [COS 97]. A. Vesel et J. Žerovnik [VES 00] ont comparé F_{DSATUR} et F_{RLF} avec des algorithmes de type Petford-Welsh [PET 89]. Force est de constater que les résultats obtenus par ces algorithmes de fourmis ne sont pas de très grande qualité. S. Ahn *et al.* [AHN 03] ont amélioré quelque peu F_{RLF} en basant cet algorithme de fourmis sur une version améliorée de RLF, appelée XRLF, et décrite dans [JOH 91]. De plus, M. Bessedik *et al.* [BES 05] ont montré qu'il est possible d'améliorer ces algorithmes de fourmis en les combinant avec des techniques de recherche locale, sans pour autant les rendre compétitifs avec les meilleurs algorithmes de coloration.

11.3.2. *Les fourmis se promènent sur le graphe*

La deuxième approche par colonies de fourmis pour colorer les sommets d'un graphe consiste à tenter de résoudre le k-PCSG, pour un entier k donné, en plaçant des fourmis sur le graphe à colorer, et en les laissant se promener de sommet en sommet, chaque fourmi ayant le pouvoir de modifier la couleur des sommets qu'elle visite. Plus précisément, notons S l'ensemble de toutes les k-colorations (pas nécessairement légales) du graphe considéré. Le but de ces algorithmes de fourmis est d'agir collectivement, afin de déterminer un élément $s \in S$ qui ne comporte aucune arête conflictuelle. La déplacement des fourmis sur le graphe permet de modifier l'emplacement des conflits, voire de les éliminer. Les deux algorithmes des sous-sections suivantes sont deux exemples d'utilisation d'une telle approche.

11.3.2.1. *Les fourmis changent la couleur des sommets vers lesquels elles se rendent*

Dans l'algorithme proposé par F. Comellas et J. Ozón, que nous noterons F_{CO}, un ensemble de fourmis est dispersé sur les sommets du graphe à colorer. Etant donnée une k-coloration $s \in S$, notons $n_v(s)$ le nombre d'arêtes conflictuelles ayant v comme extrémité. En d'autres termes, $n_v(s)$ représente le nombre de sommets adjacents à v et ayant la même couleur que v. A chaque itération de l'algorithme, chaque fourmi se déplace avec la probabilité p du sommet v sur lequel elle se trouve vers un sommet adjacent w de valeur $n_w(s)$ maximale, et avec la probabilité (1-p) vers un sommet w adjacent à v choisi au hasard. Une fois parvenue sur ce nouveau sommet w, la fourmi change la couleur de w en lui attribuant, avec la probabilité q, une nouvelle couleur qui crée le moins de conflits possibles, et avec la probabilité (1-q), une nouvelle couleur choisie au hasard. Lorsque toutes les fourmis ont réalisé leur modification de couleur

d'un sommet, les valeurs $n_v(s)$ sont mises à jour pour tous les sommets v du graphe et une nouvelle itération débute. Les valeurs de p et q ainsi que le nombre de fourmis qui se promènent sur le graphe sont des paramètres de l'algorithme. Notons qu'aucun système explicite de trace n'est utilisé dans cette méthode.

11.3.2.2. *Les fourmis sont colorées et votent pour leur couleur là où elles se trouvent*

A. Hertz et N. Zufferey [HER 06, ZUF 02] proposent d'associer une couleur (c'est-à-dire un entier dans $\{1, \ldots, k\}$) à chaque fourmi. A chaque itération de leur algorithme que nous noterons F_{HZ}, il y a exactement k fourmis sur chaque sommet. Initialement, chaque sommet possède exactement une fourmi de chaque couleur, puis celles-ci se déplacent pour tenter de modifier la couleur des sommets en conflit.

Un ingrédient important de cet algorithme de fourmis est la procédure, que nous appellerons *Color*, dont le but est de compléter une coloration partielle. Plus précisément, supposons qu'un sous-ensemble de sommets de G est déjà coloré et soit U l'ensemble des sommets qu'il reste à colorer. La coloration des sommets de U se fait séquentiellement, selon des principes similaires à ceux de l'algorithme DSATUR (voir section 11.3.1). A chaque étape, le prochain sommet à colorer est choisi parmi ceux ayant le plus grand degré de saturation. En cas d'égalité, l'algorithme en choisit un ayant un nombre maximum de sommets adjacents non colorés. Ce qui change par rapport à DSATUR, c'est le choix de la couleur à attribuer à ce sommet choisi. Tout d'abord, cette couleur doit être portée par au moins une fourmi présente sur le sommet à colorer. Ensuite, parmi toutes ces couleurs possibles, le choix se porte sur l'une d'entre elles qui crée un nombre minimum de nouvelles arêtes conflictuelles. Si plusieurs couleurs sont encore possibles à ce stade, on en choisira une représentée par le plus grand nombre de fourmis sur ce sommet.

Cet algorithme est illustré dans la figure 11.3. Les nombres à l'intérieur des grands cercles indiquent les couleurs des fourmis présentes sur les sommets qu'il reste à colorer. Une fois un sommet coloré, celui-ci est représenté par un petit cercle et sa couleur par un entier à ses côtés. Dans la coloration partielle représentée en 3(a), il reste les trois sommets d, e et f à colorer. Les sommets les plus saturés sont d et e et l'algorithme choisira par exemple d. Etant donné que la couleur 2 n'est pas présente sur d, elle n'est pas choisie, bien qu'elle ne créerait aucun conflit. Parmi les deux couleurs possibles, chacune crée un conflit, et on choisit donc la couleur la plus représentée, c'est-à-dire la couleur 1. La nouvelle coloration partielle est représentée en 3(b) avec l'arête conflictuelle reliant les sommets c et d en gras. Le prochain sommet le plus saturé est le sommet e puisqu'il est adjacent à deux couleurs différentes, alors que le sommet f n'est adjacent qu'à une couleur. A nouveau, on ne peut pas donner la couleur 2 à e car aucune fourmi de couleur 2 n'est présente sur ce sommet. Etant donné qu'en attribuant la couleur 3 à e on ne crée qu'un conflit, alors qu'on en créerait deux avec la couleur 1, l'algorithme attribue la couleur 3 à e pour obtenir la coloration partielle représentée en 3(c). Finalement, le dernier sommet f est coloré avec la couleur

2, qui ne crée aucun nouveau conflit et on obtient la 3-coloration complète du graphe représentée en 3(d) avec deux arêtes conflictuelles.

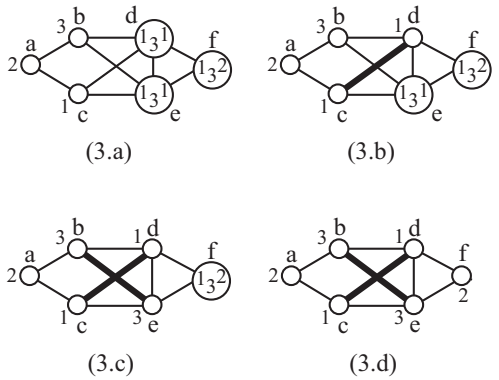


Figure 11.3. *Complétion d'une coloration partielle par l'algorithme Color*

A chaque itération de l'algorithme F_{HZ}, l'objectif visé est d'éliminer au moins une arête conflictuelle, quitte à en ajouter d'autres ailleurs. Pour ce faire, les sommets s'échangent des fourmis, en tentant de faire disparaître les fourmis qui sont responsables de la création de conflits. Plus précisément, notons $c(v)$ la couleur du sommet v dans la k-coloration courante. A chaque itération de l'algorithme, un sommet v en conflit est choisi et toutes les fourmis de couleur $c(v)$ sont éliminées de v, pour s'assurer que l'algorithme *Color* ne redonnera pas cette couleur à v. Ces fourmis de couleur $c(v)$ sont éliminées de v en effectuant une série d'échanges, que nous noterons $m = (v, w, i)$, et qui consistent à permuter une fourmi de couleur $c(v)$ sur v avec une fourmi de couleur $i \neq c(v)$ sur $w \neq v$. Il nous faut encore décrire comment ce sommet v en conflit est choisi, et comment les échanges $m = (v, w, i)$ sont déterminés. Ces choix, comme dans tout algorithme de fourmis, sont basés sur les deux facteurs que sont l'attrait et la trace. Nous décrivons tout d'abord l'attrait $A(m)$ d'un échange m.

Les quatre situations suivantes rendent l'échange $m = (v, w, i)$ attrayant : v contient beaucoup de fourmis de couleur i ; w contient beaucoup de fourmis de couleur $c(v)$; il y a beaucoup de fourmis de couleur $c(v)$ sur des sommets adjacents à v ; il y a beaucoup de fourmis de couleur i sur des sommets adjacents à w. A l'opposé, il semble peu attrayant de réaliser un tel échange si w contient beaucoup de fourmis de couleur i, s'il y a beaucoup de fourmis de couleur i sur des sommets adjacents à v, ou s'il y a beaucoup de fourmis de couleur $c(v)$ sur des sommets adjacents à w. Notons que le nombre de fourmis de couleur $c(v)$ sur v n'influence pas l'attrait de m car, si un tel échange est effectué, toutes les fourmis de couleur $c(v)$ seront éliminées de v, quel que soit leur nombre. En résumé, soit $S_j(u, w)$ le nombre de fourmis de couleur j sur les

sommets adjacents à u autres que w, et soit $N_j(u)$ le nombre de fourmis de couleur j sur u. L'attrait $A(m)$ défini dans [HER 06] est :

$$A(m) = N_i(v)^2 + N_{c(v)}(w)^2 - N_i(w)^2 + S_{c(v)}(v, w) + S_i(w, v) - S_{c(v)}(w, v) - S_i(v, w)$$

Les carrés qui apparaissent dans cette formule permettent de donner plus d'importance au fait qu'il faut tâcher de regrouper les fourmis de même couleur.

Les traces $t(w, i)$, définies dans [HER 06], existent pour toute paire (w, i) où w est un sommet et i une couleur. Lorsque l'on effectue un échange $m = (v, w, i)$, les traces $t(v, i))$ et $t(w, c(v))$ ont tendance à se renforcer, alors que $t(v, c(v))$ et $t(w, i)$ sont atténuées. Ces traces s'évaporent graduellement, cette évaporation étant plus importante pour la trace $t(v, c(v))$ lorsque toutes les fourmis de couleur $c(v)$ ont été retirées de v. La trace $T(m)$ d'un échange $m = (v, w, i)$ est alors définie comme $T(m) = t(v, i) + t(w, c(v)) - t(v, c(v)) - t(w, i)$. Plus de détails sont donnés dans [HER 06].

On peut maintenant décrire une itération de l'algorithme de fourmis F_{HZ}. A chaque itération, les attraits $A(m)$ et traces $T(m)$ des échanges m possibles sont normalisés dans l'intervalle [0,1] et une valeur $P(m) = \alpha A(m) + \beta T(m)$ est calculée pour tout échange m, où α et β sont deux paramètres qui permettent de donner plus ou moins d'importance à l'attrait ou la trace d'un échange. L'algorithme sélectionne ensuite un échange m de valeur $P(m)$ maximale. Supposons que cet échange m choisi consiste à remplacer une fourmi de couleur $c(v)$ sur v par une fourmi de couleur différente. Les autres fourmis de couleur $c(v)$ sont ensuite retirées séquentiellement du sommet v, en choisissant à chaque fois un échange de valeur $P(m)$ maximale. Puis, tous les sommets dont l'ensemble des fourmis a été modifié ainsi que tous ceux qui étaient en conflit au début de l'itération sont décolorés, alors que le reste du graphe conserve sa coloration. L'algorithme *Color* complète finalement cette coloration partielle et les traces sont mises à jour.

Ce processus est illustré dans la figure 11.4. En 4(a), nous avons représenté une coloration avec deux arêtes conflictuelles. Comme précédemment, les nombres dans les grands cercles correspondent aux couleurs des fourmis présentes sur les sommets alors que les entiers en dehors des cercles représentent les couleurs des sommets. Supposons que l'échange m choisi en premier consiste à ôter une fourmi de couleur $c(d) = 1$ du sommet d pour la remplacer par une fourmi de couleur 3 prise sur b. La deuxième fourmi de couleur 1 sur d est ensuite échangée, par exemple avec une fourmi de couleur 2 sur f. Il n'y a alors plus aucune fourmi de couleur 1 sur d. Comme les sommets b, d et f ont modifié leur ensemble de fourmis, ils sont décolorés. De plus, comme les sommets c et e étaient impliqués dans des conflits, ils sont décolorés également. L'algorithme *Color* démarre donc sa coloration partielle avec uniquement la couleur 2 sur le sommet a, tel que représenté en 4(b). La séquence de coloration produite par *Color* pourrait alors être l'attribution de la couleur 1 sur b, puis 3 sur e, 2 sur d, 1 sur c et finalement 1 sur f, ce qui donne la coloration représentée en 4(c), qui ne comporte plus aucune arête conflictuelle.

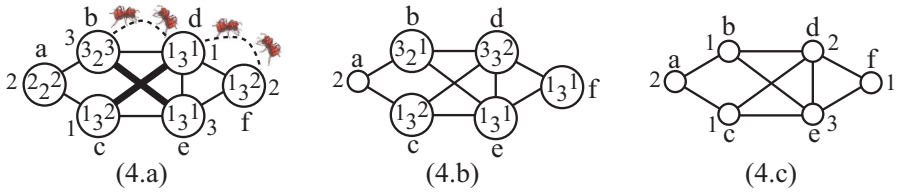


Figure 11.4. *Une itération de l'algorithme décrit dans [HER 06]*

Quelques ingrédients supplémentaires sont utilisés dans F_{HZ}. Par exemple, lorsqu'une itération a fait disparaître toutes les fourmis de couleur $c(v)$ sur un sommet v, l'algorithme interdit de remettre des fourmis de couleur $c(v)$ sur v pour un certain nombre d'itérations, à l'instar d'une recherche tabou. Pour plus de détails, le lecteur est invité à consulter [HER 06] ou [ZUF 02].

11.3.3. *Chaque fourmi est une procédure de recherche locale*

Dans la troisième et dernière approche présentée dans ce chapitre, les fourmis tentent à nouveau de résoudre le k-PCSG avec un entier k fixé. Cependant, le rôle joué par chaque fourmi est cette fois-ci beaucoup plus important, puisqu'il ne se limite pas à tenter de changer la couleur des sommets grâce à des déplacements sur le graphe. Chaque fourmi agit comme une procédure de recherche locale et les traces laissées lors de chaque tentative de diminuer le nombre d'arêtes conflictuelles influencent les recherches futures. Nous décrivons ci-dessous deux algorithmes récents qui entrent dans ce cadre.

11.3.3.1. *Chaque fourmi est un algorithme de type Petford-Welsh [PET 89]*

J. Shawe-Taylor et J. Žerovnik [SHA 02] proposent de considérer chaque fourmi comme une procédure itérative qui tente d'éliminer les arêtes conflictuelles, en utilisant une généralisation de l'algorithme de Petford-Welsh [PET 89]. Etant donnée une k-coloration avec des arêtes conflictuelles, l'algorithme de Petford-Welsh choisit, à chaque itération, un sommet en conflit et lui attribue une nouvelle couleur. L'idée principale décrite dans [SHA 02] est de travailler parallèlement sur le graphe G à colorer, ainsi que sur un graphe G' qui est initialement égal à G et auquel on ajoute périodiquement des arêtes. Ces arêtes additionnelles sont en quelque sorte une trace laissée par les fourmis lors d'utilisations précédentes de l'algorithme de Petford-Welsh. Lorsque beaucoup de fourmis ont donné deux couleurs différentes à deux sommets v et w non adjacents et ont obtenu de bonnes colorations, la trace laissée par les fourmis pour mémoriser ce fait est l'ajout d'une arête dans G'.

Plus précisément, chaque fourmi tente d'améliorer sa k-coloration en utilisant l'algorithme de Petford-Welsh pour un nombre fixé M d'itérations. Tel qu'indiqué ci-dessus, chaque fourmi choisit à chaque itération un sommet v en conflit. Ce choix est

effectué aléatoirement parmi les sommets à l'extrémité d'une arête conflictuelle de G. Une nouvelle couleur est alors attribuée au sommet v. Pour chaque couleur i différente de la couleur actuelle $c(v)$ de v, notons n_i le nombre de sommets de couleur i adjacents à v dans G'. J. Shawe-Taylor et J. Žerovnik proposent de choisir i comme nouvelle couleur avec une probabilité proportionnelle à $e^{\frac{-n_i}{T}}$, où T est un paramètre de l'algorithme, au même titre que le nombre M d'itérations.

A la fin de ces M itérations, notons X la population de k-colorations obtenues, et $f(x)$ le nombre d'arêtes conflictuelles dans G pour la k-coloration $x \in X$. Pour chaque paire de sommets v et w non adjacents dans G', notons finalement X_{vw} le sous-ensemble des k-colorations de X dans lesquelles v et w ont des couleurs différentes. *L'évidence* $E(v, w)$ de cette non-arête de G' est définie comme suit :

$$E(v, w) = \sum_{x \in X_{vw}} e^{\frac{-f(x)}{T'}}$$

où T' est un troisième paramètre de cet algorithme.

