La mémoire dans les algorithmes à colonie de fourmis : applications à l'optimisation combinatoire et à la programmation automatique

Olivier Roux

▶ **To cite this version:**

Olivier Roux. La mémoire dans les algorithmes à colonie de fourmis : applications à l'optimisation combinatoire et à la programmation automatique. Autre [cs.OH]. Université du Littoral Côte d'Opale, 2001. Français. NNT : . tel-00008597

HAL Id: tel-00008597

https://theses.hal.science/tel-00008597

Submitted on 28 Feb 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université du Littoral Côte d'Opale

THÈSE

pour obtenir le grade de

Docteur de l'Université du Littoral Côte d'Opale

discipline : informatique

présentée et soutenue publiquement

par

Olivier Roux

le 13 décembre 2001

La mémoire dans les algorithmes à colonie de fourmis : applications à l'optimisation et à la programmation automatique

---

Jury :

M.	Henri BASSON	Président
MM.	Jin-Kao HAO	Rapporteurs
	El-ghazali TALBI	
MM.	Pierre COLLET	Examinateurs
	Jean LOUCHET	
M.	Philippe PREUX	Directeur
MM.	Cyril FONLUPT	Co-directeurs
	Denis ROBILLIARD	

Si nous prenons la nature pour guide,
nous ne nous égarerons jamais.
[Cicéron]

Remerciements

MERCI !

C'est un petit mot tout simple
Mais qui pèse lourd
Si mes lèvres l'expriment avec douceur
C'est qu'il prend naissance au fond de mon cœur

Un grand Merci, un petit Merci
Peu importe sa taille
Il n'a pas de dimension ...
Que ce soit dans la joie ou dans la tristesse !
C'est un signe de reconnaissance
Qui ne connaît pas l'indifférence !
Merci !

Un petit mot qui fait du bien quand on le prononce
Un petit mot gracieux qui calme et réjouit
Merci ! Merci !
Merci de m'avoir permis de te dire,
De vous dire :
Merci !

[Auteur anonyme]

Voici quelques mercis. Malheureusement je crains d'oublier de citer certaines personnes ; j'espère quelles ne m'en tiendront pas grief, et je peux leur assurer qu'elles ont une place particulière dans mon cœur.

Merci tout particulièrement :

- À mes proches pour leur soutien.
- À Christian Alvès sans lequel je n'aurais jamais pu envisager cette thèse.
- À Philippe Preux d'avoir accepté d'être mon directeur de thèse et pour m'avoir accueilli au sein de son laboratoire.
- À mes encadrants pour cette thèse : Cyril Fonlupt et Denis Robilliard pour leurs marques d'amitiés, leurs conseils avisés et leur aide tout au long de ces trois années.
- Aux rapporteurs d'avoir pris sur leur temps pour lire ce mémoire, et aux membres du jury de l'intérêt dont ils font preuve à mon égard en assistant à ma soutenance.
- À Pierre Pérucaud et à ma maman sans qui ce rapport aurait ressemblé à un écrit du XIVème siècle (où la phonétique était de mise).
- À Nicolas Monmarché pour les connaissances en dont il m'a fait profiter.
- Aux membres du LIL pour leurs témoignages d'amitiés et leurs encouragements : plus particulièrement à Virginie (et Philou), Jean-Marc, Eric, Bruno, François, Claire, Mourad, Laurent, Raphaël, Grégory (bis), Nordine, Samuel, ...

– À tous ceux dont j'ai fait la connaissance dans le cadre des mes enseignements pour leurs conseils et leur aide : Karine Silini, David Duvivier, Patrick Martin, ...
– À Dominique Verhaghe sans qui ma petite linuxette n'aurait pas survécu aux mauvais traitements que je lui ai fait subir tout au long de ces années.
– À tous les enseignants qui au long de mon cursus ont su éveiller et conforter mon intérêt pour linformatique, en particulier à M. Durannel pour m'avoir fait découvrir le TRS-80 et à Cina Motamed pour avoir su me convaincre de reprendre mon parcours universitaire, ...
– Enfin merci aux personnes qui liront ces quelques pages, reflet de trois années de recherche, où les fourmis y sont dotées de capacités hors du commun et se meuvent dans un univers cybernétique rempli de valeurs.

Table des matières

1 Introduction **13**

I Les fourmis, l'informatique et la mémoire **17**

2 Les fourmis, insectes de communication **19**
2.1 Introduction 19
2.2 Un peu d'anatomie 20
2.3 La communication 20
2.3.1 La communication sonore 21
2.3.2 La communication tactile. 21
2.3.3 La communication visuelle 21
2.3.4 La communication chimique 22
2.4 Entomologie de la fourmi 22
2.4.1 L'habitat des fourmis 23
2.4.2 La vie sociale des fourmis 23
2.5 La fourmi insecte eusocial. 23
2.5.1 Définition de l'eusociabilité 24
2.5.2 Qu'est ce que la parentèle ? 24
2.5.3 Autres sociétés eusociales 25
2.5.4 L'eusociabilité, une classification amendée. 25
2.6 Quelques exemples de résolution de problèmes complexes 25
2.6.1 La formation d'un nid végétal 25
2.6.2 La recherche de nourriture ou fourragement 26
2.6.3 Stigmergie : définition 27
2.7 Conclusion 28

3 Les fourmis artificielles **29**
3.1 La fourmi numérique 29
3.2 La fourmi virtuelle vue comme un agent 29
3.3 Le modèle de coopération 32
3.4 Les différents domaines d'application 33
3.5 Les fourmis artificielles pour la recherche opérationnelle 33
3.5.1 Le voyageur de commerce 34
3.5.2 Problème d'ordonnancement séquentiel 42
3.5.3 Détection de graphe hamiltonien 44
3.5.4 Le problème d'affectation quadratique 45

3.5.5 Le problème de routage dans un réseau non commuté 45
3.5.6 Le problème d'optimisation numérique 47
3.5.7 Un aperçu des différents domaines d'application des OCF 50
3.6 La fourmi artificielle et la programmation automatique 50
3.7 Conclusion 50

4 Les fourmis face aux autres méthodes 53
4.1 Introduction 53
4.2 Les autres méthodes "naturelles" disponibles 53
4.2.1 Les algorithmes génétiques 54
4.2.2 Les autres méthodes guidées 59
4.3 Comment les fourmis utilisent-elles leur mémoire ? 65
4.3.1 Emergence d'un modèle fédérateur : AMP 65
4.3.2 Approfondissement du modèle et adjonction de nouveaux critères 66
4.3.3 Quelques heuristiques décrites sous la forme étendue des AMP . 67
4.3.4 Les différents types d'utilisation de la mémoire 69
4.3.5 La difficulté d'une comparaison 71
4.3.6 La difficulté d'une classification 71
4.4 Conclusion 71