Après M itérations, toutes les non-arêtes de G' d'évidence maximale sont ajoutées à G', ce qui correspond à la mise à jour des traces, et un nouveau cycle de M itérations est initialisé. Aucune évaporation des traces n'est prévue dans cet algorithme, ce qui signifie qu'aucune arête n'est retirée de G'. J. Shawe-Taylor et J. Žerovnik justifient ce fait en faisant une analogie avec le comportement humain lors de la résolution de problèmes complexes. Il est en effet assez courant d'ajouter graduellement des contraintes qui ne semblent pas exclure la solution optimale recherchée, ces ajouts ayant pour but de modifier l'espace de recherche, afin de mieux guider la quête de l'optimum. Lorsque de nombreuses fourmis obtiennent des k-colorations avec très peu d'arêtes conflictuelles et que deux sommets v et w n'ont jamais la même couleur dans toutes ces bonnes colorations, il semble raisonnable de faire l'hypothèse que v et w n'ont pas la même couleur, dans une solution optimale sans conflit. Pour inciter l'algorithme à donner des couleurs différentes à v et w, une arête est donc ajoutée à G'.

11.3.3.2. *Chaque fourmi modifie une k-coloration partielle légale*

L'algorithme de fourmis proposé par M. Plumettaz *et al.* [PLU 09], noté F_{PSZ}, tente également de résoudre le k-PCSG, mais avec une approche tout à fait différente. Une k-coloration légale doit satisfaire deux contraintes : chaque sommet doit avoir une couleur choisie dans $\{1, \ldots, k\}$ et les extrémités de chaque arête doivent avoir des couleurs différentes. Alors que l'algorithme précédent respectait la première contrainte, mais autorisait les violations de la deuxième contrainte, tout en les pénalisant, l'algorithme F_{PSZ} interdit toute arête conflictuelle, mais n'impose pas que tous les sommets soient colorés. Chaque sommet non coloré induit une pénalité et l'objectif est donc de minimiser le nombre de sommets non colorés. Cette stratégie a été initialement proposée par C. Morgenstern [MOR 96] dans le cadre d'un recuit simulé, puis utilisée par

I. Blöchliger et N. Zufferey [BLO 08] dans une recherche tabou. Une k-coloration sans arête conflictuelle dans laquelle quelques sommets ne sont pas colorés est appelée une *k-coloration partielle légale*.

Les fourmis de l'algorithme F_{PSZ} résolvent le k-PCSG en généralisant un algorithme de recherche tabou efficace proposé par I. Blöchliger et N. Zufferey [BLO 08] que nous noterons BZ. Plus précisément, soit V_i $(1 \leq i \leq k)$ l'ensemble des sommets de couleur i et V_{k+1} l'ensemble des sommets non colorés. A chaque itération, chaque fourmi tente d'améliorer sa k-coloration partielle légale en colorant un sommet v de V_{k+1} avec l'une des k couleurs à disposition, disons i. Si V_i contient des sommets adjacents à v, ceux-ci sont décolorés, c'est-à-dire qu'ils sont déplacés dans V_{k+1}. A titre d'illustration, le graphe de gauche de la figure 11.5 représente une 3-coloration partielle légale avec $V_1 = \{c\}$, $V_2 = \{a, f\}$, $V_3 = \{b\}$ et $V_4 = \{d, e\}$. Si une fourmi décide d'attribuer la couleur 1 au sommet d, le sommet c est décoloré, tel que représenté à droite de la figure 11.5. On observe que l'itération suivante pourra attribuer la couleur 3 à c en ne décolorant aucun sommet.

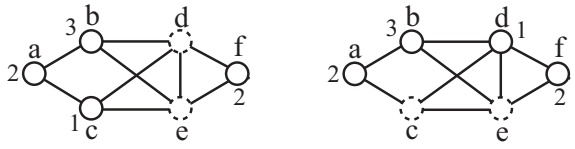


Figure 11.5. *Une itération de l'algorithme décrit dans [PLU 09]*

L'attrait $A(v, i)$ de l'attribution de la couleur i à un sommet $v \in V_{k+1}$ est défini comme l'inverse du nombre de sommets dans V_i qui devront être décolorés, c'est-à-dire qui sont adjacents à v. Une trace $t(v, w)$ est définie pour toute paire (v, w) de sommets. A chaque itération, chaque fourmi ajoute une quantité $\mid V_j \mid^2$ aux traces $t(v, w)$ sur les paires de sommets (v, w) de même couleur $j \in \{1, \ldots, k\}$. Ainsi, par exemple, si une fourmi construit la 3-coloration partielle légale représentée à droite de la figure 11.5, elle ajoute $\mid V_2 \mid^2 = 4$ à la trace $t(a, f)$. Ces traces diminuent régulièrement grâce à un facteur d'évaporation ρ. Lorsqu'une fourmi envisage d'attribuer la couleur i à un sommet $v \in V_{k+1}$, elle lui associe une trace $T(v, i)$ définie comme suit :

$$T(v, i) = \sum_{w \in V_i} t(v, w)$$

En notant C l'ensemble des paires (v, i) avec $v \in V_{k+1}$ et $i \in \{1, \ldots, k\}$, chaque fourmi doit donc choisir sa prochaine action dans cet ensemble C. Il est assez standard dans un algorithme de fourmi de choisir une paire $(v, i) \in C$ de façon aléatoire, avec une probabilité $P(v, i)$ de choisir (v, i) définie comme suit :

$$P(v, i) = \frac{A(v, i)^\alpha T(v, i)^\beta}{\sum\limits_{(w, j) \in C} A(w, j)^\alpha T(w, j)^\beta}$$

où α et β sont deux paramètres qui donnent plus ou moins d'importance à l'attrait et à la trace. Pour accélérer leur algorithme, M. Plumettaz *et al.* proposent d'utiliser une autre stratégie de choix. Ils déterminent tout d'abord le sous-ensemble $C' \subseteq C$ de paires (v, i) d'attrait $A(v, i)$ maximum. Si ce sous-ensemble C' contient plus d'une paire, alors celles-ci sont départagées grâce à la trace, c'est-à-dire que l'on choisit la paire (v, i) de C' ayant la plus grande trace $T(v, i)$. En d'autres termes, contrairement à la plupart des algorithmes de fourmis, l'attrait et la trace sont utilisés successivement et non pas simultanément dans le processus de décision. Il en résulte un gain important au niveau du temps de résolution, puisque l'on évite le calcul des probabilités $P(v, i)$ définies ci-dessus.

D'autres ingrédients sont utilisés dans F_{PSZ}, comme par exemple l'interdiction pour un certain nombre d'itérations de décolorer un sommet v lorsque celui-ci vient d'être transféré de l'ensemble V_{k+1} des sommets non colorés à un ensemble V_i. Pour plus de détails sur cet algorithme, le lecteur est invité à consulter [PLU 09].

11.4. Quelques comparaisons numériques

Dans cette section, nous donnons quelques résultats numériques qui permettront de comparer les trois approches décrites précédemment. Nous n'avons bien sûr pas reprogrammé chacun des algorithmes et nous nous contenterons donc de reproduire quelques résultats tirés des articles dans lesquels les algorithmes sont décrits. Le lecteur doit cependant être conscient que ces résultats ont été obtenus sur des ordinateurs très différents les uns des autres, et que la puissance des ordinateurs en 1997 n'était de loin pas comparable à celles des machines d'aujourd'hui. Nous n'indiquerons donc pas de temps de calcul, mais simplement les nombres de couleurs produits par chaque algorithme. Les différences sont cependant si significatives que la comparaison reste malgré tout pertinente.

Nous comparons tout d'abord les divers algorithmes de fourmis sur des *graphes aléatoires*. Ces graphes, notés $G_{n;d}$, ont un nombre n de sommets, et chaque arête a une probabilité d d'exister, indépendamment des autres arêtes. Cette probabilité d est appelée la *densité* du graphe. Ainsi, par exemple, le graphe $G_{100;0,5}$ a environ $d\frac{n(n-1)}{2} = 2\,475$ arêtes. Il est reconnu que les graphes aléatoires les plus difficiles à colorer ont une densité proche de 0,5, ce qui explique que la plupart des algorithmes de coloration ont été testés sur ces graphes. Nous présentons dans le tableau 11.1 quelques résultats pour les graphes $G_{n;0,5}$ avec $n = 100$, 300, 500 et 1 000. Les divers algorithmes testés sont les suivants :

– les algorithmes de fourmis F_{DSATUR}, F_{RLF}, F_{CO}, F_{HZ} et F_{PSZ} décrits dans les sections précédentes. Nous ne donnons aucun résultat pour l'algorithme de fourmis proposé par J. Shawe-Taylor et J. Žerovnik [SHA 02] (voir paragraphe 11.3.3.1) car les auteurs de cet algorithme n'ont effectué des tests numériques que sur un petit échantillon de graphes très particuliers ;

– les algorithmes constructifs DSATUR [BRE 79] et RLF [LEI 79], afin de mesurer la pertinence des extensions qui ont mené aux algorithmes F_{DSATUR} et F_{RLF} ;

– la recherche tabou BZ décrite dans [BLO 08], afin de mesurer les différences avec l'algorithme de fourmis F_{PSZ} inspiré de cette méthode ;

– l'algorithme GH proposé par P. Galinier et J.K. Hao [GAL 99] qui est reconnu comme l'un des meilleurs algorithmes de coloration connu à ce jour.

Etant donné que certains de ces algorithmes résolvent le PCSG, alors que d'autres résolvent une suite de k-PCSGs, nous indiquons pour chaque algorithme de cette deuxième catégorie la plus petite valeur k pour laquelle il a été possible de déterminer une k-coloration légale.

n	F_{RLF}	F_{DSATUR}	F_{HZ}	F_{CO}	F_{PSZ}	RLF	DSATUR	BZ	GH
100	15	15	16	14	14	17	18	14	14
300	35	38	37	34	33	39	42	33	33
500	55	67	57	53	48	60	65	49	48
1 000	111	121	105	99	84	107	114	89	84

Tableau 11.1. *Comparaison pour des graphes aléatoires* $G_{n;0,5}$

Nous observons tout d'abord que les algorithmes de fourmis F_{DSATUR} et F_{RLF} de la première approche sont meilleurs que DSATUR et RLF uniquement sur des graphes de petite taille. En effet, à partir de 500 sommets, F_{DSATUR} utilise plus de couleurs que DSATUR, alors que l'avantage de RLF sur F_{RLF} est un peu plus tardif, puisqu'il n'est visible que sur les graphes à 1000 sommets. Les algorithmes F_{HZ} et F_{CO} de la deuxième catégorie ont considérablement réduit le rôle de chaque fourmi, puisque celles-ci se promènent sur le graphe et ne peuvent qu'influencer le choix de la couleur du sommet sur lequel elles se trouvent. Chaque fourmi n'a donc qu'un pouvoir local, alors que dans les algorithmes de la première catégorie, chaque fourmi pouvait choisir la couleur de chacun des sommets. On observe cependant que, malgré cette diminution de pouvoir des fourmis, elles produisent de meilleurs résultats que ceux obtenus par les algorithmes de la première catégorie. Cependant, en comparant F_{HZ} et F_{CO} avec BZ et GH, on remarque des différences pouvant aller jusqu'à plus de vingt couleurs pour des graphes à 1 000 sommets. Ces algorithmes de fourmis ne sont donc toujours pas compétitifs avec les meilleurs algorithmes de coloration connus à ce jour. L'algorithme F_{PSZ} de la troisième catégorie produit les mêmes résultats que l'algorithme GH et est parfois meilleur que BZ dont il s'est inspiré. Il a donc fallu attendre 2007, soit une bonne dizaine d'années depuis la parution du premier algorithme de fourmis pour le PCSG, pour enfin disposer d'un algorithme de fourmis capable de rivaliser avec les meilleurs algorithmes de coloration des sommets d'un graphe.

Les quatre graphes ci-dessus constituent un échantillon plutôt limité pour comparer des algorithmes. Bien qu'il apparaisse clairement que les algorithmes de fourmis

des deux premières catégories font bien pâle figure face à des algorithmes tels que BZ et GH, il semble pertinent de pousser la comparaison un peu plus loin, pour l'algorithme de fourmis F_{PSZ} de la troisième catégorie.

Graphe	n	$\chi(G)$	k^*	F_{PSZ}	BZ	GH
DSJC1000.1	1 000	-	20	20	21	20
DSJC1000.5	1 000	-	83	84	89	83
DSJC1000.9	1 000	-	224	224	226	224
DSJC500.1	500	-	12	12	12	12
DSJC500.5	500	-	48	48	49	48
DSJC500.9	500	-	126	126	127	126
DSJR500.1c	500	-	85	85	85	-
DSJR500.5	500	-	122	125	125	-
flat1000_50_0	1 000	50	50	50	50	50
flat1000_60_0	1 000	60	60	60	60	60
flat1000_76_0	1 000	76	82	83	87	83
flat300_28_0	300	28	28	29	28	31
le450_15c	450	15	15	15	15	15
le450_15d	450	15	15	15	15	15
le450_25c	450	25	25	26	25	26
le450_25d	450	25	25	25	25	26

Tableau 11.2. *Comparaisons entre F_{PSZ}, BZ et GH*

Nous indiquons dans le tableau 11.2 les meilleures colorations obtenus par F_{PSZ}, BZ et GH sur les instances DIMACS réputées difficiles que l'on peut trouver sur le site *web* à l'adresse http://mat.gsia.cmu.edu/COLOR/instances.html. Pour chacun de ces graphes, nous indiquons son nombre n de sommets, son nombre chromatique $\chi(G)$ s'il est connu, ainsi que le plus petit nombre k^* pour lequel une heuristique de coloration a réussi à produire une k^*-coloration légale du graphe considéré

Nous observons que F_{PSZ} est meilleur que BZ sur six des seize instances testées, alors que l'avantage de BZ sur F_{PSZ} ne peut être observé que sur deux instances. L'adaptation de BZ pour en faire l'algorithme de fourmis F_{PSZ} semble donc pertinente. Lorsque l'on compare F_{PSZ} avec GH, l'avantage est pour F_{PSZ} sur une instance et pour GH sur une autre, les deux algorithmes donnant les mêmes résultats sur les quatorze autres instances.

11.5. Discussion et conclusion

Voilà déjà une décennie que les chercheurs proposent des algorithmes de fourmis pour le PCSG ou le k-PSCG. On peut regrouper les différentes publications dans ce domaine en trois classes, selon le rôle que les fourmis jouent dans l'algorithme. Dans

les premiers algorithmes proposés, chaque fourmi est un algorithme séquentiel de coloration, qui laisse une trace sur chaque paire de sommets non adjacents, pour indiquer si ces sommets ont reçu la même couleur. La deuxième catégorie comprend les algorithmes dans lesquels les fourmis peuvent être assimilées à des pinceaux se promenant dans le graphe, dans le but de modifier la couleur des sommets qu'elles visitent. Finalement, dans la troisième catégorie, les fourmis sont devenues des algorithmes de recherche locale, qui laissent des traces sur l'exploration qu'elles ont faites de l'espace de recherche. Alors que les premiers algorithmes de fourmis n'étaient pas compétitifs avec d'autres métaheuristiques pour la coloration des sommets d'un graphe, les algorithmes les plus récents (ceux de la troisième catégorie) rivalisent positivement avec les meilleurs algorithmes connus à ce jour.

On peut cependant se demander si des algorithmes tels que F_{PSZ} peuvent toujours être considérés comme des algorithmes de fourmis, les déviations par rapport aux algorithmes de fourmis standards étant nombreuses. A titre d'exemple, les algorithmes F_{HZ} et F_{PSZ} utilisent une liste taboue pour éviter de rester bloquer dans des *minima* locaux de l'espace de recherche, ce qui est plutôt un ingrédient d'une recherche tabou. Aussi, un élément qui semble distinguer les algorithmes de fourmis d'une recherche tabou est la façon de réaliser un choix de modification de solution à chaque itération. Dans une recherche tabou, ce choix est souvent le meilleur possible parmi un ensemble de choix candidats, la qualité d'une modification pouvant être mesurée non seulement par la valeur de la fonction à optimiser, mais également par les similitudes entre la solution vers laquelle on tente de se mouvoir et le contenu d'une mémoire. Un algorithme de fourmis ne choisit généralement pas la meilleure modification, puisqu'il réalise un choix aléatoire biaisé par l'attrait et la trace de chaque mouvement possible. Dans l'algorithme F_{PSZ}, une modification (v, i) est choisie parmi celles d'attrait $A(v, i)$ maximum, et la trace $T(v, i)$ ne sert qu'à départager les égalités (voir paragraphe 11.3.3.2 pour les détails). En d'autres termes, l'algorithme F_{PSZ} choisit à chaque itération la modification (v, i) de valeur $A(v, i) + \omega \cdot T(v, i)$ maximale, où ω est une constante suffisamment grande pour donner plus d'importance à l'attrait qu'à la trace. Ce type de choix est plutôt similaire à ce qui se fait dans une recherche tabou que dans un algorithme de fourmis.

La communauté scientifique n'est pas unanime sur le nom à donner aux diverses métaheuristiques existantes. Certains considèrent les algorithmes de fourmis comme des cas particuliers d'autres algorithmes plus anciens, alors que d'autres les considèrent comme des généralisations ou des variations. Pour illustrer ces propos, mentionnons que F. Glover et M. Laguna (p. 4 dans [GLO 97]) précisent qu'à chaque itération d'une recherche tabou, chaque décision peut faire appel aux quatre types de mémoire suivants :

– une mémoire basée sur la *fréquence* permet de tenir compte de caractéristiques que l'on a souvent observées durant le processus de recherche. Une telle caractéristique dans un problème de coloration des sommets d'un graphe peut, par exemple, être

le fait que deux sommets non adjacents ont souvent la même couleur ;

– une mémoire basée sur la *récence* gère les caractéristiques des solutions les plus récentes. Une caractéristique fréquente en début de recherche, mais peu présente lors des dernières itérations, n'est donc pas stockée dans ce type de mémoire. A l'inverse, une nouvelle caractéristique qui n'est apparue que récemment n'est probablement pas encore fréquente, mais peut être qualifiée de récente ;

– une mémoire basée sur l'*influence* a pour objectif de stocker les décisions (d'ajouter une caractéristique) qui ont eu le meilleur impact sur la valeur de la fonction à optimiser ;

– une mémoire basée sur la *qualité* ne se souviendra que des caractéristiques présentes sur les meilleures solutions rencontrées durant la recherche.

Avec une définition si large de l'utilisation possible de la mémoire, certains qualifieront les algorithmes de fourmis comme des cas particuliers d'une recherche tabou. En effet, les traces laissées par les fourmis combinent les quatre types de mémoire susmentionnés : le renforcement d'une trace laissée sur une caractéristique peut être vu comme une mémoire basée sur la fréquence de ces caractéristiques ; ce renforcement est souvent proportionnel à la variation de la fonction à optimiser lors de l'ajout d'une caractéristique (cette variation correspond donc à une influence) ; ce renforcement peut également être proportionnel à la qualité globale de la solution obtenue en ajoutant une caractéristique ; le facteur d'évaporation permet d'oublier les caractéristiques moins récentes. A l'opposé, d'autres prétendront que la liste taboue dans une recherche tabou est une trace qui apparaît lors d'un mouvement vers une solution voisine, et cette trace s'évapore après un nombre fixé d'itérations correspondant à la longueur de la liste taboue.

On pourrait débattre encore longuement de ce sujet. Mais, à titre de conclusion, il nous semble que puisque les chercheurs fourmillent d'idées, il vaut mieux les laisser utiliser leur terminologie préférée, pour autant qu'ils laissent une trace dans les meilleures revues scientifiques et éveillent ainsi la curiosité et suscitent l'intérêt d'autres chercheurs pour le développement d'algorithmes toujours plus attrayants et performants pour la résolution de problèmes d'optimisation combinatoire complexes.

11.6. Bibliographie

[AHN 03] Ahn S., Lee S., Chung T., « Modified Ant Colony System for Coloring Graphs », *ICICS - PCM*, Singapour, 15-18 décembre 2003.

[AVA 03] Avanthay C., Hertz A., Zufferey N., « A Variable Neighborhood Search for Graph Coloring », *European Journal of Operational Research*, vol. 151, p. 379–388, 2003.

[BES 05] Bessedik M., Laib R., Boulmerka A., Drias H., « Ant Colony System for Graph Coloring Problem », M. Mohammadian (dir.), *International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on*

Intelligent Agents, Web Technologies and Internet Commerce, Vienne, Autriche, 28-30 novembre 2005.

[BLO 08] Bloechliger I., Zufferey N., « A Graph Coloring Heuristic Using Partial Solutions and a Reactive Tabu Scheme », *Computers & Operations Research*, vol. 35, p. 960–975, 2008.

[BRE 79] Brélaz D., « New Methods to Color Vertices of a Graph », *Communications of the Association for Computing Machinery*, vol. 22, p. 251–256, 1979.

[CHA 87] Chams D., Hertz A., de Werra D., « Some Experiments with Simulated Annealing for Coloring Graphs », *European Journal of Operational Research*, vol. 32, p. 260–266, 1987.

[COS 97] Costa D., Hertz A., « Ants can colour graphs », *Journal of the Operational Research Society*, vol. 48, p. 295–305, 1997.

[FLE 96] Fleurent C., Ferland J. A., « Genetic and hybrid algorithms for graph coloring », *Annals of Operations Research*, vol. 63, n° 3, p. 437–461, 1996.

[GAL 99] Galinier P., Hao J., « Hybrid Evolutionary Algorithms for Graph Coloring », *Journal of Combinatorial Optimization*, vol. 3, n° 4, p. 379–397, 1999.

[GAL 06] Galinier P., Hertz A., « A Survey of Local Search Methods for Graph Coloring », *Computers & Operations Research*, vol. 33, p. 2547–2562, 2006.

[GAL 08] Galinier P., Hertz A., Zufferey N., « An Adaptive Memory Algorithm for the k-Coloring Problem », *Discrete Applied Mathematics*, vol. 156, n° 2, p. 267–279, janvier 2008.

[GAM 92] Gamst A., Rave W., « On the frequency assignment in mobile automatic telephone systems », *Proceedings of GLOBECOM'92*, 1992.

[GAR 79] Garey M., Johnson D., *Computer and Intractability : a Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[GLO 97] Glover F., Laguna M., *Tabu Search*, Kluwer Academic Publishers, Boston, MA, 1997.

[HER 87] Hertz A., de Werra D., « Using Tabu Search Techniques for Graph Coloring », *Computing*, vol. 39, p. 345–351, 1987.

[HER 00] Hertz A., Kobler D., « A Framework for the description of Evolutionary algorithms », *European Journal of Operational Research*, vol. 126, p. 1–12, 2000.

[HER 02] Herrmann F., Hertz A., « Finding the Chromatic Number by Means of Critical Graphs », *ACM Journal of Experimental Algorithmics*, vol. 7, n° 10, p. 1–9, 2002.

[HER 06] Hertz A., Zufferey N., « A New Ant Colony Algorithm for the Graph Coloring Problem », *Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization, NICSO 2006*, p. 51–60, Grenade, Espagne, 29-30 juin 2006.

[HER 08] Hertz A., Plumettaz M., Zufferey N., « Variable Space Search for Graph Coloring », *Discrete Applied Mathematics*, vol. 156, n° 13, p. 2551–2560, juillet 2008.

[JOH 91] Johnson D., Aragon C., McGeoch L., Schevon C., « Optimization by Simulated Annealing : an Experimental Evaluation, Part II Graph Coloring and Number Partitioning », *Operations Research*, vol. 39, p. 378–406, 1991.

[KIR 83] Kirkpatrick S., Gelatt C. D., Vecchi M., « Optimization by Simulated Annealing », *Science*, vol. 220, n° 5498, p. 671–680, 1983.

[LEI 79] Leighton F. T., « A graph coloring algorithm for large scheduling problems », *Journal of Research of the National Bureau Standard*, vol. 84, p. 489–505, 1979.

[MAL 05] Malaguti E., Monaci M., Toth P., A Metaheuristic Approach for the Vertex Coloring Problem, Rapport n° OR/05/3, Université de Bologne, Italie, 2005.

[MEH 96] Mehrotra A., Trick M., « A Column Generation Approach for Exact Graph Coloring », *INFORMS Journal on Computing*, vol. 8, p. 344–354, 1996.

[MOR 96] Morgenstern C., « Distributed Coloration Neighborhood Search », *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 26, p. 335–357, 1996.

[PET 89] Petford A., Welsh D., « A Randomised 3-colouring Algorithm », *Discrete Mathematics*, vol. 74, p. 253–261, 1989.

[PLU 09] Plumettaz M., Schindl D., Zufferey N., « Ant Local Search and its Efficient Adaptation to Graph Colouring », *Journal of the Operational Research Society*, avril 2009.

[SHA 02] Shawe-Taylor J., Zerovnik J., « Ants and Graph Coloring », *Proceedings of ICANNGA'01*, p. 593–597, 2002.

[STE 85] Stecke K., « Design planning, scheduling and control problems of flexible manufacturing », *Annals of Operations Research*, vol. 3, p. 3–12, 1985.

[VES 00] Vesel A., Zerovnik J., « How Well Can Ants Color Graphs », *Journal of Computing and Information Technology*, vol. 8, n° 2, p. 131–136, 2000.

[ZUF 02] Zufferey N., Heuristiques pour les Problèmes de la Coloration des Sommets d'un Graphe et d'Affectation de Fréquences avec Polarités, thèse de doctorat, Ecole Polytechnique Fédérale de Lausanne (EPFL), Suisse, 2002.

Chapitre 12

Apprentissage des modèles de Markov cachés par l'algorithme API

Les modèles de Markov cachés sont des outils statistiques permettant de modéliser des phénomènes temporels (un signal, une série, etc.) sous la forme de processus stochastiques. Ces modèles sont utilisés avec succès pour résoudre de nombreux problèmes, comme en témoignent les très nombreux travaux les utilisant. Ces modèles trouvent leurs applications dans des domaines aussi variés que la reconnaissance et la synthèse de la parole, la biologie, l'ordonnancement, l'indexation de documents, la reconnaissance d'images, la prédiction de séries temporelles, la robotique, etc.

Dans ce chapitre, nous présentons les modèles de Markov cachés (MMC), ainsi que les algorithmes et problèmes qui leur sont associés. L'apprentissage de MMC constitue le principal sujet de ces travaux. Dans ce cadre, nous nous intéressons à la métaheuristique API, inspirée du comportement de fourragement des fourmis tropicales de l'espèce *Pachycondyla apicalis*. Plusieurs travaux antérieurs ont permis de montrer que cette métaheuristique était efficace sur le problème de l'apprentissage de MMC. Les opérateurs de l'algorithme API sont un moyen d'améliorer encore son efficacité. Nous proposons donc une analyse statistique des opérateurs d'origine et nous montrons comment ceux-ci peuvent être généralisés et redéfinis de manière à améliorer l'efficacité de la métaheuristique.

Chapitre rédigé par Sébastien Aupetit, Nicolas Monmarché et Mohamed Slimane.

12.1. Les modèles de Markov cachés

Les modèles de Markov cachés ont une longue histoire derrière eux. Tout a commencé en 1913, lorsque A.A. Markov a mis au point la théorie des chaînes de Markov [MAR 13]. Mais, ce n'est qu'à partir des années 1960-1970 que la théorie des modèles de Markov cachés, dérivée de la théorie des chaînes de Markov, voit ses principes fondamentaux posés et ses premiers algorithmes efficaces définis [BAU 67, BAU 72, FOR 73, VIT 67]. Depuis lors, de nombreuses variantes des modèles de Markov cachés originels ont été créées et utilisées avec succès dans de nombreuses applications [SLI 02].

12.1.1. *Le jeu du lancer de pièces*

Afin d'exposer le plus simplement possible les principes des modèles de Markov cachés, nous proposons d'étudier un exemple : le jeu du lancer de pièces nécessitant deux joueurs, que nous appellerons Pierre et Paul. Dès le début du jeu, Pierre et Paul se placent de part et d'autre d'un mur opaque : aucun des deux ne peut voir ce que fait l'autre. Pierre ouvre son porte monnaie et y prend trois pièces : une de 1 centime, une de 2 centimes et une de 5 centimes. Ces trois pièces possèdent un défaut de fabrication qui a pour effet de les déséquilibrer. Lorsqu'une de ces pièces tombe au sol, un côté de la pièce est plus souvent visible que l'autre : la pièce est biaisée.

Pour commencer le jeu, Pierre choisit une des trois pièces, la lance vers le ciel et attend que celle-ci tombe au sol. Pierre lit alors la face visible de la pièce et annonce « Face » à Paul. Pierre remet la pièce avec les deux autres, choisit de nouveau une pièce, la lance et annonce « Face » à Paul. Pierre réitère ses actions un certain nombre de fois. De l'autre côté du mur, Paul ne perçoit des activités de Pierre qu'une suite d'annonces « Pile » ou « Face ». L'objectif de Paul est alors de reconstituer les actions de Pierre en fonction de la suite des tirages annoncés : dans quel ordre les pièces ont été choisies et quels sont les biais des pièces ? Pour que Paul puisse reconstituer les actions de Pierre, Paul sait que Pierre joue avec trois pièces et que le jeu impose à Pierre de ne pas choisir les pièces n'importe comment : il doit suivre une loi de probabilité très stricte. Cette loi dit que la probabilité de choix d'une pièce ne dépend que du choix de la pièce précédente et uniquement celle-ci. Finalement, la suite des faces annoncées par Pierre est « FFPPPPPFPFPFFPPPPF » en notant F pour « Face » et P pour « Pile ». La figure 12.1 présente la scène et adopte la notation P_1, P_2 et P_3 pour représenter les trois pièces.

Dans un premier temps, Paul se dit que ce jeu pourrait être modélisé par un modèle de Markov. En effet, les états du système associé pourraient correspondre aux couples (pièce, résultat du lancer). Cependant, l'absence de connaissances sur la séquence des pièces rend difficile, voire impossible, l'estimation des probabilités du modèle (les probabilités des pièces et la loi de probabilités pour le choix des pièces). Une

alternative consiste à modéliser le jeu sous la forme d'un modèle de Markov caché et à apprendre les probabilités du modèle à partir de la séquence annoncée. A partir de ce modèle, Paul peut répondre aux différentes questions du jeu, tout en indiquant une probabilité de vraisemblance des réponses. Il peut alors déterminer le biais des pièces et la séquence la plus probable des pièces, voire prédire les prochaines annonces de Pierre.

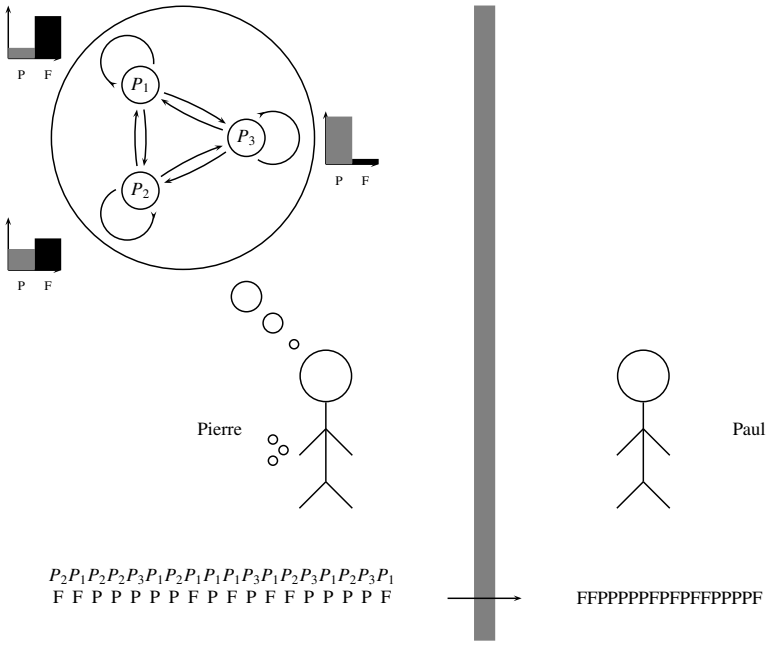


Figure 12.1. *Pierre et Paul jouent avec des pièces*

Admettons maintenant que Paul ait réussi à trouver un modèle suffisamment représentatif du jeu. Pierre propose à Paul de recommencer le jeu. Pierre annonce alors une nouvelle séquence de « Pile » et « Face » à Paul. Connaissant bien Pierre, Paul pense qu'il est possible que Pierre ait changé les pièces d'origine afin de l'induire en erreur. Paul cherche alors à déterminer si Pierre a effectivement changé de pièces ou non. Une solution pour répondre à cette interrogation consiste à utiliser le modèle de Markov caché précédemment déterminé et à calculer la vraisemblance de la nouvelle séquence par rapport au modèle. Si cette probabilité est faible, alors Pierre a changé les pièces du jeu. Dans le cas contraire, la séquence a pu être générée par les pièces du jeu précédent.

Les problèmes que Paul cherche à résoudre sont difficiles, mais pas insolubles, en présence de certaines hypothèses, lorsque des modèles de Markov cachés peuvent être utilisés.