II Les fourmis pour le PAQ 73

5 Le problème d'affectation quadratique 75
5.1 Principe 75
5.2 La version simplifiée 76
5.2.1 L'exemple de l'implantation d'usines 77
5.2.2 Un exemple pédagogique 77
5.2.3 Instances du PAQ 78
5.3 Méthode de résolution pour le PAQ 79
5.3.1 Les heuristiques pour le PAQ 79
5.3.2 Les OCF pour le PAQ 79
5.3.3 *Ant-System* pour le PAQ 81
5.3.4 L'algorithme ANTS pour le PAQ 83
5.3.5 $\mathcal{MM}$AS pour le PAQ 83
5.3.6 L'algorithme FANT 84
5.3.7 HAS-QAP 85
5.4 Notre méthode hybride : ANTabu 85
5.4.1 Les éléments clés de ANTabu 86
5.4.2 Exemple de déroulement pour ANTabu 89
5.4.3 Implantation parallèle 90
5.5 Conclusion 93

6 Etude expérimentale d'ANTabu 95
6.1 Introduction 95
6.2 Validation des choix pour ANTabu 96
6.2.1 ANTabu pour de courtes exécutions 96

6.2.2 Validation du paramétrage du système de fourmis 96
6.3 Coopération contre force brute 101
6.4 Comparaisons avec ANTabu 102
6.4.1 Comparaison avec HAS-QAP 102
6.4.2 ANTabu face aux autres méthodes 102
6.5 Conclusion 103

III Les fourmis et la programmation automatique 107

7 La programmation automatique 109
7.1 Introduction 109
7.2 La programmation automatique 110
7.3 La programmation génétique 111
7.4 Améliorations actuelles de la PG 114
7.5 L'algorithme de programmation par fourmis (AP) 115
7.5.1 Le détail de notre algorithme 116
7.5.2 Exemple de déroulement pour AP 119
7.6 Conclusion 122

8 Résultats d'Ant Programming 125
8.1 Les problèmes 125
8.2 Paramétrage 128
8.3 Comparaison avec la PG 128
8.4 Extensions 130
8.4.1 La génération partielle des individus 130
8.4.2 La gestion des constantes 132
8.5 Conclusion 133

IV Les fourmis et après ? 135

9 Conclusion 137

Liste des tableaux

3.1 Equations relatives aux mouvements des fourmis pour les différents algorithmes 36
3.2 Equations relatives à la mise à jour de la phéromone pour les différents algorithmes 37
3.3 Comparaison de l'AS (Ant-cycle) avec d'autres algorithmes sur l'instance de PVC Oliver30 38
3.4 Comparaison de Ant-Q avec d'autre méthodes classiques pour cinq instance de 50 villes. Les valeurs présentées sont les erreurs en pourcentage 39
3.5 comparaison de AntQ avec des méthode exactes pour des instance de PVC asymétrique 40
3.6 Comparaison entre AS, ACS et MMAS pour ry48p 40
3.7 Comparaison d'AS$_{rank}$ avec AS et une autre variante proposée dans le même article, on trouve également les résultats obtenus avec des méthodes classiques. Les résultats sont présentés pour des instances de tailles différentes. 41
3.8 Comparaison sur des instances symétriques et asymétriques de l'ACS-3opt avec un algorithme génétique (ATSP), les résultats étant obtenus pour dix exécutions [64]. 42
3.9 Algorithmes utilisant le modèle des OCF pour différents problèmes d'optimisation combinatoire 51

4.1 Utilisation qui est faite de la mémoire pour différentes heuristiques 70

5.1 Méthode de recherche locale utilisée par les OCF 81
5.2 Caractéristiques des différents OCF qui ont été appliqués au PAQ. 82

6.1 Résultats de l'ANTabu sur de petites instances et de courtes exécutions (10 itérations) 97
6.2 Comparaison entre une diversification aléatoire et notre diversification . 98
6.3 Performance de ANTabu sur des instances irrégulières 99
6.4 Comparaison des deux versions de l'application comportant chacune une des deux modifications avec une version qui en est dépourvue. Seules sont présentées les statistiques trouvées lors de la comparaison 99
6.5 Comparaison des résultats du PATS et de l'ANTabu. Les meilleurs résultats sont en **gras.** 101
6.6 Comparaison des résultats du HAS-QAP et de l'ANTabu. Les meilleurs résultats sont en **gras.** 103
6.7 Comparaison des résultats pour des instances régulières avec une durée de calcul identique 104

6.8 Comparaison des résultats pour des instances irrégulières avec une durée de calcul identique. *Les meilleurs résultats sont en caractères gras. Les valeurs représentent les écarts moyens avec la meilleure solution connue, obtenus pour 10 exécutions et sont exprimés en pourcentage.* 105

7.1 Tableau donnant pour quelques problèmes classiques [126] la fonction de calcul de la qualité d'un programme 111
7.2 Durée d'une révolution P (en années terrestres) en fonction de la distance A de la planète au soleil (la distance est exprimée en fonction de la distance terre/soleil) 112
7.3 Valeurs d'apprentissage de l'exemple 119

8.1 Ensemble des fonctions et des terminaux 126
8.2 Paramétrage de l'algorithme de la PG 129
8.3 Résultats : comparaison de AP avec la PG 129
8.4 Comparaison de la profondeur des solutions générées par AP et la PG 131

Table des figures

2.1 Schéma anatomique d'une fourmi 20

2.2 Dépôt de la phéromone par une fourmi : la phéromone est déposée sur le sol par la fourmi au moyen de son aiguillon 22

2.3 La parentèle : Les valeurs sur les flèches indiquent le facteur de similitude génétique entre les différents membres d'une colonie de fourmis. R désigne la reine, O une travailleuse, F une future reine, M un mâle fils de la reine et Ro le mâle qui s'est accouplé avec la reine 24

2.4 Coopération de fourmis tisserandes pour la construction de leurs nids en feuilles : pour la construction, il est nécessaire de rapprocher les feuilles ; si leur écartement est plus faible que la taille d'une fourmi, elles attrapent les deux extrémités et les rapprochent (première image de a et c) ; si elles sont plus éloignées elles construisent des ponts (b). Les extrémités sont ensuite collées avec la soie des larves (seconde image de a et d) 26