12.1.2. *Définitions*

Dans la suite, nous restreignons l'étude au cas particulier des modèles de Markov cachés discrets du premier ordre stationnaire. Un modèle de Markov caché discret du premier ordre correspond à la modélisation de deux processus stochastiques : un processus caché, modélisé par une chaîne de Markov discrète, et un processus observé, dépendant des états du processus caché. Soient $\mathbb{S} = \{s_1, \ldots, s_N\}$ l'ensemble fini et dénombrable des N états dits « cachés » du système et $S = (S_1, \ldots, S_T)$ un T-uple de variables aléatoires (v.a.) définies sur l'ensemble $\mathbb{S}$. Soient $\mathbb{V} = \{v_1, \ldots, v_M\}$ l'ensemble fini et dénombrable des M symboles dits « émissibles » (ou observables) par le système et $V = (V_1, \ldots, V_T)$ un T-uple de v.a. définies sur $\mathbb{V}$. Un modèle de Markov caché discret du premier ordre stationnaire est alors défini par les probabilités (indépendantes du temps $t \geq 1$) suivantes :

– les probabilités d'initialisation des états cachés : $P(S_1 = s_i)$, que l'on note π_i ;

– les probabilités de transition entre états cachés du premier ordre : $P(S_{t+1} = s_j | S_t = s_i$, que l'on note $a_{i,j}$;

– les probabilités d'émission d'un symbole particulier pour chacun des états cachés : $P(V_t = v_j | S_t = s_i)$, que l'on note $b_i(j)$.

On peut alors définir, pour tout $t > 1$ quelconque (car nous considérons le cas stationnaire), les matrices $A = (a_{i,j})_{1 \leq i,j \leq N}$ et $B = (b_i(j))_{1 \leq i \leq N, 1 \leq j \leq M}$, ainsi que le vecteur colonne $\Pi = (\pi_1, \ldots, \pi_N)'$. Un modèle de Markov caché discret stationnaire du premier ordre, noté λ, est donc complètement défini par le triplet (A, B, Π). Par la suite, nous utiliserons la notation $\lambda = (A, B, \Pi)$ et nous emploierons le terme modèles de Markov cachés (MMC) pour des modèles de Markov cachés stationnaires du premier ordre. L'ensemble des modèles de Markov cachés possibles pour un nombre d'états cachés N et un nombre de symboles M fixés sera noté Λ. On a donc $\lambda \in \Lambda$.

Soit $Q = (q_1, \ldots, q_T) \in \mathbb{S}^T$ une séquence d'états cachés et $O = (o_1, \ldots, o_T) \in \mathbb{V}^T$ une séquence de symboles observés, usuellement appelée « séquence d'observations ». La probabilité de réalisation simultanée de la séquence d'états cachés Q et de la séquence d'observations O par rapport au modèle de Markov caché λ est formulée de la façon suivante[1] : $P(V = O, S = Q | A, B, \Pi) = P(V = O, S = Q | \lambda)$. Cette probabilité peut être obtenue en considérant les dépendances conditionnelles des probabilités définissant le modèle. Il s'ensuit :

$$\begin{aligned} P(V = O, S = Q|\lambda) &= P(V = O|S = Q, \lambda) P(S = Q|\lambda) \\ &= \left(\prod_{t=1}^{T} b_{q_t}(o_t) \right) \cdot \left(\pi_{q_1} \prod_{t=1}^{T-1} a_{q_t, q_{t+1}} \right) \end{aligned} \tag{12.1}$$

1. Formellement, il faudrait considérer le modèle de Markov caché λ comme la réalisation d'une v.a. $P(V = O, S = Q | l = \lambda)$, mais, afin de simplifier les notations, nous avons choisi d'omettre la variable aléatoire exprimant le modèle.

A partir d'un MMC λ, d'une séquence d'états cachés Q et d'une séquence d'observations O, il est possible de calculer l'adéquation entre le modèle λ et les deux séquences Q et O. Pour cela, il suffit de calculer la probabilité $P(V = O, S = Q|\lambda)$. Cette dernière correspond à la probabilité que la séquence d'observations O ait effectivement été engendrée par le modèle λ en suivant la séquence d'états cachés Q.

Lorsque la séquence d'états cachés n'est pas connue, il est possible d'évaluer la vraisemblance d'une séquence d'observations O par rapport à un modèle λ. La vraisemblance correspond à la probabilité $P(V = O|\lambda)$ que la séquence d'observations ait été engendrée par le modèle pour l'ensemble des séquences d'états cachés possibles. On remarque alors que la formule suivante est vérifiée :

$$P(V = O|\lambda) = \sum_{Q \in \mathbb{S}^T} P(V = O, S = Q|\lambda) \tag{12.2}$$

Pour exploiter efficacement les modèles de Markov cachés, il est nécessaire de pouvoir résoudre trois problèmes fondamentaux pour un modèle λ :

1) évaluer la vraisemblance, $P(V = O|\lambda)$, d'une séquence d'observations O par rapport au modèle λ ;

2) déterminer la séquence Q^* d'états cachés qui a le plus probablement été suivie pour générer la séquence d'observations O ;

3) apprendre/ajuster un ou plusieurs MMC à partir d'une ou plusieurs séquences d'observations, lorsque l'on connaît le nombre d'états cachés du ou des MMC.

La vraisemblance peut être efficacement obtenue par l'algorithme Forward ou par l'algorithme Backward avec une complexité en $\theta(N^2T)$ [RAB 89]. En pratique, il est nécessaire d'avoir recours à des techniques de rééchelonnement de valeurs pour ne pas aboutir à des dépassements de capacité (*overflow* ou *underflow*) des nombres manipulés. Ainsi, la vraisemblance est couramment manipulée sous la forme de son logarithme : $\ln P(V = O/\lambda)$. Cette valeur est appelée « log-vraisemblance ».

De multiples séquences Q^* d'états cachés peuvent répondre au deuxième problème : tout dépend du critère à satisfaire. Usuellement, le critère utilisé est défini par $P(V = O, S = Q^*|\lambda) = \max_{Q \in \mathbb{S}^T} P(V = O, S = Q|\lambda)$. Dans ce cas, une telle séquence peut être efficacement déterminée par l'algorithme de Viterbi avec une complexité en $\theta(N^2T)$ [VIT 67]. Des techniques de rééchelonnement des valeurs doivent être utilisées, pour les mêmes raisons que précédemment.

Enfin l'apprentissage de MMC peut être vu comme un problème de maximisation d'un critère particulier, sous des contraintes de stochasticité des matrices A, B et Π du ou des modèles. Dans le cas général, les MMC permettent de modéliser des phénomènes dont une partie peut être observée. Par exemple, dans le cas d'une image représentant une voiture, la partie observable correspond aux pixels, tandis que la partie

non observable concerne l'organisation générale de l'image. L'étape d'apprentissage est alors primordiale et consiste à ajuster le modèle à la séquence des observations, de manière à conceptualiser la voiture. Si le modèle obtenu est un bon modèle de la voiture, alors il est possible de l'utiliser pour reconnaître une image contenant une voiture.

L'exploitation d'un MMC pour la reconnaissance de concept dépend de trois aspects : la constitution de la séquence d'observations à apprendre, l'apprentissage de la séquence et l'exploitation du modèle. Dans ce chapitre, nous ne nous intéressons qu'à l'apprentissage d'une séquence. Les deux autres aspects sont très fortement dépendants des domaines d'applications visés (image, signal, comportement, etc.). Quels que soient ces domaines, l'étape de conceptualisation par l'apprentissage est primordiale et dépend principalement du critère d'apprentissage utilisé.

Les critères d'apprentissage de MMC se répartissent en cinq catégories principales : les critères du type « maximisation de la vraisemblance » [BAU 67, GAN 99, KAP 98], les critères du type « maximisation de la probabilité *a posteriori* », les critères du type « maximisation de l'information mutuelle » [GIU 02, RAB 89, SCH 97, VER 04], les critères du type « minimisation du taux d'erreur de classification » [GAN 99, LJO 90, SAU 00] et le critère de *segmental k-means* [JUA 90].

Ces critères ne sont pas les seuls envisageables, mais ils correspondent à ceux qui sont les plus souvent utilisés. Pour la majorité de ces critères, il existe au moins un algorithme permettant, à partir d'un modèle initial, de trouver une suite de modèles convergeant vers un optimum local du critère. Dans certaines applications, ces optima locaux sont suffisants, mais ce n'est pas toujours le cas. Il est donc nécessaire de pouvoir s'approcher le plus possible des modèles optimaux du critère. Une possibilité consiste à utiliser des métaheuristiques pour explorer l'espace des modèles de Markov cachés.

Le critère qui va nous intéresser dans la suite est un critère du type « maximisation de la vraisemblance ». Ces critères consistent, dans leur forme la plus simple, à trouver un MMC λ^* vérifiant l'équation : $P(V = O|\lambda^*) = \max_{\lambda \in \Lambda} P(V = O|\lambda)$. Malgré la simplicité de ce critère, aucune méthode exacte et générale n'existe pour l'optimiser. Néanmoins, les algorithmes de Baum-Welch (BW) [BAU 67] et de descente de gradient [GAN 99, KAP 98] permettent d'améliorer itérativement un modèle initial. Cependant, la suite des MMC obtenus converge vers un optimum local du critère. Il est d'ores et déjà important de noter que l'apprentissage à l'aide de ce critère est un problème complexe et que la majorité des autres critères conduisent à des problèmes encore plus difficiles.

12.1.3. *Métaheuristiques pour l'apprentissage de modèles de Markov cachés*

Les premières recherches sur l'apprentissage de MMC à l'aide de métaheuristiques datent des années 1980. En 1985, les travaux de D.B. Paul [PAU 85] à l'aide du recuit simulé marquaient le début de la recherche dans ce domaine.

En raison d'un grand nombre d'applications dans le domaine de la parole et de l'ADN, de nombreux travaux se sont contentés de MMC « gauche-droite »[2]. Les travaux de Y. Hamam et T. Al Ani [HAM 96] avec du recuit simulé, de B. Maxwell et S. Anderson [MAX 99] avec un apprentissage incrémental à base de population, de T.Y. Chen *et al* [CHE 04] avec la recherche tabou ou de S. Kwong et C. Chau [KWO 97] avec des algorithmes génétiques en sont des exemples.

Avec l'augmentation des capacités des ordinateurs, il est devenu plus facile d'exploiter des métaheuristiques à base de population et de les hybrider avec des algorithmes d'optimisation locale, tels que l'algorithme de Baum-Welch ou une descente de gradient. Ainsi les travaux de M. Slimane *et al* [SLI 99], R. Thomsen [THO 02] et S. Aupetit [AUP 05] ont cherché à exploiter cette hybridation avec des algorithmes génétiques. L'algorithme de colonie de fourmis API [MON 00a, MON 02] a été adapté à plusieurs reprises sur le problème de l'apprentissage de MMC [AUP 05, AUP 06, AUP 07, AUP 08, MON 00a]. Avec le même type d'approche, l'optimisation par essaim particulaire [CLE 05, KEN 95] a également été adaptée à ce problème [AUP 05, AUP 07, AUP 08, RAS 03].

La majorité des adaptations de métaheuristiques que l'on peut trouver dans la littérature se restreignent à la forme la plus basique des critères qui sont du type maximisation de la vraisemblance. Dans ce cas, le critère se réduit à la maximisation, sous des contraintes de stochasticité des variables, d'un polynôme à plusieurs variables. Deux constats peuvent être faits :

1) le degré du polynôme dépend linéairement de la longueur de la séquence d'observations ;

2) à partir d'un MMC, il est possible d'en construire d'autres, par renumérotation des états, qui ont une valeur identique du critère. Par conséquent, l'espace de recherche comporte un nombre important d'optima locaux.

Nos travaux récents [AUP 05, AUP 06, AUP 07, AUP 08] ont été menés avec ce même critère. Ils ont permis de proposer de nouvelles adaptations de métaheuristiques (algorithme génétique, algorithme de colonie de fourmis API et optimisation par essaim particulaire) et surtout de mener diverses études comparatives des méthodes,

2. La matrice de transition entre états cachés A est triangulaire supérieure.

après en avoir déterminé des paramètres efficaces dans la majorité des cas. Les principales conclusions tirées de ces travaux sont reprises ici : l'hybridation d'une métaheuristique avec l'algorithme de Baum-Welch (BW) est profitable et permet d'obtenir des modèles significativement meilleurs que ceux que l'on obtient en utilisant l'algorithme de Baum-Welch seul. L'hybridation d'une métaheuristique avec l'algorithme BW n'est efficace que si le nombre d'itérations de l'algorithme BW est suffisant, mais limité : deux itérations de l'algorithme sont en général suffisantes. Nous avons également pu remarquer que l'algorithme de colonie de fourmis API permettait d'obtenir des modèles compétitifs avec les autres métaheuristiques, quel que soit le nombre de solutions évaluées dans l'espace de recherche.

Nous consacrons la suite de ce chapitre à la présentation de l'algorithme de colonie de fourmis API et à l'étude de moyens permettant d'améliorer son efficacité. Cette étude se situe dans le cadre de l'apprentissage des modèles de Markov cachés.

12.2. L'algorithme de colonie de fourmis API

Les applications les plus efficaces des algorithmes inspirés du comportement de fourmis ont permis d'établir les principes généraux de ces algorithmes. Dans la plupart des cas, une mémoire globale partagée est mise en œuvre pour guider la recherche des agents vers des solutions prometteuses. La matérialisation de cette mémoire s'effectue de façon similaire à ce que font les vraies fourmis : par le dépôt de substances volatiles appelées phéromones sur les chemins conduisant à de la nourriture ou à de bonnes solutions. Ce mécanisme est connu sous le nom de stigmergie [DOR 00]. La métaheuristique *Ant Colony Optimization* (ACO) [BON 99] est un exemple de formalisation et d'application efficace de ces principes (voir chapitres 2 et 3).

Contrairement à ce que l'on pourrait croire en consultant la littérature, notamment en optimisation combinatoire, toutes les colonies de fourmis ne régissent pas leur vie et leurs comportements en utilisant des phéromones. Certaines colonies, déclarées comme primitives, n'utilisent pas de phéromone, car elle sont en général moins populeuses et n'ont donc pas eu besoin de mettre en œuvre ce type de recrutement de masse. Les fourmis ponérines *Pachycondyla apicalis* [FRE 85, FRE 94] en sont un exemple. La modélisation du comportement de ces fourmis pour des problèmes d'optimisation a conduit à la création de la métaheuristique API [AUP 06, MON 00a, MON 00b]. Ces fourmis n'utilisent pas de phéromone, mais sont capables de réaliser une exploration coordonnée de l'espace de recherche, en adoptant un point de vue à la fois local et global sur l'espace. En sortant du nid, les fourmis couvrent une surface donnée en la partitionnant en sites de chasse. Chaque fourmi explore le ou les sites de chasse qu'elle a choisi(s), en tenant compte des succès et échecs de capture de proies pour chaque site. De plus, la métaheuristique API fait partie des algorithmes [DRE 04, MON 00b, SOC 08] capables d'exploiter un espace de recherche à variables continues (voir chapitre 4 du volume 1, pour plus de détails).

Les principes généraux de la métaheuristique API ont été définis de façon précise par les travaux de N. Monmarché [MON 00a, MON 00b]. Cependant, leur mise en œuvre concrète laisse une grande latitude, permettant une modulation du comportement de l'algorithme et une grande adaptabilité de l'algorithme au problème à résoudre. C'est dans cette flexibilité que réside un potentiel non négligeable d'amélioration des performances originelles de l'algorithme. Dans la suite de cette section, nous présentons les bases et principes de la métaheuristique API, puis nous montrons comment la forme classique [MON 00a] de la métaheuristique API peut être étendue et modifiée sans remettre en cause ses principes fondamentaux.

12.2.1. *La stratégie de fourragement des fourmis Pachycondyla apicalis*

Les fourmis *Pachycondyla apicalis* vivent habituellement dans la forêt tropicale mexicaine à proximité des côtes du Guatemala [FRE 85, FRE 94]. Leurs colonies sont constituées d'une vingtaine à une centaine d'individus, dont seulement 20 % à 30 % partent chasser des proies hors du nid.

La stratégie globale de recherche de proies peut se résumer comme suit. Les fourmis choisissent des sites de chasse aléatoirement et uniformément dans un périmètre d'environ dix mètres autour du nid. Chaque site de chasse a un rayon approximatif de 2,5 mètres. Ce partitionnement du périmètre du nid en sites de chasse permet aux fourmis de globalement le couvrir. Périodiquement, le nid de la colonie est déplacé. Ces déplacements peuvent être expliqués par le fait que le nid devient de moins en moins confortable et stable au fil du temps ou par un appauvrissement du périmètre du nid en proies. Le mécanisme mis en œuvre par les fourmis pour le déplacement du nid est relativement complexe et s'appuie sur des fourmis spécialisées dans la recherche d'un nouveau nid, ainsi que sur un mécanisme de recrutement (recrutement en tandem : *tandem running*).

La stratégie locale de fourragement d'une fourmi est définie comme suit. Initialement, une fourmi choisit aléatoirement un site de chasse à proximité du nid. Lorsque la fourmi réussit à capturer une proie sur un site de chasse, elle mémorise ce site de chasse, en considérant cette capture comme un succès (renforcement positif) et ramène la proie au nid. Par la suite, la fourmi a tendance à retourner sur les derniers sites de chasse sur lesquels elle avait eu des succès de capture, en suivant les mêmes chemins. Pour suivre un chemin, la fourmi n'utilise pas de phéromone, mais des repères visuels. Lorsqu'une proie est capturée sur un site, la fourmi y retourne systématiquement la fois suivante (à sa prochaine sortie du nid). Quand un site de chasse s'appauvrit en proies et que la fourmi n'arrive plus à y capturer de nouvelles proies, la fourmi abandonne le site et part explorer un autre site de chasse, qu'il soit nouveau ou qu'il soit un des sites mémorisés par la fourmi.

12.2.2. *Les principes fondamentaux de l'algorithme API*

Rappelons les principes de cette métaheuristique. Le lecteur notera dès maintenant que la métaheuristique API présentée ici est une forme simplifiée de la métaheuristique initiale. En effet, nous avons montré expérimentalement dans [AUP 05] qu'il était équivalent d'augmenter la taille de la mémoire de chaque fourmi au lieu d'augmenter le nombre de fourmis. En conséquence, afin de diminuer le nombre de paramètres à régler, nous ne donnons aux fourmis qu'un seul site de chasse en mémoire.

Dans le cas général, le problème considéré est sous la forme d'une fonction $f : \mathbb{S} \rightarrow \mathbb{R}$ à maximiser. L'espace de recherche $\mathbb{S}$ peut être, par exemple, un espace continu ($\mathbb{S} \equiv \mathbb{R}^{\ell}$), un espace binaire ($\mathbb{S} = \{0, 1\}^{\ell}$) ou un espace de permutations, comme dans le problème du voyageur de commerce. Comme nous le verrons plus loin, aucune contrainte particulière n'est imposée sur la nature de cet espace de solutions. Quoi qu'il en soit, un point s de $\mathbb{S}$ doit être une solution valide du problème considéré.

Soient $a_1, \ldots, a_n$ une population de n fourmis constituant la colonie. Chaque fourmi est associée à une position dans l'espace de recherche $\mathbb{S}$ et cherche à maximiser la fonction f. Pour définir la métaheuristique API et donc la stratégie de fourragement des fourmis, il est nécessaire de définir deux opérateurs :

– O_{init} qui a pour objectif de définir la position initiale du nid dans l'espace $\mathbb{S}$;

– O_{explo} qui, à partir d'un point $s \in \mathbb{S}$, génère un point $s' \in \mathbb{S}$ situé dans le voisinage de s. Cet opérateur dépend habituellement d'une valeur $\mathcal{A} \in [0; 1]$ définissant la taille du voisinage (si possible relativement à l'étendue de l'espace des solutions, de manière à ne pas dépendre de la taille de ce dernier). Cette valeur est appelée « amplitude d'exploration ».

Il est important de mentionner que les deux opérateurs O_{init} et O_{explo} ne nécessitent pas d'être complètement aléatoires. O_{init} peut donner une bonne position initiale du nid en utilisant une heuristique ou une autre métaheuristique, et O_{explo} peut incorporer des heuristiques spécifiques au problème, telles que des descentes de gradient.

La métaheuristique API peut être décrite de la façon suivante. Au commencement, la position du nid $\mathcal{N}$ est définie grâce à l'opérateur O_{init}. Le nid $\mathcal{N}$ est déplacé tous les $\mathcal{T}_{\text{depl}}$ mouvements des n fourmis[3] et est positionné sur la meilleure position connue par les fourmis depuis le début de l'algorithme. Par conséquent, le déplacement du nid s'effectue tous les $n \times \mathcal{T}_{\text{depl}}$ mouvements individuels des fourmis.

Chaque fois qu'une fourmi a_i a besoin de choisir un site de chasse, elle quitte le nid et choisit le site dans le voisinage du nid, grâce à l'opérateur O_{explo}. Cet opérateur

3. A chaque itération de l'algorithme, toutes les fourmis effectuent un mouvement simultanément.

est en général paramétré spécifiquement pour chaque fourmi. Lorsqu'une fourmi a_i a besoin d'explorer le voisinage d'un site de chasse s_i, elle calcule une position p du voisinage de s_i en utilisant l'opérateur O_{explo}. Tout comme précédemment, cet opérateur est en général spécifiquement paramétré pour chaque fourmi. Une exploration locale est dite réussie si elle conduit à une meilleure valeur de la fonction f à maximiser, c'est-à-dire si $f(p) > f(s_i)$. Lors d'une exploration locale réussie, la fourmi remplace son site de chasse par la nouvelle position p. Dans le cas contraire, le site de chasse est conservé. Si un site de chasse s_i a été exploré plus de P_i^{local} fois consécutives sans succès, alors le site de chasse est abandonné et oublié. La prochaine action de la fourmi sera alors de quitter le nid à la recherche d'un nouveau site de chasse. Finalement, lorsque le nid est déplacé, les fourmis oublient tous leurs sites de chasse (cela améliore la capacité de l'algorithme à éviter les minima locaux).

La coopération des fourmis au sein de la métaheuristique API n'est pas aussi évidente que dans nombre d'algorithmes utilisant des phéromones (tels que la métaheuristique ACO). La coopération s'effectue de façon implicite à travers le déplacement du nid et l'exploration par voisinage. Le déplacement du nid peut être vu comme la mise en commun de l'effort individuel des fourmis pour la recherche d'un nouveau nid. Le déplacement du nid vers un meilleur endroit de l'espace profite alors à toute la colonie. En effet, les explorations de l'espace par les fourmis sont toutes liées à la position du nid et donc liées aux efforts de chacun des agents.

La forme générale de la métaheuristique API est donnée par l'algorithme 12.1 et est illustrée par la figure 12.2. L'adaptation de cette métaheuristique à un problème particulier s'effectue alors en définissant les opérateurs O_{init} et O_{explo}.

L'opérateur O_{init}, bien que spécifique à l'application visée, consiste usuellement à choisir une position aléatoirement dans l'espace $\mathbb{S}$. On a classiquement $O_{\text{init}} = \mathcal{R}(\mathbb{S})$ avec $\mathcal{R}$ l'opérateur retournant de façon aléatoire un élément de l'ensemble fourni en paramètre. Dans la section suivante, nous nous focalisons sur l'étude de l'opérateur O_{explo} sous sa forme classique, sa forme classique généralisée et sous une nouvelle forme.

12.2.3. *Mise en œuvre classique de l'opérateur O_{explo} et généralisation*

Le choix d'un site de chasse par la fourmi a_i est le résultat de l'opérateur $O_{\text{explo}}(\mathcal{N}, \text{site}, a_i)$, défini par :

$$O_{\text{explo}}(\mathcal{N}, \text{site}, a_i) = \mathcal{R}\left(\mathcal{V}\left(\mathcal{N}, \mathcal{A}_i^{\text{site}}\right)\right) \tag{12.3}$$

$\mathcal{V}(x, r)$ représente le voisinage de la position $x \in \mathbb{S}$ d'amplitude $r \geq 0$, c'est-à-dire l'ensemble des éléments de $\mathbb{S}$ situés à une amplitude maximale r de x. L'amplitude représente alors la quantité de modification maximale applicable à la position x. Dans le

Algorithme 12.1 La métaheuristique API.

Choisir la position initiale du nid : $\mathcal{N} = O_{\text{init}}$
pour $t = 1$ **à** $\mathcal{T}_{\max}$ **faire**
  pour chaque fourmi a_i **faire**
    si la fourmi a_i n'a pas de site de chasse **alors**
      La fourmi construit un site de chasse : $s_i \leftarrow O_{\text{explo}}(\mathcal{N}, \text{site}, a_i)$
      $\text{NbEchec}_i \leftarrow 0$
    sinon
      Construire une nouvelle solution p dans le voisinage du site de chasse s_i de la fourmi a_i : $p \leftarrow O_{\text{explo}}(s_i, \text{local}, a_i)$
      si $f(p) > f(s_i)$ **alors**
        Remplacer le site de chasse par la nouvelle solution : $s_i \leftarrow p$
        $\text{NbEchec}_i \leftarrow 0$
      sinon
        $\text{NbEchec}_i \leftarrow \text{NbEchec}_i + 1$
        si $\text{NbEchec}_i \geq P_i^{\text{local}}$ **alors**
          Abandonner et oublier le site de chasse
        fin si
      fin si
    fin si
  fin pour
  si t **mod** $\mathcal{T}_{\text{depl}} = 0$ **alors**
    Déplacer le nid sur la meilleure solution connue
    Vider la mémoire de toutes les fourmis
  fin si
fin pour

cas où $\mathbb{S}$ est un espace réel, l'ensemble $\mathcal{V}(x, r)$ peut, par exemple, être vu comme une boule centrée sur le point x et de rayon r. L'amplitude correspond alors à une distance maximale. Dans le cas où $\mathbb{S}$ est un espace de permutation, $\mathcal{V}(x, r)$ peut, par exemple, être l'ensemble des permutations atteignables à partir de x en réalisant au plus une quantité proportionnelle à r d'échanges. Dans toute la suite, nous considérerons que les amplitudes sont normalisées et qu'elles prennent leurs valeurs dans l'intervalle $[0; 1]$. Dans ce cas, on a $\mathcal{V}(x, 0) = \{x\}$ et $\mathcal{V}(x, 1) = \mathbb{S}$ quel que soit le point $x \in \mathbb{S}$. On note également $\mathcal{A}_i^{\text{site}} \geq 0$ l'amplitude de choix du site de chasse de la fourmi a_i.

L'opérateur d'exploration locale d'un site de chasse s_i pour une fourmi a_i a une forme similaire :

$$O_{\text{explo}}(s_i, \text{local}, a_i) = \mathcal{R}\left(\mathcal{V}\left(s_i, \mathcal{A}_i^{\text{local}}\right)\right) \tag{12.4}$$

$\mathcal{A}_i^{\text{local}} \geq 0$ représente alors l'amplitude locale d'un site de chasse, c'est-à-dire la taille du voisinage de s_i que la fourmi explore.

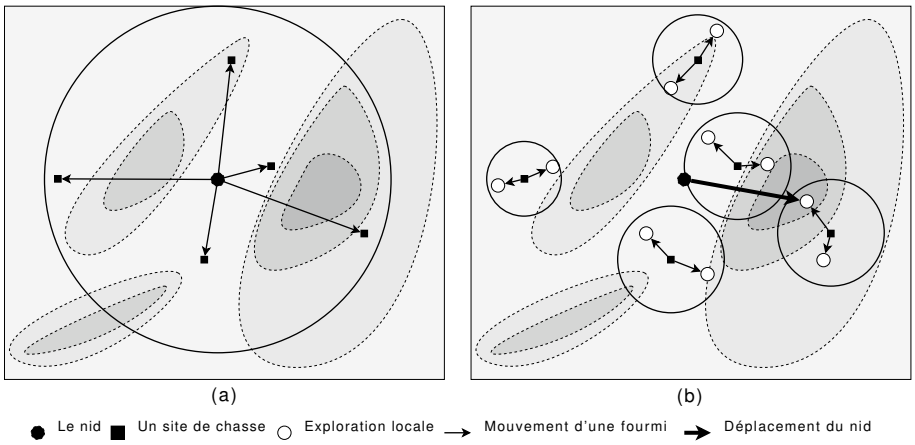


Figure 12.2. *Les deux principales étapes de la métaheuristique API : (a) les fourmis quittent le nid et choisissent leurs sites de chasse, (b) les fourmis explorent leurs sites de chasse et finalement le nid est déplacé sur la meilleure position connue*

12.2.3.1. *Définition classique des amplitudes*

Lors des multiples expérimentations qui ont conduit à la mise au point de la métaheuristique API [MON 00a] et lors de son utilisation [AUP 05], plusieurs définitions des valeurs des amplitudes $\mathcal{A}_i^{\text{site}}$ et $\mathcal{A}_i^{\text{local}}$ ont été considérées. Il s'est rapidement avéré que l'utilisation d'une population de fourmis ne présentant pas un paramétrage uniforme menait à des stratégies plus robustes. Le paramétrage hétérogène des fourmis s'est vite imposé. La définition qui s'est montrée expérimentalement efficace est $\mathcal{A}_i^{\text{site}} = 100^{\frac{i-n}{n}}$ et $\mathcal{A}_i^{\text{local}} = \mathcal{A}_i^{\text{site}}/10$ pour $i = 1..n$. En revanche, les patiences P_i^{local} des fourmis (correspondant au nombre d'échecs consécutifs autorisés pour un site de chasse, avant sa remise en cause par une fourmi) ont été considérées fixes et identiques.

Une telle définition des amplitudes n'est pas anodine. En effet, elle définit la distribution des sites de chasse explorés par la colonie dans l'espace. Pour mettre en évidence cette distribution, considérons $\mathbb{S}$ comme un espace réel et $\mathcal{V}(x, r)$ comme une boule de centre x et de rayon r. La fonction $\mathcal{A}_i^{\text{site}}$ et la distribution des distances des sites de chasse sont données par la figure 12.3. Comme nous pouvons le constater, les sites de chasse d'une grande partie des fourmis sont en majorité proches du nid, ce qui permet une exploitation efficace de l'information transmise par la position du nid. A l'opposé, peu de fourmis sont chargées de l'exploration de solutions éloignées du nid. Ce choix peut paraître judicieux, dans le sens où il est peu probable de trouver de bonnes solutions très loin du nid, mais il est quand même intéressant de chercher loin du nid, pour éviter de rester bloqué sur des optima locaux.

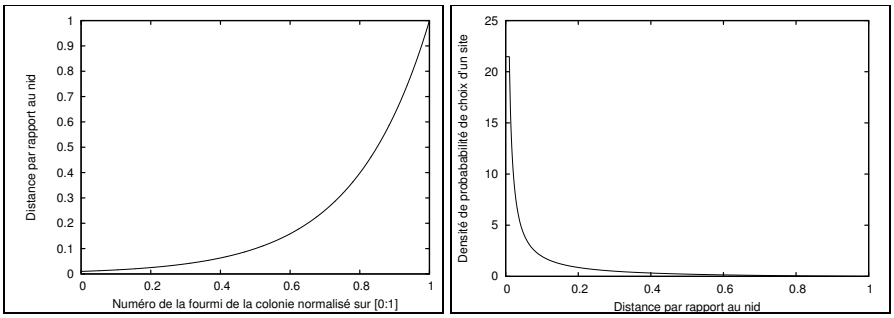


Figure 12.3. *Fonction $\mathcal{A}_i^{site}$ (à gauche) et distribution des distances des sites de chasse de la colonie par rapport au nid (à droite), dans le cas de la définition classique des amplitudes*

Dès lors, il est possible de se demander, sans remettre en cause les opérateurs définis plus haut, s'il est possible de moduler la répartition des fourmis dans l'espace et de moduler la distribution des sites de chasse.

12.2.3.2. *Généralisation des définitions classiques des amplitudes*

Nous proposons de paramétrer la fonction $\mathcal{A}_i^{\text{site}}$ par une fonction de la forme $a + bZ^{i/n-1}$ à valeurs dans l'intervalle $[0.01; 1]$ et dépendant de $Z > 0$. On obtient :

$$\mathcal{A}_i^{\text{site}} = \begin{cases} 1 + (0.01 - 1)\frac{i-N}{1-N} & \text{si } Z = 1 \\ 1 + \frac{1-0.01}{1-Z^{1/n-1}}\left(Z^{i/n-1} - 1\right) & \text{sinon} \end{cases} \tag{12.5}$$

Les évolutions de la fonction $\mathcal{A}_i^{\text{site}}$ et de la distribution des sites de chasse en fonction de Z sont données par les figures 12.4 et 12.5. Comme nous pouvons le constater, la définition de la fonction $\mathcal{A}_i^{\text{site}}$ en fonction de Z permet de moduler la distribution des distances des sites de chasse, d'une stratégie hyper-concentrée sur le nid (Z est grand) à une stratégie quasi uniforme (du point de vue de la distance) sur l'espace de recherche (Z proche de 0).

Cette modulation possède néanmoins un gros inconvénient. En raison de la forme de l'opérateur $O_{\text{explo}}(\cdot, \text{site}, \cdot)$, le voisinage de chasse de la colonie correspond à l'union de voisinages de même centre vérifiant $\mathcal{V}\left(\mathcal{N}, \mathcal{A}_i^{\text{site}}\right) \subseteq \mathcal{V}\left(\mathcal{N}, \mathcal{A}_{i+1}^{\text{site}}\right)$ dans le cas général[4]. Cette structure de voisinage de la colonie implique que le périmètre d'action de chaque fourmi peut être partiellement ou totalement inclus dans le périmètre d'action

4. Cette propriété n'est pas requise par la métaheuristique, mais elle est fortement conseillée, pour avoir une cohérence avec la définition classique de l'opérateur $O_{\text{explo}}(\cdot, \text{site}, \cdot)$.

d'une autre fourmi (voir figure 12.6). On peut remarquer que, lorsque Z tend vers 0, le voisinage de choix des sites pour toutes les fourmis devient l'espace des solutions et que, lorsque Z tend vers l'infini, les fourmis n'explorent quasiment qu'un voisinage extrêmement resserré autour du nid.