2.5 L'expérience du double pont : a) l'environnement de l'expérience, b) les graphes donnent la distribution des pourcentages de sélection de la branche la plus courte pour un jeu d'expériences. La branche du haut est r fois plus longue que l'autre. Le graphe de gauche (14 expériences) donne la répartition lorsque les deux branches sont présentées simultanément. Pour le graphe de droite (14 expériences), la branche la plus courte n'est présentée que 30 minutes après l'autre : la branche la plus longue est toujours exploitée à cause du dépôt de phéromone initial. . . 27

3.1 Comportement d'une fourmi naturelle lors de la recherche de nourriture (dans le cas des fourmis artificielles, il n'y a pas obligatoirement de phase de recrutement.) 30

3.2 Exemple d'une instance créée de 92 villes, accompagnée de sa solution 34

3.3 La méthode d'optimisation locale 3-opt consiste en l'échange de trois arcs : on choisit d'abord un premier arc, puis un second de telle façon que si l'on échange les extrémités de ces arcs la longueur du tour soit diminuée : dans ce cas on réitère cette opération en choisissant un troisième arc de la même manière . . . 43

3.4 Résultats obtenus pour de petites instances avec HAS-SOP face à ceux d'autres algorithmes. Les graphiques représentent le pourcentage d'erreur par rapport à la meilleure solution connue en fonction du temps d'exécution. Les résultats sont les moyennes obtenues sur 5 exécutions de 120 secondes 43

3.5 Résultats obtenus pour de grandes instances avec HAS-SOP face à ceux d'autres algorithmes. Les graphiques représentent le pourcentage d'erreur par rapport à la meilleure solution connue en fonction du temps d'exécution. Les résultats sont les moyennes obtenues sur 5 exécutions de 600 secondes. 44

3.6 Schéma d'un réseau américain 45

3.7 Exemple d'un problème de routage : Quand une fourmi arrive sur un nœud du graphe, elle va choisir le nœud suivant en se basant sur les valeurs placées dans la table de routage. Ici elle vient du nœud 5 et elle se dirige ves le nœud 2, c'est donc les valeurs de la colonne 2 qui seront utilisées et c'est le nœud 3 qui aura le plus de chance d'être choisi car il a la plus forte valeur. 47
3.8 Résultats obtenus avec AntNet : Pour la comparaison, on simule la saturation d'un réseau pendant 120 secondes, 400 secondes après le début du test. Le graphique du haut donne le débit (le plus large est le meilleur) tandis que le graphique du bas donne le retard moyen dépassant les 5 secondes (le plus bas est le meilleur). AntNet est compétitif avec les meilleurs algorithmes pour le débit et obtient les meilleures performances pour le retard moyen (seul deamon est meilleur, mais il s'agit d'un modèle théorique) 48
3.9 Le déplacement d'une fourmi peut être mis sous la forme d'un vecteur. 48
3.10 Les départs du nid : Le nombre de vecteurs (déplacement des fourmis) partant du nid doit être fixé 49

4.1 Schéma général de fonctionnement d'un AG 56
4.2 La recombinaison un point : Les deux chromosomes vont être coupés au même endroit, les éléments ainsi formés vont être échangés pour produire de nouveaux individus 57
4.3 La recombinaison deux points : Les deux chromosomes sont coupés en trois segments, le segment du milieu est alors échangé pour obtenir les nouveaux individus. 57
4.4 La mutation : un géne du chromosome est sélectionné, il est alors remplacé par un nouveau choisi aléatoirement 58
4.5 L'influence de la température pour le recuit : dans le premier schéma la température est élevée (large domaine de recherche), ce qui permet à l'algorithme de facilement sortir d'une vallée contenant un optimum local pour se diriger vers de meilleurs sites. Dans le second cas, la température est plus faible (espace de recherche plus restreint) ce qui bloque la méthode au niveau de cet optimum local. 61
4.6 La recherche tabou : lors du passage de voisin en voisin les retours en arrière et les boucles sont interdits par l'utilisation de la liste tabou. 62

5.1 Schéma représentant l'implantation d'usines 77
5.2 Schéma représentant un exemple de PAQ 78
5.3 Le choix de la prochaine affectation de la fourmi : arrivée en position 1, elle choisit l'objet à affecter, puis suivant les valeurs de phéromone (τ) et heuristiques (η), elle fait son choix entre les trois positions disponibles $(2, 3, 4)$ 80
5.4 Exemple d'évolution de la qualité avec une méthode pour le PAQ adoptant une stratégie de modification des solutions. π^k indique la solution de la fourmi k, π^k_m est la solution modifiée et π^k_o celle obtenue après la recherche locale. 87
5.5 Echange d'informations pour l'ANTabu. 92
5.6 Modèle parallèle de ANTabu. 93

6.1 Illustration de la notion de structure d'une instance par forme de son paysage . 96
6.2 Corrélations entre les performances de ANTabu et la taille, le flot de dominance et la distance dominance des instances 100

7.1 Schéma d'évolution de la programmation génétique 110
7.2 Arbre généré par PG à l'initialisation correspondant à $A\frac{\sqrt{A}}{\sqrt{A}}$ 112
7.3 Recombinaison de deux programmes parents et génération de deux nouveaux programmes fils. Ici la recombinaison se réalise par l'échange d'un sous-arbre entre les parents, le point de coupure est indiqué par un éclair noir 113
7.4 Mutation d'un programme, il s'agit de remplacer un sous-arbre par un nouvel arbre, généré aléatoirement 113
7.5 Représentation de trois solutions obtenues avec PG et des valeurs du jeu d'apprentissage 114
7.6 Effet de la recombinaison 115
7.7 Représentation de la programmation génétique fortement typée 116
7.8 Arbre de phéromone : Dans chaque nœud est placée la quantité de phéromone associée à chaque fonction ou terminal possible. 116
7.9 Sélection d'un nœud lors de la génération d'un programme par AP. Cet élément est choisi grâce à une méthode de roulette, c'est-à-dire que la sélection se fait avec une probabilité proportionnelle au taux de phéromone associé aux différents composants possibles correspondant à la position de ce nœud. 117
7.10 L'arbre de phéromone après initialisation 120
7.11 L'arbre de phéromone après initialisation 120
7.12 L'arbre de phéromone après la première itération 121
7.13 L'arbre de phéromone après n itérations : la formule du volume y apparaît (cercle), on peut également trouver une autre représentation de cette formule (souligné) . 122