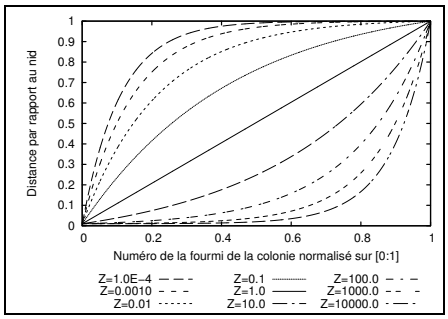


Figure 12.4. *Fonction $\mathcal{A}_i^{site}$ en fonction de Z*

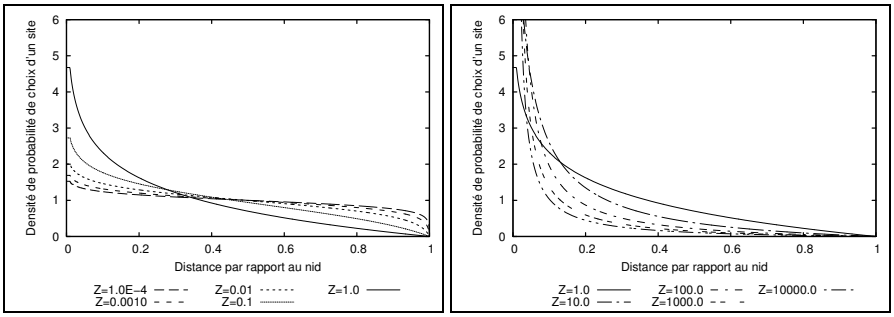


Figure 12.5. *Les distributions initiales des sites de chasse de la colonie par rapport au nid en fonction de Z, pour l'opérateur $O_{explo}(\cdot, site, \cdot)$ classique*

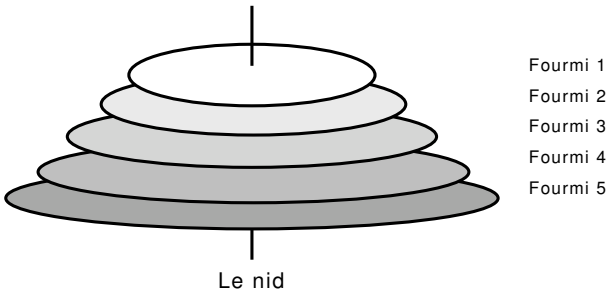


Figure 12.6. *Empilement des voisinages de choix des sites de chasse, pour la définition classique des opérateurs*

Dans l'optique où la répartition des fourmis dans l'espace peut être contrôlée, il devient nécessaire d'étudier l'impact de la patience des fourmis. Est-il intéressant ou non de fixer des patiences différentes aux fourmis ? Rappelons que la patience d'une fourmi est le nombre maximal d'échecs successifs autorisés pour un site de chasse. Ainsi, une patience faible signifie que la fourmi n'a le droit qu'à peu d'échecs avant de remettre en cause un site de chasse. Cette patience est directement dépendante de deux choses : la probabilité de succès d'une exploration pour le site de chasse et la difficulté du problème. En considérant l'empilement des voisinages de choix des sites de chasse de la colonie, on constate que l'espace des solutions est divisé en n morceaux (ou couches concentriques), notés $F_1, \ldots, F_n$ (voir figure 12.6), définis par :

$$F_i = \begin{cases} \mathcal{V}\left(\mathcal{N}, \mathcal{A}_1^{\text{site}}\right) & \text{si } i = 1 \\ \mathcal{V}\left(\mathcal{N}, \mathcal{A}_i^{\text{site}}\right) - \mathcal{V}\left(\mathcal{N}, \mathcal{A}_{i-1}^{\text{site}}\right) & \text{si } i > 1 \end{cases} \tag{12.6}$$

La patience moyenne appliquée sur chacun des morceaux par les fourmis est :

$$\bar{P}_i = \frac{\sum_{k=i}^{n} P_k^{\text{local}}}{n - i + 1} \tag{12.7}$$

S'il est possible de choisir $\bar{P}_i$ de façon idéale pour le problème à résoudre, il suffit de définir $P_i^{\text{local}} = \sum_{k=i}^{n}(n - k + 1)(-1)^{k-i}\bar{P}_k$. La difficulté de cette approche réside dans le fait que la patience d'une fourmi a_i est identique, quel que soit le morceau dans lequel se trouve le site de chasse. Par conséquent, sur un morceau F_i, les fourmis explorent des sites de chasse avec une patience inférieure, égale ou supérieure à la patience $\bar{P}_i$. Par conséquent, sur chaque morceau F_i, certaines fourmis sont trop ou pas assez patientes. Du fait de la relative difficulté de donner un sens à la patience avec la définition classique de l'opérateur O_{explo}, on considère habituellement toutes les patiences des fourmis comme égales.

Dans la nature, les fourmis ont tendance à se spécialiser pour une couche particulière : les fourmis les plus âgées osent chasser beaucoup plus loin que les fourmis plus jeunes. Cette remarque, et la difficulté de donner un sens à la patience des fourmis dans les définitions précédentes, nous conduisent à la définition de nouveaux opérateurs.

12.2.4. *L'opérateur O_{explo} en peau d'oignon*

Le nouvel opérateur O_{explo} doit permettre de contrôler simplement la répartition des sites de chasse et la patience pour chaque fourmi. Pour cela, nous proposons de nous inspirer de la structure d'un oignon. Un oignon est constitué de multiples couches de peau concentriques centrées sur le cœur de l'oignon. Supposons que l'espace des solutions $\mathbb{S}$ soit décomposable en couches concentriques centrées sur une position quelconque. Dans ce cas, si une fourmi est affectée à chaque couche, alors il devient

possible de contrôler la répartition des sites de chasse, en définissant la position et l'épaisseur des couches de peau. De même, cette approche permet de contrôler finement la patience des fourmis sur chaque couche de peau.

Pour cela, nous proposons de reprendre la structure en morceaux définie plus haut par les ensembles F_i ($i = 1..n$). On note ΔF_i l'épaisseur de la couche F_i, définie par :

$$\Delta F_i = \begin{cases} \mathcal{A}_1^{\text{site}} & \text{si } i = 1 \\ \mathcal{A}_i^{\text{site}} - \mathcal{A}_{i-1}^{\text{site}} & \text{si } i > 1 \end{cases} \tag{12.8}$$

Le choix d'un site de chasse par la fourmi a_i est le résultat de l'opérateur défini par : $O_{\text{explo}}(\mathcal{N}, \text{site}, a_i) = \mathcal{R}(F_i)$. Pour comprendre le fonctionnement de cet opérateur, il est important de garder à l'esprit qu'une seule fourmi est affectée à l'exploration d'une couche de peau. Ainsi, la probabilité que la colonie choisisse un site de chasse $x \in \mathbb{S}$ particulier (du point de vue de la distance au nid) est directement liée à la couche dans laquelle le point se trouve. Supposons que le point x se trouve dans la couche F_k. La probabilité que la colonie choisisse ce site est directement proportionnelle au quotient $1/\Delta F_k$. Lorsque la couche est peu épaisse, alors la probabilité que la colonie choisisse un site de chasse dans celle-ci est élevée. Inversement, une couche épaisse implique une probabilité de choix faible.

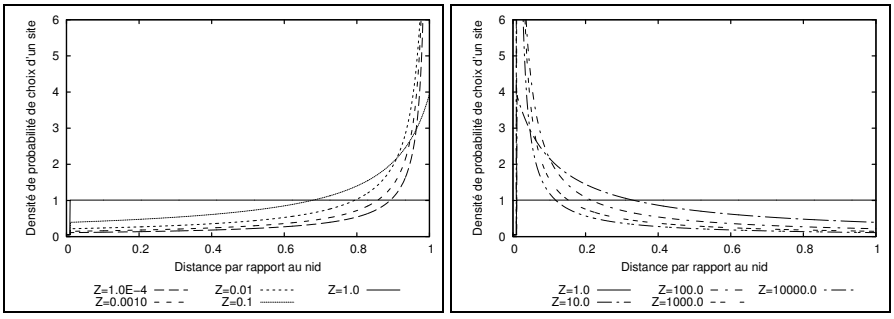


Figure 12.7. *Les distributions des sites de chasse de la colonie par rapport au nid pour le nouvel opérateur $O_{explo}(\cdot, site, \cdot)$ en fonction de Z*

Pour définir l'épaisseur des couches de peau, nous proposons de reprendre la même définition que précédemment pour $\mathcal{A}_i^{\text{site}}$. On a :

$$\mathcal{A}_i^{\text{site}} = \begin{cases} 1 + (0.01 - 1)\frac{i-N}{1-N} & \text{si } Z = 1 \\ 1 + \frac{1-0.01}{1-Z^{1/n-1}}\left(Z^{i/n-1} - 1\right) & \text{sinon} \end{cases} \tag{12.9}$$

Dans ce cas, l'évolution de la fonction $\mathcal{A}_i^{\text{site}}$ et de la distribution des sites de chasse en fonction de Z sont données par les figures 12.4 et 12.7. Comme nous pouvons

le constater, la définition de ce nouvel opérateur et l'utilisation de cette amplitude permettent un contrôle plus flexible que précédemment de la répartition des efforts de recherche de la colonie. Il est possible de concentrer les efforts à proximité du nid, de façon homogène sur l'espace, ou de concentrer les efforts loin du nid. Cette dernière option pourrait être particulièrement intéressante, dans le cas de la définition d'une forme adaptative de la métaheuristique, pour la mise en œuvre de mécanismes de sortie de bassin d'attraction.

Pour que les couches F_i soient correctement définies, il faut que $\mathcal{N} \in F_1$ et $\bigcup_{i=1}^{n} F_i = \mathbb{S}$. Dans le cas idéal, il faudrait que $F_i \cap F_j = \emptyset$, pour tout i et j dans $1..n$, afin de garantir qu'une zone de l'espace est explorée au maximum par une seule fourmi. En pratique, on pourra se satisfaire du cas $F_i \cap F_j \neq \emptyset$, si cet ensemble est de taille négligeable par rapport à la taille des ensembles F_i et F_j.

L'opérateur d'exploration locale d'un site de chasse s_i pour une fourmi a_i est identique au cas de la définition classique de l'opérateur. On a :

$$O_{\text{explo}}(s_i, \text{local}, a_i) = \mathcal{R}\left(\mathcal{V}\left(s_i, \mathcal{A}_i^{\text{local}}\right)\right) \tag{12.10}$$

$\mathcal{A}_i^{\text{local}} \geq 0$ est l'amplitude locale du site de chasse de la fourmi et est de la forme $\gamma \Delta F_i$. Soit $f_i = \bigcup_{s \in F_i} \mathcal{V}\left(s, \mathcal{A}_i^{\text{local}}\right)$ où γ est un paramètre. En concordance avec la remarque du paragraphe précédent sur le non-recouvrement des morceaux, il faut que $f_i \cap f_{i+1}$ soit de taille négligeable par rapport à la taille des ensembles f_i et f_{i+1}.

Contrairement à la définition classique des opérateurs, les nouveaux opérateurs permettent de contrôler et spécifier finement la patience des fourmis sur les différentes couches. En partant du principe que la patience d'une fourmi doit être directement liée à la probabilité qu'il y ait une meilleure solution dans le voisinage d'un site et que cette dernière probabilité varie de façon monotone avec l'éloignement du site de chasse par rapport au nid, on peut définir de façon arbitraire la patience. Soit $P_{\text{inner}}^{\text{local}}$ la patience de la fourmi a_1 et $P_{\text{outer}}^{\text{local}}$ la patience de la fourmi a_n. Soit $Y > 0$. La patience des fourmis peut être paramétrée de façon similaire aux amplitudes de choix de site en utilisant :

$$P_i^{\text{local}} = \begin{cases} P_{\text{outer}}^{\text{local}} + (P_{\text{inner}}^{\text{local}} - P_{\text{outer}}^{\text{local}}) \frac{i-N}{1-N} & \text{si } Y = 1 \\ P_{\text{outer}}^{\text{local}} + \frac{P_{\text{outer}}^{\text{local}} - P_{\text{inner}}^{\text{local}}}{1-Y^{1/n-1}} \left(Y^{i/n-1} - 1\right) & \text{sinon} \end{cases} \tag{12.11}$$

12.3. Adaptation de la métaheuristique API à l'apprentissage de modèles de Markov cachés

Dans la suite, nous considérons l'apprentissage de MMC pour le critère du maximum de vraisemblance. En considérant les définitions et notations précédentes, il suffit de définir la notion de voisinage $\mathcal{V}(x, r) \subseteq \Lambda$ ou de couche $F_i \subseteq \Lambda$ pour adapter la métaheuristique API à l'apprentissage de modèles de Markov cachés.

Pour résoudre le problème d'apprentissage de MMC à l'aide de la métaheuristique, nous utilisons usuellement un opérateur de voisinage dont la définition est simple. Celui-ci consiste à ajouter, à chacun des coefficients du modèle, une valeur aléatoire, dont l'importance dépend de l'amplitude, puis à transformer les coefficients obtenus, de manière à retrouver des contraintes stochastiques (plus de détails peuvent être trouvés dans [AUP 05]). De par sa définition, cet opérateur est difficilement utilisable de façon cohérente avec le nouvel opérateur d'exploration en peau d'oignon. En lieu et place, nous proposons d'exploiter la structure d'espace vectoriel sur l'ensemble des MMC, telle que définie dans [AUP 05], et dont les principes sont rappelés ci-après.

12.3.1. *Adaptation du voisinage et des couches au problème*

Pour définir la notion de voisinage d'amplitude r pour un modèle de Markov caché, il est nécessaire de rappeler les principales propriétés permettant de munir l'espace des MMC d'une structure vectorielle. Pour cela, nous devons d'abord introduire la notion de voisinage d'un vecteur stochastique dans un espace muni d'une structure vectorielle.

12.3.1.1. *Structure vectorielle sur un ensemble de vecteurs stochastiques*

Soit $\mathbb{G}_K$ l'ensemble des vecteurs stochastiques de dimension K. Rappelons que tout vecteur stochastique $X = (x_1, \ldots, x_K)' \in \mathbb{G}_K$ vérifie les propriétés suivantes :

$$\forall\ i = 1..K,\ x_i \geq 0 \qquad \text{et} \qquad \sum\nolimits_{i=1}^{K} x_i = 1 \tag{12.12}$$

Soit $\mathbb{G}_K^* \subset \mathbb{G}_K$ l'ensemble des vecteurs stochastiques de dimension K dont aucune des composantes n'est nulle. Soient $\oplus : \mathbb{G}_K^* \times \mathbb{G}_K^* \mapsto \mathbb{G}_K^*$, $\odot : \mathbb{R} \times \mathbb{G}_K^* \mapsto \mathbb{G}_K^*$ et $\|.\| : \mathbb{G}_K \mapsto \mathbb{R}^+$ les fonctions définies par :

$$(X \oplus Y)_i = \frac{x_i y_i}{\sum_{j=1}^{K} x_j y_j}, \qquad (c \odot X)_i = \frac{x_i^c}{\sum_{j=1}^{K} x_j^c} \qquad \text{et} \qquad \|X\| = -\ln \frac{\min_{i=1..K} x_i}{\max_{i=1..K} x_i} \tag{12.13}$$

On peut alors montrer que $(\mathbb{G}_K^*, \oplus, \odot)$ est un espace vectoriel et que $\|\cdot\|$ est une norme sur cet espace.