8.1 Multiplexeur 11 bits avec comme entrée 11001000000 et comme sortie 1 125
8.2 Problème de régression : $f(x) = x^4 + x^3 + x^2 + x$ 126
8.3 Problème de régression : $f(x) = x^3 \cdot e^{-x} \cdot cos(x) \cdot sin(x) \cdot (sin^2(x) \cdot cos(x) - 1)$ 127
8.4 Representation du problème des deux spirales entrelacées 127
8.5 Representation du "Santa Fe Trail Problem" 128
8.6 $f(x) = x^3 \cdot e^{-x} \cdot cos(x) \cdot sin(x) \cdot (sin^2(x) \cdot cos(x) - 1)$ ainsi que la meilleure solution trouvée par AP 130
8.7 Génération d'un sous-arbre par utilisation de la phéromone 131
8.8 Choix des constantes. On peut choisir de générer une nouvelle constante (la case ?) ou sélectionner avec une certaine probabilité (proportionnelle à sa qualité) une des quatre déjà utilisée 132

Liste des algorithmes

3.1 L'algorithme général des OCF. 34
3.2 L'algorithme AS-TSP. 38
3.3 L'algorithme VAW. 44
3.4 L'algorithme d'optimisation numérique. 49
4.1 Algorithme simplifié d'un Algorithme génétique. 58
4.2 Algorithme simplifié du recuit simulé. 60
4.3 Algorithme tabou. 63
4.4 L'apprentissage progressif basé sur la population. 64
4.5 Forme générale d'un AMP. 66
5.1 Algorithme simplifié de l'ANTabu. 86
5.2 La forme la plus simple de la recherche tabou. 87
7.1 Algorithme AP. 117

Chapitre 1

Introduction

Le travail des chercheurs a souvent été comparé celui d'une fourmi, laborieuse, minutieuse, qui œuvre pour le bien du groupe. Le rapport entre cet insecte et ma thèse ne s'arrête pas là. Durant ces quelques années de thèse, j'ai travaillé sur leurs consœurs virtuelles, travailleuses acharnées, ayant des capacités surprenantes pour la résolution de problèmes, comme nous le verrons dans ce mémoire.

M. Dorigo a été parmi les premiers à utiliser les fourmis en informatique. Comme le laisse entendre la petite histoire, l'idée lui en serait venue après avoir assisté à un séminaire... C'est de l'émerveillement face à ces insectes dotés de capacités collectives impressionnantes qu'il décida de simuler leurs homologues cybernétiques afin de les tester dans un environnement numérique où leur coopération permettrait de résoudre des problèmes complexes.

Ces insectes réussissent en effet à effectuer naturellement des tâches très complexes à travers une coopération inconsciente. Ce mécanisme décrit par Grassé [105] est connu sous le terme de stigmergie. Elle permet aux fourmis, grâce à une communication indirecte utilisant un marqueur chimique appelé phéromone, de collaborer à la résolution de problèmes tels que la construction de nids, l'élevage des larves ou encore la récolte de nourriture.

Un nouveau domaine d'étude est apparu, issu de la collaboration des laboratoires d'éthologie et des informaticiens dans leur tentative d'appliquer le modèle de coopération des fourmis à la résolution de problèmes complexes. Ce nouveau domaine étudie l'intelligence en essaim et les algorithmes à essaim. Les agents de ces systèmes, en général incapables de tout apprentissage individuel, réussissent à résoudre des problèmes ardus par l'intermédiaire d'un système mémoriel collectif.

Pour certains, dont Arkin [2], ces études s'intègrent dans ce que l'on appelle la biomimétique, c'est-à-dire dans les sciences qui s'inspirent du vivant. Elles reposent sur la constatation que les animaux et les insectes n'utilisent pas de connaissances complexes, de matériaux exotiques ou d'énormes quantités d'énergie au niveau individuel pour résoudre les problèmes auxquels ils sont confrontés. Par des comportements simples et des interactions limitées, on assiste à l'émergence de comportements et de modèles d'auto-organisation.

Que l'on parle d'intelligence à essaim ou de mécanismes biomimétiques, le modèle des fourmis est riche en applications potentielles. Les modes de coopération inspirés de ces comportements en essaim ont été appliqués avec succès à de nombreux domaines d'optimisation combinatoire comme le voyageur de commerce, le coloriage de

graphe, ... ou encore des problèmes dynamiques comme ceux du routage dans des réseaux de communication. La principale application est donc celle des méthodes d'optimisation à base de colonie de fourmis que l'on nommera OCF par la suite dans ce mémoire. Comme le promet la biomimétique, les principes mis en œuvre sont simples et les solutions émergent de la collaboration de l'essaim de fourmis.

Dans la première partie, nous observerons la transition entre la fourmi naturelle et sa consœur artificielle, cette dernière étant particulièrement décrite pour les OCF. Ce modèle sera placé parmi les heuristiques existantes.

Le chapitre 2 du mémoire abordera la présentation de la fourmi naturelle. Il présentera son caractère eusocial (présence de pouponnières, côtoiement entre les générations et spécialisation des individus). Cette approche nous permettra de voir quelles sont les spécificités de la fourmi naturelle et introduira son double numérique présenté dans le chapitre suivant.

Le chapitre 3 sera l'occasion de présenter succinctement les différents domaines d'application des OCF et les différentes variantes qui ont été proposées. En particulier, nous nous intéresserons aux problèmes classiques du voyageur de commerce, d'ordonnancement, d'optimisation numérique, de routage de réseaux, d'affectation quadratique, ...

Les OCF sont-ils si différents des autres méthodes d'optimisation combinatoire ou ne sont-ils qu'une évolution vers un type de méthode plus robuste ? Nous avons choisi comme approche de comparaison le concept d'utilisation de la mémoire. Cette approche nous donne l'occasion de proposer une taxinomie qui permet de classer les OCF parmi les méthodes existantes. Cette taxinomie sera étudiée et discutée au chapitre 4.

Il arrive fréquemment que conjointement aux OCF soit introduite une méthode de recherche locale pour améliorer les performances du modèle. On peut s'interroger sur l'influence de la recherche locale par rapport à la coopération des fourmis. C'est dans ce cadre que nous avons proposé ANTabu, une hybridation entre un OCF et une méthode de recherche locale robuste (la recherche tabou) pour le problème d'affectation quadratique, ce problème étant connu comme très complexe à résoudre. Cette étude fera l'objet de la seconde partie de ce mémoire. Dans le chapitre 5, nous décrirons la formalisation du PAQ et les OCF qui lui ont été appliquées. Nous examinerons de manière fine les influences respectives du mécanisme de collaboration des fourmis et de la recherche locale dans le chapitre 6, où nous comparerons également les résultats de ANTabu avec d'autres heuristiques.

La troisième partie de cette thèse présente une nouvelle application du modèle de coopération des fourmis à la programmation automatique. Il s'agit ici de voir si la coopération entre les fourmis peut faire émerger de bonnes solutions pour la génération automatique de programmes. En nous basant sur la mémoire collective représentée par la phéromone, nous proposons une méthode de programmation automatique nommée Ant Programming (AP). La fourmi n'est guidée, au cours de la construction de sa solution, que par l'information laissée par ses consœurs sous la forme de "phéromone", la stigmergie faisant émerger la forme des bonnes solutions par les apports successifs des fourmis. Notre algorithme est présenté et détaillé, après une rapide description de la programmation automatique et plus précisément de la programmation génétique dans le chapitre 7.

Pour valider cette approche, nous avons confronté les résultats obtenus avec ceux de la programmation génétique, méthode de programmation automatique la plus utilisée

à l'heure actuelle. Ces comparaisons sont présentées dans le chapitre 8.

C'est tout naturellement que ce mémoire se terminera par le bilan des résultats obtenus et par les perspectives qui découlent de ce travail de thèse.

Première partie

Les fourmis, l'informatique et la mémoire

Chapitre 2

Les fourmis, insectes de communication

Dans ce chapitre, nous allons aborder succinctement quelques aspects du comportement propre aux fourmis biologiques. Un bref survol de leur anatomie et de leur mode de vie nous conduira à nous attarder sur leurs moyens de communication, leur sociabilité qui est des plus évoluée ainsi que sur leurs capacités collectives. L'étude de ces insectes a permis une adaptation de leur comportement à l'informatique en vue de la création d'un modèle de fourmi artificielle permettant son utilisation pour l'optimisation combinatoire. C'est ce modèle que nous traiterons dans le chapitre suivant.

2.1 Introduction

Le terme "fourmi" est la pierre angulaire de cette thèse. Derrière ce mot se profilent plusieurs domaines : celui de la biologie ou plus précisément de la myrmécologie qui est l'étude du comportement naturel des fourmis, celui de la robotique qui utilise leur comportement pour concevoir des nouvelles machines, et celui de l'informatique où ces créatures sont modélisées pour la simulation ou la création d'algorithmes. C'est cette dernière application qui nous intéresse dans ce mémoire. Mais comment aborder ce sujet sans nous attarder sur le modèle naturel, donc sur ces fourmis ? Dans ce chapitre, nous allons étudier quelques-unes des caractéristiques des fourmis, celles qui nous semblent les plus pertinentes pour la compréhension du reste de ce document (à aucun moment les descriptions ne se veulent exhaustives). Les informations qui sont données proviennent principalement de deux ouvrages qui sont *Voyage chez les fourmis* de Hölldobler et *Swarm intelligence from natural to artificial systems* de Bonabeau, Dorigo et Theraulaz. Nous allons définir rapidement l'anatomie de la fourmi et la situer dans l'écosystème.

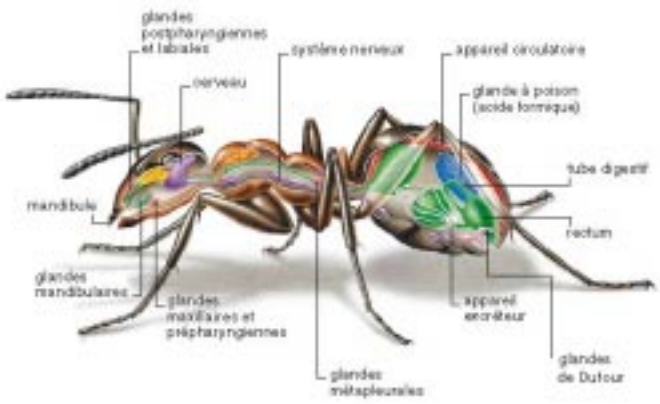


FIG. 2.1 – Schéma anatomique d'une fourmi

2.2 Un peu d'anatomie

La taille des fourmis (1mm à 3cm) varie selon l'espèce. On estime que 9500 espèces sont connues à l'heure actuelle, et que le nombre total est le double. Leur anatomie (Fig. 2.1) se décompose en trois segments (tête, thorax et abdomen), bien différenciés, qui sont unis entre eux par d'étroits pédoncules. Leur tête porte deux antennes formées de plusieurs segments articulés. Leurs pattes, au nombre de six, sont attachées à leur thorax. La possession des ailes est liée à la division en castes ; seuls les mâles et les femelles fécondes, ou reines, en sont munis. Il s'agit d'un reliquat hérité d'un ancêtre commun avec les abeilles. De nombreuses espèces sont dotées d'un aiguillon plus ou moins développé à l'extrémité de leur abdomen. On peut observer que cet aiguillon est alimenté par deux glandes :

- une glande acide, venimeuse, dorsale qui débouche dans un réservoir situé à la base de l'aiguillon ;
- une glande alcaline appelée glande de Dufour, plus discrète, ventrale, qui débouche à la base de l'appareil vulnérant.

Le composé chimique volatil secrété par cette glande, lorsqu'il est déposé sur le sol par un individu, forme une trace odorante qui a pour but d'inciter ses congénères à emprunter le même chemin. Lorsqu'une fourmi se trouve face à une source de nourriture importante, elle en découpe une partie qu'elle emporte, puis sécrète une trace odorante qu'elle dépose sur le chemin menant à la fourmilière. Le composé déposé sur le sol a pour but d'inciter les autres individus à se diriger vers la source de nourriture. Cette substance, qui est un outil de communication chez les fourmis, est appelée phéromone. Ce mécanisme leur permet de résoudre des tâches complexes comme nous le verrons par la suite.

2.3 La communication

Les mécanismes de communication sont très élaborés chez les fourmis. Ils sont sonores, tactiles, visuels ou chimiques. Si l'on considère leurs organes des sens, ils sont plus

ou moins développés. Ainsi, si dans certaines espèces la vue est faible et même parfois nulle, le toucher, l'odorat et le goût sont très développés. Ils permettent, entre autres propriétés, aux individus d'une même fourmilière de se reconnaître mutuellement.

2.3.1 La communication sonore

Elle est obtenue par les frottements d'un mince grattoir transversal situé sous la taille, contre un plateau de fines crêtes parallèles, placé sur l'abdomen.