Nous proposons d'utiliser la notion de boule sur l'espace vectoriel $(\mathbb{G}_K^*, \oplus, \odot)$ en utilisant la norme $\|\cdot\|$ pour définir la notion de voisinage. Soit $\mathcal{B}_{\mathbb{G}_K^*}(X, R)$ la boule de centre X et de rayon $R \geq 0$ telle que :

$$\mathcal{B}_{\mathbb{G}_K^*}(X, R) = \{Z \in \mathbb{G}_K^* \,|\, \exists Y \in \mathbb{G}_K^* \text{ tel que } \|Y\| \leq R \text{ et } Z = X \oplus Y\} \tag{12.14}$$

Pour définir un voisinage efficace, il est nécessaire de prendre en compte les particularités de la norme $\|\cdot\|$ et donc de s'intéresser au segment de droite vectorielle $\mathcal{D}_{\mathbb{G}_K^*}(Y, D)$ de direction $Y \in \mathbb{G}_K^*$ et de support $D \subset \mathbb{R}^+$, défini par :

$$\mathcal{D}_{\mathbb{G}_K^*}(Y, D) = \{d \odot Y | d \in D\} \subset \mathbb{G}_K^* \tag{12.15}$$

Soit $\rho : \mathbb{G}_K^* \mapsto [\frac{1}{K}; 1]$ la fonction telle que $\rho(X) = \max_{k=1..K} x_k$. Si $Y \in \mathbb{G}_K^*$ est un vecteur unitaire ($\|Y\| = 1$), alors on montre trivialement que $\rho(d \odot Y)$ est une fonction strictement croissante pour le paramètre $d \in \mathbb{R}$. Pour garantir une exploration efficace de l'espace de recherche et notamment garantir que tout vecteur stochastique est atteignable à partir d'un vecteur Y, il suffit de garantir que $\rho(d \odot Y)$ puisse atteindre une valeur suffisamment proche de 1. Soit $\epsilon > 0$ un réel positif proche de 0. On montre facilement que, pour avoir $\rho(d \odot Y) \geq 1 - \epsilon$, il suffit de considérer $d \geq D_K = \ln \frac{(K-1)(1-\epsilon)}{\epsilon}$. Dans ce cas, la somme des autres composantes du vecteur est donc inférieure à ϵ. Par conséquent le segment $\mathcal{D}_{\mathbb{G}_K^*}(Y, [0; D_K])$ est inclus et quasiment égal à la demie droite $\mathcal{D}_{\mathbb{G}_K^*}(Y, \mathbb{R}^+)$. Pour $\epsilon > 0$ suffisamment petit, il est donc inutile de considérer les vecteurs $d \odot Y$ tels que $d \geq D_K$, car ceux-ci seront quasiment égaux à $D_K \odot Y$. A partir de ces constatations, nous pouvons définir le voisinage $\mathcal{V}_{\mathbb{G}_K^*}(X, r)$ d'un vecteur stochastique X de rayon r en prenant :

$$\mathcal{V}_{\mathbb{G}_K^*}(X, r) = \mathcal{B}_{\mathbb{G}_K^*}(X, r \cdot D_K) \tag{12.16}$$

Dans la suite, nous considérons que $\epsilon = 0.001$. Du fait des imprécisions des calculs numériques, nous obtenons en pratique $\mathcal{V}_{\mathbb{G}_K}(X, r) = \mathcal{V}_{\mathbb{G}_K^*}(X, r)$.

12.3.1.2. *Voisinage sur l'ensemble des modèles de Markov cachés*

L'extension de ce voisinage aux modèles de Markov cachés s'effectue facilement en considérant un MMC comme formé de plusieurs vecteurs stochastiques et en définissant la norme d'un MMC $\lambda = (\Pi, A, B) \in \Lambda$ par :

$$\|\lambda\| = \max_{i=1..N} \{\|\Pi\|, \|A_{i,\cdot}\|, \|B_{i,\cdot}\|\} \tag{12.17}$$

Le voisinage $\mathcal{V}(\lambda, r)$ du MMC λ d'amplitude $r \geq 0$ est alors le produit cartésien des voisinages d'amplitude r de chacun de ses vecteurs stochastiques, c'est-à-dire :

$$\mathcal{V}(\lambda, r) = \mathcal{V}_{\mathbb{G}_N}(\Pi, r) \times \left(\bigtimes_{i=1}^{N} \mathcal{V}_{\mathbb{G}_N}(A_{i,\cdot}, r) \right) \times \left(\bigtimes_{i=1}^{N} \mathcal{V}_{\mathbb{G}_M}(B_{i,\cdot}, r) \right) \tag{12.18}$$

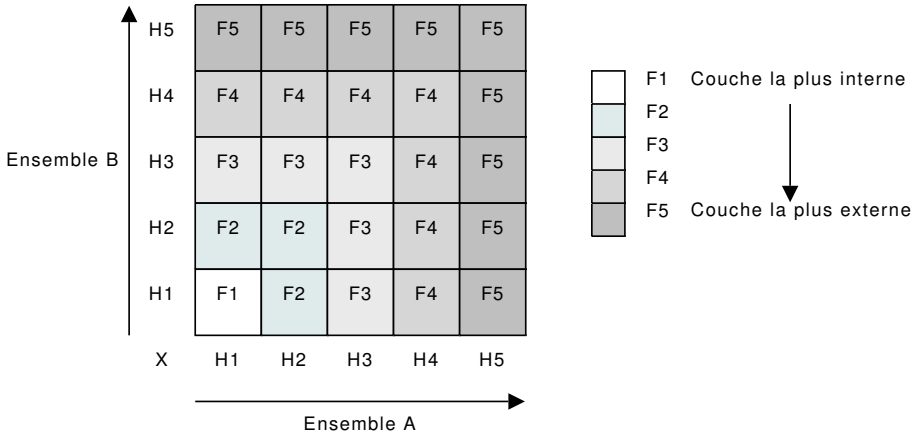


Figure 12.8. *Structure en couches induite par les ensembles H_i. A et B sont deux ensembles de vecteurs. Les ensembles sont partitionnés respectivement en $(H_i(A,X))_{i=1..5}$ et $(H_i(B,X))_{i=1..5}$ à partir du centre X. La norme du couple (a,b) défini sur $A \times B$ est $\|(a,b)\| = \max\{\|a\|, \|b\|\}$*

12.3.1.3. *Structuration en couches de l'ensemble des modèles de Markov cachés*

Nous avons vu dans les sections précédentes qu'une façon simple de définir la notion de couches était d'utiliser les voisinages avec la relation suivante, pour tout $i = 1..n$:

$$F_i = \begin{cases} \mathcal{V}\left(\mathcal{N}, \mathcal{A}_1^{\text{site}}\right) & \text{si } i = 1 \\ \mathcal{V}\left(\mathcal{N}, \mathcal{A}_i^{\text{site}}\right) - \mathcal{V}\left(\mathcal{N}, \mathcal{A}_{i-1}^{\text{site}}\right) & \text{si } i > 1 \end{cases} \tag{12.19}$$

De par la définition de $\mathcal{V}(\lambda, r)$, l'utilisation de F_i sous cette forme reste difficile en pratique. Il est nécessaire d'adopter une autre stratégie pour définir les ensembles $(F_i)_{i=1..n}$.

Soit $(H_i(\mathbb{G}_K^*, X))_{i=1..n}$ le partitionnement de l'espace $\mathbb{G}_K^*$ en n couches concentriques de centre $X \in \mathbb{G}_K^*$, tel que :

$$H_1(\mathbb{G}_K^*, X) = \mathcal{V}_{\mathbb{G}_K^*}(X, \mathcal{A}_1^{\text{site}}) \tag{12.20}$$

c'est-à-dire :

$$H_1(\mathbb{G}_K^*, X) = \left\{ Z \in \mathbb{G}_K^* \middle| \exists Y \in \mathbb{G}_K^* \text{ tel que } \|Y\| \leq \mathcal{A}_1^{\text{site}} \cdot D_K \text{ et } Z = X \oplus Y \right\} \tag{12.21}$$

et, pour tout $i = 2..n$:

$$H_i(\mathbb{G}_K^*, X) = \left\{ Z \in \mathbb{G}_K^* \middle| \exists Y \in \mathbb{G}_K^* \text{ tel que } \mathcal{A}_{i-1}^{\text{site}} \cdot D_K < \|Y\| \leq \mathcal{A}_i^{\text{site}} \cdot D_K \text{ et } Z = X \oplus Y \right\} \tag{12.22}$$

A partir de ce partitionnement, il est possible de construire un partitionnement de l'espace des MMC en couches concentriques, en exploitant la norme $||\lambda||$ d'un MMC définie plus haut (voir figure 12.8). En considérant le centre $\lambda = (\Pi, A, B)$, on obtient :

$$F_i = \bigcup_{\substack{(p,\alpha_1,\ldots,\alpha_N,\beta_1,\ldots,\beta_N)\in[\![1;n]\!]^{N\times N+1} \\ \text{tels que } \max\{p,\alpha_1,\ldots\beta_N\}=i}} H_p(\mathbb{G}_N^*, \Pi) \times \left(\bigtimes_{j=1}^{N} H_{\alpha_j}(\mathbb{G}_N^*, A_{j,\cdot}) \right) \times \left(\bigtimes_{j=1}^{N} H_{\beta_j}(\mathbb{G}_M^*, B_{j,\cdot}) \right) \tag{12.23}$$

Dans ce cas, les propriétés minimales $\lambda \in F_1$ et $\bigcup_{i=1}^{n} F_i = \Lambda^*$ sont vérifiées. La propriété souhaitée $F_i \cap F_j = \emptyset$ est également vraie. De plus, du fait des limites de précision de calcul, l'espace réellement atteint par ces définitions est Λ.

12.3.2. *Etudes expérimentales des adaptations*

Dans la suite de ce chapitre, notre attention se porte sur l'étude de l'impact des opérateurs d'exploration sur l'efficacité de la métaheuristique pour l'apprentissage de MMC. Etant donnée la durée totale d'apprentissage induite par une exploration quasi systématique des différents paramètres et pour réduire notre étude à l'impact des opérateurs d'exploration, nous avons arbitrairement fixé certains paramètres à des valeurs classiques. Il faut dès lors noter que ces choix n'ont aucune influence sur la difficulté de l'apprentissage. Nous supposons que la colonie est constituée de $n = 20$ fourmis. Le nid de la colonie est déplacé toutes les $\mathcal{T}_{\text{depl}} = 8$ itérations de l'algorithme, c'est-à-dire toutes les $8 * 20 = 160$ évaluations de MMC dans l'espace des solutions. Le nombre d'itérations de l'algorithme est fixé à 100. Le paramètre γ a pour valeur 0,1 pour être cohérent avec le rapport entre les amplitudes de choix des sites et les amplitudes locales des fourmis pour l'opérateur classique. Comme nous l'avons évoqué dans les sections précédentes, l'hybridation de la méthode de recherche avec l'algorithme de Baum-Welch est souvent très fructueuse. Nous l'avons donc considérée dans nos expérimentations. Cette hybridation est appliquée après chaque choix (choix d'un site, choix pour l'exploration locale) d'un MMC par l'algorithme. Chaque solution choisie est remplacée par la solution obtenue par l'application de deux itérations de l'algorithme de Baum-Welch.

Nous avons considéré deux séquences d'observations codées sur 256 symboles et issues de deux images en niveaux de gris. La première image, contenant le visage d'une personne, est issue de la base de visages ORL [AT 06] et a pour dimensions 92×112. La seconde image, représentant des voitures dans une rue, est issue de [AGA 04] et a pour dimensions 235×176. Ces deux images sont transformées en séquence, suivant le procédé décrit par la figure 12.9. Celui-ci consiste à découper l'image en blocs de 10×10 pixels, puis à concaténer les lignes de chacun de ces blocs, en considérant les lignes de haut en bas. Finalement, les séquences obtenues pour chacun des blocs sont concaténées en considérant les blocs de gauche à droite puis de

haut en bas. Les niveaux de gris de l'image correspondent alors aux symboles. Les MMC explorés par l'algorithme API ont dix états cachés et 256 symboles.

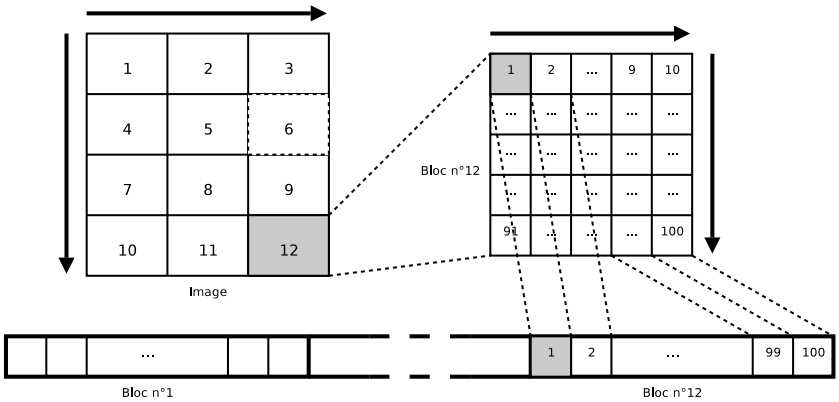


Figure 12.9. *Encodage d'une image en une séquence d'observations*

Dans un premier temps, nous considérons que la patience est identique pour toutes les fourmis. Dans ce cas, le comportement de la métaheuristique ne dépend plus que de deux paramètres : le facteur Z et la patience des fourmis. Pour les expérimentations, nous faisons varier Z logarithmiquement en base 10 dans l'intervalle $[-6;6]$, c'est-à-dire de 10^{-6} à 10^{6}. P_i^{local} prend alors une valeur dans l'ensemble $\{1,2,3,4\}$. Pour chacun de ces paramètres et pour chacune des images, trente apprentissages ont été réalisés. Les figures 12.10 et 12.11 montrent les résultats de ces expérimentations. Plusieurs constatations peuvent alors être faites.

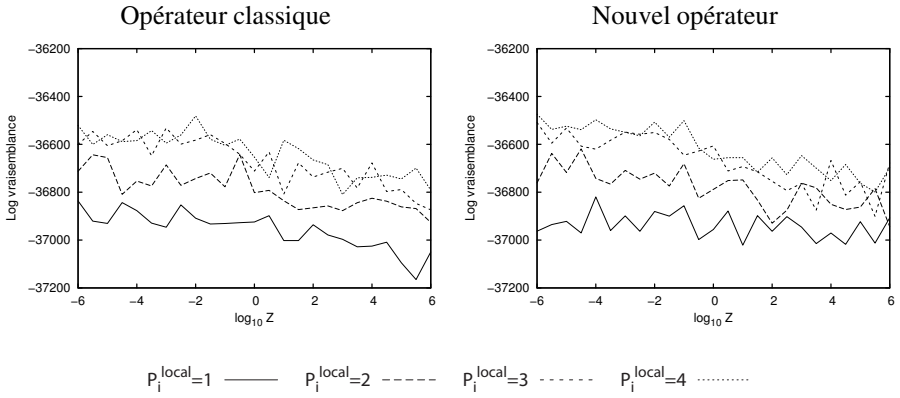


Figure 12.10. *Moyenne des log-vraisemblances obtenues pour la première image*

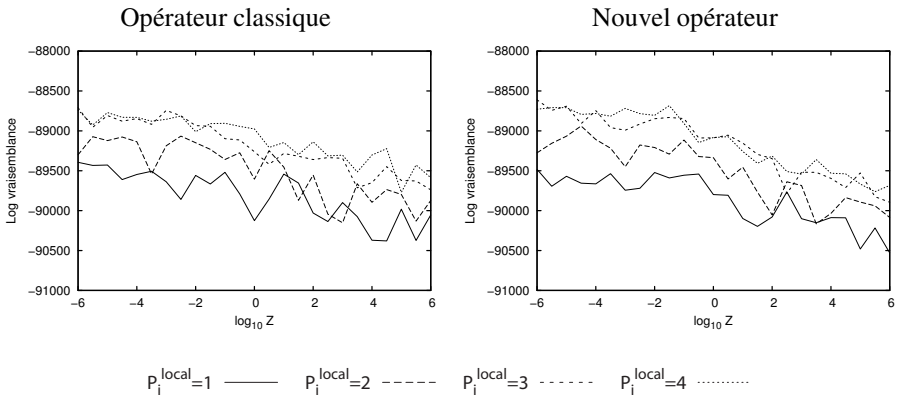


Figure 12.11. *Moyenne des log-vraisemblances obtenues pour la deuxième image*

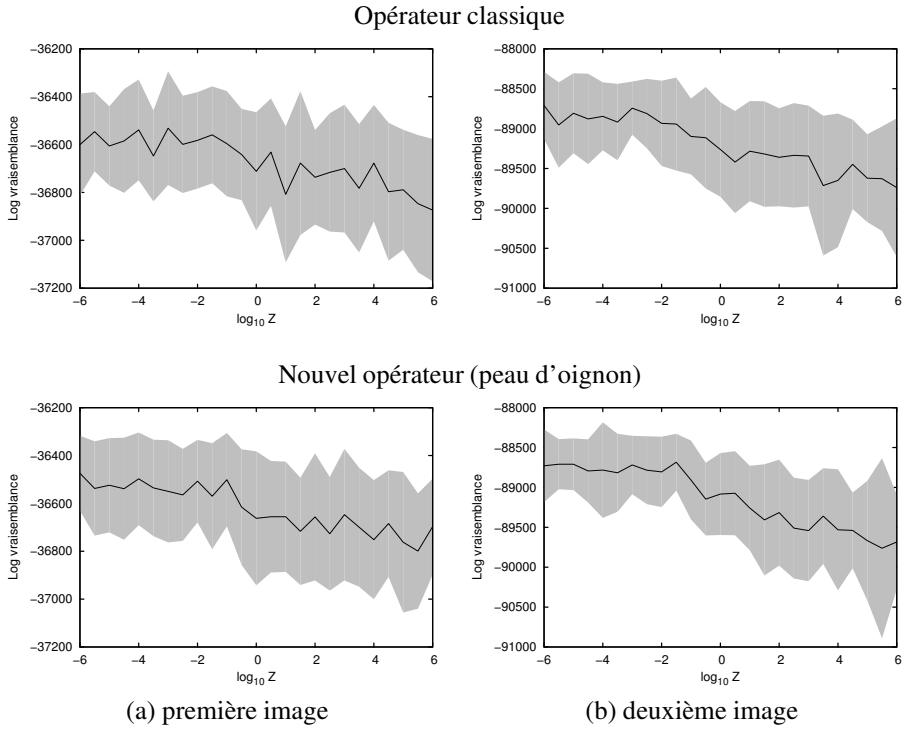
La performance de la métaheuristique API est directement liée à la patience des fourmis : plus la patience est élevée, meilleures sont les performances. Ainsi, l'hypothèse qu'« une fourmi patiente est beaucoup plus utile qu'une fourmi impatiente si le problème est difficile » est vérifiée. De plus, pour une patience équivalente et une répartition similaire[5] de l'effort de recherche de la colonie dans l'espace, les deux opérateurs ont une efficacité équivalente.