Elle est utilisée selon les circonstances et en fonction de l'espèce :

- comme signal de détresse : émis par l'ouvrière en danger, il se propage par le sol et il est reçu par les pattes de l'insecte, véritables détecteurs ultra-sensibles aux vibrations du sol ;
- comme signal de qualité de l'alimentation : les fourmis récolteuses, difficiles quant aux choix des végétaux qu'elles sélectionnent, émettent une vibration d'intensité variable, selon la valeur nutritive de la nourriture repérée ;
- comme signal de renforcement : l'aphenogaster ou fourmi du désert crisse pour obtenir l'aide des autres individus lorsqu'elle a trouvé un aliment de grande taille ;
- comme signal de danger pour la fourmilière : certaines espèces se frappent la tête sur une matière dure, permettant ainsi la propagation d'un message d'alerte. D'autres espèces se servent de leur abdomen de la même façon.

2.3.2 La communication tactile.

Chez les fourmis, certains messages simples sont transmis par des contacts physiques tels que les attouchements, les tapotements ou les effleurements. Ce type de communications est réalisé en grande majorité par l'intermédiaire des antennes. Par exemple, la fourmi tisserande, qui utilise des larves productrices de soie pour assembler des feuilles entre elles, fait vibrer l'extrémité de ses antennes une dizaine de fois autour de la tête de la larve afin qu'elle déclenche cette sécrétion. Néanmoins, la communication par l'intermédiaire des pattes est aussi très fréquente.

2.3.3 La communication visuelle

Très utilisée au stade primaire, elle n'est pratiquée que par certaines espèces et pour des situations bien particulières. Les expériences menées sur l'espèce méditerranéenne Cataglyphis cursor par les chercheurs du laboratoire d'éthologie et de psychologie animale (CNRS-Université Paul-Sabatier, Toulouse) et publiées dans la revue *Nature* (24/06/99) [189] ont donné des résultats stupéfiants. Leur objectif : entraîner des fourmis à retourner au nid par le chemin le plus court, en traversant un labyrinthe constitué de quatre boîtes successives. Sur chacune des boîtes, deux issues, chacune surmontée d'un dessin noir ne différant de l'autre que par sa forme géométrique (rond/croix, étoile/carré, rectangle/triangle, losange/ovale). De ces deux issues, une seule, conduit à la boite suivante. Or les fourmis, après quelques séances d'entraînement, choisissent sans se tromper, sans même hésiter, et dans le bon ordre, la séquence des repères visuels qui les ramèneront le plus rapidement au nid.

FIG. 2.2 – Dépôt de la phéromone par une fourmi : la phéromone est déposée sur le sol par la fourmi au moyen de son aiguillon

2.3.4 La communication chimique

Les moyens de communication utilisés par les fourmis sont donc nombreux. Toutefois leur efficacité est insignifiante comparée à celle de la communication chimique. Elle se fait grâce à des phéromones (Fig. 2.2), que seules les fourmis peuvent sentir. C'est grâce aux antennes que les odeurs sont distinguées. Ces phéromones sont des subtances olfactives et volatiles : il s'agit en fait de composés aromatiques. Les fourmis ont un odorat très développé comparativement à l'homme, ce qui leur permet de l'utiliser pour communiquer.

La phéromone déposée par la fourmi est une véritable "carte d'identité", elle contient à la fois les informations sur l'espèce, sur la société, sur la caste et sur le stade de développement des individus.

C'est un système de communication très perfectionné qui permet à la fourmi d'avertir ses congénères d'un danger ou de la localisation de la nourriture.

La phéromone entre en jeu également pour d'autres types de messages. La reine, par secrétion de ces substances, peut inhiber la ponte chez ses filles. De même chez les mâles, la caste des soldats pourra à l'aide du même procédé contrôler le nombre de larves destinées à être soldats. Ces substances permettent donc d'obtenir un véritable équilibre social au sein de la fourmilière. Toutefois d'autres formes de communications chimiques peuvent intervenir, comme les déchets fécaux, porteurs d'une odeur particulière, caractéristique à chaque colonie. Elles permettent la délimitation de leur territoire chez les tisserandes.

Les fourmis utilisent donc des messages chimiques dont chacun est porteur de plusieurs significations, parmi lesquelles on peut distinguer :

- l'attraction ;
- le recrutement ;
- l'alerte ;
- l'identification des autres castes ;
- la reconnaissance des différents stades de développement ;
- la discrimination entre fourmis étrangères et congénères.

2.4 Entomologie de la fourmi

La fourmi est un insecte social de la superfamille des Formicoïdes et appartient à l'ordre des hyménoptères comme les abeilles et les guêpes. Elle vit en colonies dans des habitations collectives à la structure complexe, les fourmilières.

2.4.1 L'habitat des fourmis

Chez beaucoup d'espèces, les fourmilières sont constituées de chambres et de galeries creusées sous des souches, des pierres ou à même le sol. Ces structures peuvent s'étendre sur plusieurs mètres en profondeur. Pour d'autres espèces, comme les fourmis tisserandes, le nid est fait à base de feuilles. Chez les fourmis rousses des bois, ce sera un amas d'aiguilles de pin et autres débris végétaux placés à la surface du sol et dont la hauteur varie en fonction du climat. Plus le climat est rude, plus le dôme est élevé, jusqu'à atteindre 1m de haut. A l'intérieur, se trouve une véritable cité très organisée : les ouvrières d'entretien nettoient les galeries sans relâche et les issues sont ouvertes régulièrement pour assurer l'aération.

2.4.2 La vie sociale des fourmis

Les fourmis sont apparues sur terre il y a 100 millions d'années, contre 90000 ans pour l'homo sapiens. Le nombre d'individus par colonie est très variable. Chez les magnans on peut compter jusqu'à 20 millions d'individus, contre guère plus de 20 chez d'autres espèces. Chaque colonie est divisée en castes, constituées d'individus qui ont des rôles et des aspects différents. C'est aussi le cas pour d'autres insectes sociaux comme les termites, les abeilles et les guêpes. On y trouve :

- la caste des reproducteurs qui comprend des femelles (reines) et des mâles ;
- la caste des ouvrières spécialisées, constituée de nombreuses petites femelles stériles, qui ont comme rôle, soit la recherche de la nourriture, soit la construction et la réparation du nid, soit le nettoyage, soit les soins des larves, soit encore comme chez les fourmis charpentières, n'importe quelle activité en cas de nécessité ;
- la caste des soldats, spécialisée dans l'attaque et la défense de la communauté, présente chez diverses fourmis comme les magnans d'Afrique.

Les fourmis sont les insectes les plus répandus sur la surface de la terre : elles sont présentes des steppes de l'Oural en passant par la forêt amazonienne jusque dans des déserts. On ne trouve pas un kilomètre carré dépourvu de ces insectes. Elles ne pèsent qu'un millionième de notre poids, mais si nous considérons leur masse totale, elles peuvent nous disputer le titre d'organisme social prédominant à la surface du sol. L'entomologiste britannique C.B. Williams a tenté de donner quelques chiffres : il estime le nombre d'insectes à 10^{18}, dont les fourmis représentent environ 1%, ce qui donne un poids de fourmis du même ordre de grandeur que celui de l'humanité. Les espèces hautement sociales d'insectes sont au nombre de 13500 sur 750000 espèces d'insectes répertoriées, et parmi ces espèces sociales 9500 sont des fourmis.

2.5 La fourmi insecte eusocial.

Le caractère le plus connu des fourmis est leur eusociabilité (mode de vie sociale le plus évolué). Il est à noter que "vie sociale" n'est pas synonyme de "vie de groupe". En effet, de nombreux stades séparent l'animal solitaire de l'animal eusocial : il y a l'animal grégaire (par l'interattraction entre congénères), subsocial (par l'apparition des comportements parentaux), colonial (par l'existence d'un site d'élevage commun

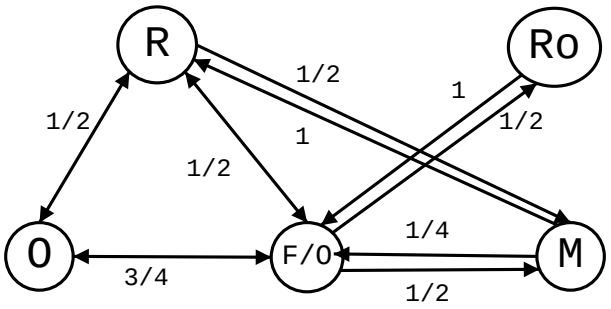


FIG. 2.3 – La parentèle : Les valeurs sur les flèches indiquent le facteur de similitude génétique entre les différents membres d'une colonie de fourmis. R désigne la reine, O une travailleuse, F une future reine, M un mâle fils de la reine et Ro le mâle qui s'est accouplé avec la reine

des jeunes, mais où chaque femelle/mâle travaille pour soi-même), communal (où les femelles/mâles coopèrent aux soins des jeunes).

2.5.1 Définition de l'eusociabilité

L'eusociabilité, beaucoup étudiée par le sociobiologiste E. O. WILSON[238, 117, 118], se définit par certains critères stricts :

- une coopération dans le soin aux jeunes ;
- le chevauchement des générations (les descendants aident leurs parents) ;
- des individus spécialisés dans la reproduction, tandis que d'autres, plus ou moins stériles (de façon définitive ou temporaire), se chargent des tâches communautaires en se les répartissant (polyéthisme).

Le secret de l'altruisme de certains membres de la société qui sacrifient leur sexualité à un petit nombre d'individus, chargés de les représenter, réside dans leur apparentement, et est dû à la sélection de la parentèle.

2.5.2 Qu'est ce que la parentèle ?

Une définition succincte de ce principe a été introduite par DARWIN dans *De l'origine des espèces*[46], ouvrage offrant les bases du modèle de la parentèle (fig. 2.3). Pour DARWIN, la sélection ne se fait pas pour les fourmis au niveau des individus mais au niveau de la famille, ce qui explique avec l'évolution, l'apparition de membres altruistes qui renoncent à leur reproduction. La théorie de la sélection de parentèle est donc une version modifiée de la sélection naturelle. Elle explique pourquoi certains individus favorisent ou défavorisent la reproduction d'autres individus qui leurs sont apparentés, en accord avec leurs degrés de similitude génétique.

En 1964, l'entomologiste et généticien britannique HAMILTON [110] définit ainsi la parentèle : pour que le trait altruiste d'un individu se répande au cours de l'évolution, le bénéfice obtenu par les individus qui lui sont apparentés doit dépasser l'inverse du taux de parenté. Dans le cas d'un frère ou d'une sœur, qui partagent en moyenne la moitié de leurs gènes, l'inverse d'un demi etant deux, le comportement d'altruisme peut donc apparaître si le sacrifice de l'un fait au moins doubler la fécondité de l'autre.

2.5.3 Autres sociétés eusociales

Les insectes eusociaux se trouvent dans l'ordre des isoptères (termites) et dans celui des hyménoptères (fourmis, abeilles et guêpes) et il existe quelques exemples de vertébrés eusociaux comme les rats-taupes. HAMILTON en déduit que l'altruisme s'est particulièrement développé chez les hyménoptères à cause de leur mode de reproduction qui est haplodiploïde : les œufs fécondés, donc diploïdes (c'est-à-dire, munis de deux jeux de chromosomes) donnent des femelles, les œufs non fécondés, donc haploïdes (munis d'un seul jeu) donnent des mâles. Deux sœurs ont en commun les trois quarts de leurs gènes en moyenne. Ce taux exceptionnel de parenté est dû au fait que leur père, issu d'un œuf non fécondé, transmet donc à ses filles des gènes tous identiques. Par conséquent, les sœurs chez les hyménoptères sont génétiquement plus proches les unes des autres que chez les autres animaux. Cet atout peut devenir un problème dans les nids comprenant plusieurs reines (polygynie), où leur surnombre peut conduire à des conflits, voire à la disparition de la colonie.

2.5.4 L'eusociabilité, une classification amendée.

Comme dans toute tentative de classification, certains cas posent problème. Ainsi les fourmis Prismyrmex pungens n'ont pas de reine et se reproduisent toutes par parthénogenèse thélytoque (ne donnant que des femelles). Malgré leurs particularités, il semblerait ridicule d'écarter ces fourmis japonaises des insectes eusociaux. Une distinction est donc faite entre les espèces primitivement eusociales (celles dont toutes les femelles se reproduiraient) et les espèces hautement eusociales (avec spécialisation dans la reproduction). Même si la vie sociale paraît hautement intéressante, ses inconvénients sont indéniables. Les bénéfices sont un meilleur repérage des prédateurs, une défense collective, la recherche de nourriture et l'élevage des jeunes en commun. Les inconvénients sont la concurrence alimentaire ou reproductive, les risques d'épidémies, le parasitisme social (certains profitant des autres sans restituer de contrepartie équivalente). Autrement dit, les sociétés animales, pas plus que les sociétés humaines, ne sont idéales, et dans les deux cas des conflits sont possibles.

2.6 Quelques exemples de résolution de problèmes complexes

Les fourmis peuvent résoudre de façon collective des problèmes complexes, c'est ce que nous allons illustrer à l'aide de deux exemples.