Les performances de la métaheuristique augmentent lorsque Z diminue. Dans le cas de l'opérateur classique, les efforts d'exploration doivent donc être répartis sur tout l'espace uniformément, alors que, dans le cas du nouvel opérateur, les efforts doivent être concentrés surtout loin du nid. Au premier abord, ce comportement peut paraître en contradiction avec la métaheuristique API, dont le fonctionnement classique consiste à plus explorer des solutions proches du nid. Cependant, il est nécessaire de rappeler que, dans le cas de l'apprentissage de MMC, la métaheuristique est hybridée avec l'algorithme de Baum-Welch. Cette hybridation permet de corriger les positions, à la fois des sites de chasse et des solutions locales explorées. L'hybridation permet d'accélérer et d'améliorer significativement l'efficacité de l'apprentissage, mais peut avoir pour inconvénient de ramener les fourmis quasi systématiquement dans les mêmes bassins d'attraction, ces bassins d'attraction conduisant toujours vers les mêmes optima locaux. Pour contrebalancer les effets néfastes introduits par l'hybridation, il est donc nécessaire d'explorer plus loin du nid. La généralisation des

5. Les valeurs de Z ne correspondent pas entre les deux opérateurs. Une répartition uniforme de l'effort de recherche correspond à $Z > 0$ petit pour l'opérateur classique, alors qu'elle correspond à $Z = 1$ dans le cas du nouvel opérateur. Dans les deux cas, lorsque Z tend vers l'infini, l'effort de la colonie se concentre sur le nid.

opérateurs d'origine de API a donc permis d'améliorer les résultats, dans ce cas particulier de l'apprentissage d'images.

Nous pouvons donc conclure que la définition originelle de l'opérateur (correspondant à $Z = 100$) est probablement sous optimale pour l'apprentissage de MMC.



(a) première image (b) deuxième image

Figure 12.12. *Log-vraisemblances pour la première image (colonne a) et pour la deuxième image (colonne b), avec une patience fixée à 4 (la courbe en trait plein est la moyenne de la log-vraisemblance, la zone grisée représente la variance)*

Intéressons-nous au cas particulier d'une patience locale de 4. La figure 12.12 présentent les moyennes et écarts types associés. On remarque alors que l'écart-type de la performance diminue lorsque la performance augmente. Des résultats similaires apparaissent pour une patience de 1, 2 ou 3. Par conséquent, le paramétrage de la distribution des efforts de la colonie est donc très intéressant à la fois pour augmenter la performance et pour réduire la variabilité de l'apprentissage de MMC. Nous pouvons également remarquer que, en raison de sa plus grande flexibilité dans la définition de la répartition des efforts de la colonie, le nouvel opérateur est capable d'atteindre des performances supérieures à celles de l'opérateur classique.

12.4. Conclusion et perspectives

Dans ces travaux, nous nous sommes intéressés au problème de l'apprentissage de modèles de Markov cachés (MMC) pour le critère du maximum de vraisemblance. Cet apprentissage s'exprime sous la forme d'un problème d'optimisation continue sous contraintes possédant de très nombreux optima locaux. Pour résoudre ce problème et obtenir un optimum local aussi proche que possible d'un maximum de la fonction, nous considérons la métaheuristique API. Cette métaheuristique, inspirée du comportement de fourragement des fourmis tropicales *Pachycondyla apicalis* et n'utilisant pas de phéromone, a déjà montré une certaine efficacité dans la résolution de ce problème. Cette métaheuristique s'appuie sur une recherche de bonnes solutions (1) en coordonnant l'exploration des fourmis par rapport au nid de la colonie et (2) par des explorations locales de sites de chasse.

Cette métaheuristique est déjà efficace sur le problème, cependant, ses opérateurs d'exploration de l'espace peuvent être modifiés, sans remettre en cause les principes de celle-ci. Nous proposons donc une étude statistique de la distribution des efforts de la colonie avec ses opérateurs actuels. A partir de ces résultats, nous montrons que cette distribution peut être modulée et donc adaptée au problème à résoudre, grâce à l'utilisation d'un nouveau paramètre définissant la répartition des fourmis. Cette forme généralisée des opérateurs classiques permet de moduler les efforts d'une distribution concentrée près du nid à une distribution quasiment uniforme sur l'espace des solutions.

Après avoir constaté certaines insuffisances de la généralisation des opérateurs classiques, principalement dues au recouvrement des zones de chasse des fourmis, nous proposons de nouveaux opérateurs. Ces opérateurs définissent un partitionnement des zones de chasse des fourmis selon un principe rappelant la structure de la peau d'un oignon. Cette structure évite aux fourmis d'avoir des zones de chasse qui se recouvrent. L'introduction d'un paramètre similaire à celui évoqué précédemment permet de moduler les efforts de la colonie d'une distribution concentrée sur le nid à une distribution principalement éloignée du nid, tout en passant par la distribution uniforme sur l'espace des solutions.

Ces différents opérateurs sont expérimentés sur le problème de l'apprentissage de MMC. Pour cela, les contraintes de stochasticité des MMC sont naturellement incorporées au modèle, en munissant l'espace de recherche d'une structure vectorielle et d'une norme adaptées. Ces deux outils mathématiques permettent alors d'adapter naturellement et de façon comparable les différents opérateurs au problème. Les expérimentations mettent en évidence le grand intérêt de la capacité de modulation de la distribution des efforts de la colonie, pour améliorer les performances et réduire la variabilité des résultats. Elles montrent également que les nouveaux opérateurs proposés sont au moins aussi performants et flexibles que ceux obtenus par la généralisation des opérateurs d'origine de API sur le problème considéré. Finalement, ces études

statistiques et ces expérimentations permettent de progresser dans la compréhension du mode d'exploration de l'espace de recherche par API. Cela permet en particulier d'affiner le comportement des opérateurs dans cet espace.

Dans nos travaux futurs, nous envisageons d'exploiter d'autres structures de voisinage de MMC non basées sur un espace vectoriel et ainsi de comparer l'impact de celles-ci sur l'algorithme.

Ces travaux ouvrent la voie à un certain nombre d'améliorations de la métaheuristique API. Si les résultats expérimentaux se confirmaient sur d'autres types de problèmes d'optimisation, il serait possible de mettre en place des mécanismes adaptatifs permettant de déterminer ou d'ajuster le nouveau paramètre, introduit de façon automatique dans le problème considéré. De plus, nous envisageons une étude similaire de l'impact de la patience locale des fourmis et de son rôle exact au sein de l'algorithme API.

12.5. Bibliographie

[AGA 04] Agarwal S., Awan A., Roth D., « Learning to detect objects in images via a sparse, part-based representation », *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, n° 11, p. 1475–1490, 2004.

[AT 06] AT&T Laboratories, « The database of faces (formerly 'The ORL Database of Faces') », http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html, 2006.

[AUP 05] Aupetit S., Contributions aux modèles de Markov cachés : métaheuristiques d'apprentissage, nouveaux modèles et visualisation de dissimilarité, thèse de doctorat, Laboratoire d'Informatique de l'Université François-Rabelais de Tours, Tours, 30 novembre 2005.

[AUP 06] Aupetit S., Monmarché N., Slimane M., Liardet P., « An Exponential Representation in the API Algorithm for Hidden Markov Models Training », E.-G. Talbi, P. Liardet, P. Collet, E. Lutton, M. Schoenauer (dir.), *Artificial Evolution : 7th International Conference, Evolution Artificielle, EA 2005, Lille, October 26-28, 2005, Revised Selected Papers*, vol. 3871 de *Lecture Notes in Computer Science*, p. 61–72, Lille, Springer-Verlag, Berlin, Heidelberg, 2006.

[AUP 07] Aupetit S., Monmarché N., Slimane M., « Utilisation des modèles de Markov cachés pour la reconnaissance robuste d'images : apprentissage par colonie de fourmis, algorithme génétique et essaim particulaire », P. Siarry (dir.), *Optimisation en traitement du signal et de l'image*, Traitement du signal et de l'image, Traité IC2, p. 245–269, Hermès Science, Paris, février 2007.

[AUP 08] Aupetit S., Monmarché N., Slimane M., « Hidden Markov models training using population-based metaheuristics », P. Siarry, Z. Michalewicz (dir.), *Advances in metaheuristics for hard optimization*, Natural computing series, p. 415–438, Springer-Verlag, Berlin, Heidelberg, 2008.

[BAU 67] Baum L. E., Eagon J. A., « An inequality with applications to statistical estimation for probabilistic functions of Markov processes to a model for ecology », *Bull American Mathematical Society*, vol. 73, p. 360–363, 1967.

[BAU 72] Baum L. E., « An inequality and associated maximisation technique in statistical estimation for probabilistic functions of Markov processes », *Inequalities*, vol. 3, p. 1–8, 1972.

[BON 99] Bonabeau E., Dorigo M., Theraulaz G., *Swarm Intelligence : From Natural to Artificial Systems*, Oxford University Press, New York, 1999.

[CHE 04] Chen T., Mei X., Pan J., Sun S., « Optimization of HMM by the Tabu Search Algorithm. », *Journal Information Science and Engineering*, vol. 20, n° 5, p. 949–957, 2004.

[CLE 05] Clerc M., *L'optimisation par essaims particulaires : versions paramétriques et adaptatives*, Hermès Science, Paris, 2005.

[DOR 00] Dorigo M., Bonabeau E., Theraulaz G., « Ant algorithms and stygmergy », *Future Generation Computer Systems*, vol. 16, n° 8, p. 851–871, 2000.

[DRE 04] Dréo J., Siarry P., « Continuous interacting ant colony algorithm based on dense heterarchy », *Future Generation Computer Systems, Special issue : Computational chemistry and molecular dynamics*, vol. 20, n° 5, p. 841–856, juin 2004.

[FOR 73] Forney Jr. G., « The Viterbi algorithm », *Proceedings of IEEE*, vol. 61, p. 268–278, mars 1973.

[FRE 85] Fresneau D., « Individual foraging and path fidelity in a ponerine ant », *Insectes Sociaux, Paris*, vol. 32, n° 2, p. 109–116, 1985.

[FRE 94] Fresneau D., Biologie et comportement social d'une fourmi ponérine néotropicale (*Pachycondyla apicalis*), thèse d'état, Université de Paris XIII, Laboratoire d'Ethologie Expérimentale et Comparée, 1994.

[GAN 99] Ganapathiraju A., « Discriminative techniques in hidden Markov models », Course paper, 1999.

[GIU 02] Giurgiu M., « Maximization of mutual information for training hidden Markov models in speech recognition », *3rd COST #276 Workshop*, p. 96–101, Budapest, Hongrie, octobre 2002.

[HAM 96] Hamam Y., Al Ani T., « Simulated annealing approach for Hidden Markov Models », *4th WG-7.6 Working Conference on Optimization-Based Computer-Aided Modeling and Design, ESIEE, France*, mai 1996.

[JUA 90] Juang B., Rabiner L., « The segmental k-means algorithm for estimating parameters of hidden Markov models », *IEEE transactions on acoustics, speech and signal processing*, vol. 38, n° 9, p. 1639–1641, septembre 1990.

[KAP 98] Kapadia S., Discriminative training of hidden Markov models, PhD thesis, Downing College, University of Cambridge, 18 mars 1998.

[KEN 95] Kennedy J., Eberhart R., « Particle swarm optimization », *Proceedings of the IEEE international joint conference on neural networks*, vol. 4, p. 1942–1948, IEEE, 1995.

[KWO 97] Kwong S., Chau C., « Analysis of Parallel Genetic Algorithms on HMM Based Speech Recognition System », *Proceedings of ICASSP'97*, p. 1229–1233, 1997.

[LJO 90] Ljolje A., Ephraim Y., Rabiner L., « Estimation of hidden Markov model parameters by minimizing empirical error rate », *IEEE International Confonference on Acoustic, Speech, Signal Processing*, p. 709–712, Albuquerque, Etat-Unis, avril 1990.

[MAR 13] Markov A. A., « An example of statistical investigation in the text of "Eugene onyegin" illustrating coupling of "tests" in chains », *Proceedings of Academic Scientific*, VI, p. 153–162, St. Petersbourg, Russie, 1913.

[MAX 99] Maxwell B., Anderson S., « Training Hidden Markov Models using Population-Based Learning », W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, R. E. Smith (dir.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, vol. 1, page944, Orlando, Floride, Etat-Unis, Morgan Kaufmann, 1999.

[MON 00a] Monmarché N., Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation, thèse de doctorat, Laboratoire d'Informatique de l'Université François-Rabelais de Tours, 20 décembre 2000.

[MON 00b] Monmarché N., Venturini G., Slimane M., « On how *Pachycondyla apicalis* ants suggest a new search algorithm », *Future Generation Computer Systems*, vol. 16, n° 8, p. 937–946, 2000.

[MON 02] Monmarché N., « Algorithmes à base de fourmis artificielles pour les problèmes à variables continues », *Actes des quatrièmes journées nationales de la ROADEF*, p. 224–225, Paris, 20-22 février 2002.

[PAU 85] Paul D. B., « Training of HMM recognizers by simulated annealing », *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, p. 13–16, 1985.

[RAB 89] Rabiner L. R., « A tutorial on hidden Markov models and selected applications in speech recognition », *Proceedings of the IEEE*, vol. 77, n° 2, p. 257–286, février 1989.

[RAS 03] Rasmussen T. K., Krink T., « Improved hidden Markov model training for multiple sequence alignment by a particle swarm optimization - evolutionary algorithm hybrid », *BioSystems*, vol. 72, p. 5–17, 2003.

[SAU 00] Saul L., Rahim M., « Maximum likelihood and minimum classification error factor analysis for automatic speech recognition », *IEEE Transactions on Speech and Audio Processing*, vol. 8, n° 2, p. 115–125, 2000.

[SCH 97] Schluter R., Macherey W., Kanthak S., Ney H., Welling L., « Comparison of optimization methods for discriminative training criteria », *EUROSPEECH '97, 5th European Conference on Speech Communication and Technology*, p. 15–18, Rhodes, Grèce, septembre 1997.

[SLI 99] Slimane M., Brouard T., Venturini G., Asselin de Beauville J.-P., « Apprentissage non-supervisé d'images par hybridation génétique d'une chaîne de Markov cachée », *Traitement du signal*, vol. 16, n° 6, p. 461–475, 1999.

[SLI 02] Slimane M., Chaînes de Markov cachées, algorithmes évolutionnaires et fourmis artificielles : contributions et hybridations, Habilitation à diriger des recherches, Université

François-Rabelais de Tours, décembre 2002.

[SOC 08] Socha K., Dorigo M., « Ant colony optimization for continuous domains », *European Journal of Operational Research*, vol. 185, n° 3, p. 1155–1173, 2008.

[THO 02] Thomsen R., « Evolving the topology of hidden Markov models using evolutionary algorithms », *Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002)*, p. 861–870, 2002.

[VER 04] Vertanen K., An overview of discriminative training for speech recognition, Rapport, University of Cambridge, 2004.

[VIT 67] Viterbi A. J., « Error bounds for convolutional codes and asymptotically optimum decoding algorithm », *IEEE transactions on information theory*, vol. 13, p. 260–269, 1967.

Index des noms propres

Index général

L'informatique, omniprésente dans notre vie, est multiforme. A la fois profondément unitaire quant à ses principes d'écriture et ceux qui sont à la base des machines, l'informatique est infiniment variée par ses applications.

INFORMATIQUE ET SYSTÈMES D'INFORMATION couvre l'ensemble des domaines suivants :

- Apprentissage
- Arithmétique des ordinateurs
- Bases de données
- Bioinformatique
- Représentation des connaissances
- Informatique parallèle et répartie
- Logique et programmation
- Recherche d'information et web
- Recherche opérationnelle

Chaque ouvrage décrit aussi bien les aspects fondamentaux qu'expérimentaux. Une classification des différents chapitres contenus dans chacun, une bibliographie et un index détaillé orientent le lecteur vers ses points d'intérêt immédiats : celui-ci dispose ainsi d'un guide pour ses réflexions et ses choix.

Sur chaque aspect, le traité s'efforce de marier chapitres de synthèse et connaissances les plus récentes pour donner au lecteur le panorama complet d'un sujet.

Hermes Science publications

www.hermes-science.com

978-2-7462-2119-2