2.6.1 La formation d'un nid végétal

La canopée n'offre que peu de cavités pour héberger les fourmis tisserandes (oecophylla). Pour pallier ce problème, elles construisent leurs nids en tissant ensemble des brindilles et des feuilles, ce qui forme de grandes chambres avec murs, sol et toit (fig. 2.4). Le premier à décrire ces nids fut JOSEPH BANKS qui accompagna le capitaine COOK lors de son voyage en Australie en 1768. Il écrivit : "Elle construit un nid, de taille comprise entre celle de la tête et celle du poing d'un homme, en courbant

a)

b) 

c) 

d) 

FIG. 2.4 – Coopération de fourmis tisserandes pour la construction de leurs nids en feuilles : pour la construction, il est nécessaire de rapprocher les feuilles ; si leur écartement est plus faible que la taille d'une fourmi, elles attrapent les deux extrémités et les rapprochent (première image de a et c) ; si elles sont plus éloignées elles construisent des ponts (b). Les extrémités sont ensuite collées avec la soie des larves (seconde image de a et d)

les feuilles les unes vers les autres et en les collant solidement ensemble à l'aide d'une substance blanchâtre cartonneuse. Leur conduite était fort curieuse : elles recourbent quatre feuilles plus larges que la main, et les déposent dans la direction de leur choix, œuvre qui requiert une force bien plus grande que celle dont ces animaux semblent capables. De fait, elles sont bien des milliers à unir leurs efforts ; je les ai vues qui pliaient une telle feuille, en nombre aussi grand que possible les unes à côté des autres, chacune tirant de toutes ses forces, tandis que d'autres s'activaient à fixer la glu."

Comme l'a décrit J. BANKS, les fourmis peuvent s'aligner avec une très grande précision par centaines et rapprocher les bords d'une feuille. Si l'écartement est supérieur à la taille d'une fourmi, une autre stratégie est utilisée. Elles s'alignent les unes à la suite des autres de manière à former un pont vivant. Une première fourmi s'agrippe à la feuille, une seconde descend le long de la première et l'attrape à la taille, d'autres fourmis font de même jusqu'à atteindre l'autre extrémité qui est saisie par la dernière, et le processus de rapprochement peut commencer. Si une seule chaîne ne suffit pas, d'autres structures identiques seront formées sur les côtés. Pendant cette tâche, certaines d'entre elles vont recruter des renforts grâce à des substances odorantes. Celles-ci sont posées non seulement sur les feuilles et les branches mais également sur les corps des fourmis formant la chaîne. Très rapidement se constitue une nappe vivante de fourmis qui œuvrent ensemble. La substance qui est déposée est la phéromone dont nous avons parlé plus haut. Dans le cas des fourmis tisserandes, elle a pour but de conduire de nouvelles recrues vers la feuille à plier et ensuite de les guider sur la chaîne pour en atteindre l'extrémité.

2.6.2 La recherche de nourriture ou fourragement

Un autre exemple est celui de la fourmi moissonneuse (messor barbarus). Lorsqu'une ouvrière chargée de rechercher de la nourriture quitte le nid, elle va déposer sur

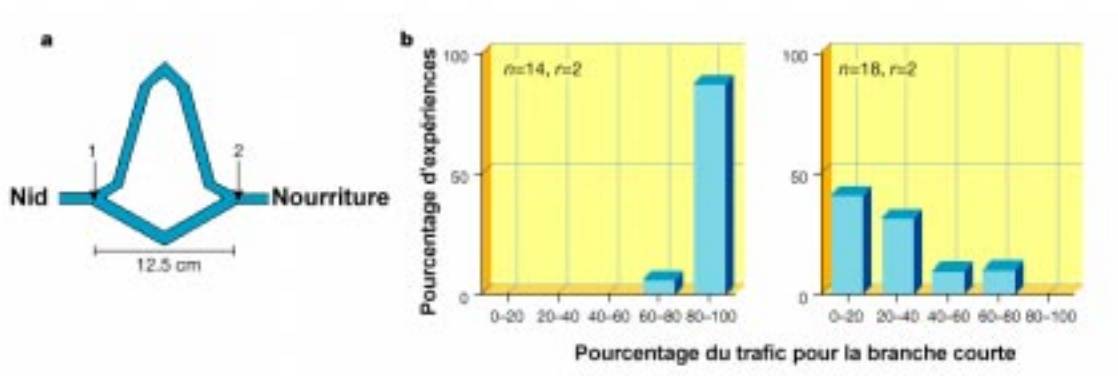


FIG. 2.5 – L'expérience du pont double : a) l'environnement de l'expérience, b) les graphes donnent la distribution des pourcentages de sélection de la branche la plus courte pour un jeu d'expériences. La branche du haut est r fois plus longue que l'autre. Le graphe de gauche (14 expériences) donne la répartition lorsque les deux branches sont présentées simultanément. Pour le graphe de droite (14 expériences), la branche la plus courte n'est présentée que 30 minutes après l'autre : la branche la plus longue est toujours exploitée à cause du dépôt de phéromone initial.

son chemin, à l'aide d'un aiguillon placé sur son abdomen, une fine trace de phéromone d'orientation. Lorsqu'elle retournera au nid après avoir trouvé une source d'approvisionnement, elle déposera à nouveau des phéromones, un peu différentes puisque ici une information sur la qualité du site trouvé sera incluse dans ce message. Dans tous les cas, la phéromone est placée là pour les autres fourmis de la colonie afin de les inciter à choisir cette piste.

Ce marquage dans l'expérience du pont double (Fig. 2.5) leur permet également de trouver le plus court chemin entre le nid et un site alimentaire. En effet, au départ chaque fourmi choisira l'une des pistes disponibles et totalement au hasard. Très rapidement le chemin le plus court sera marqué par plus de phéromone, car le va-et-vient des fourmis y est plus rapide, et pour une même période, plus de fourmis ont le temps de le parcourir, donc de déposer leur message. Les nouvelles fourmis qui quittent le nid ont une tendance naturelle à favoriser la piste qui comporte la trace olfactive la plus importante. Le caractère volatile de la phéromone accélère encore cette valorisation de la meilleure piste. C'est donc grâce à un échange d'informations que ces fourmis parviennent à trouver le chemin le plus court.

2.6.3 Stigmergie : définition

Dans les deux exemples que nous venons de citer, la phéromone est le vecteur d'une communication qui est appelée stigmergie, définie par GRASSÉ en 1959 [105]. Il s'agit d'une forme indirecte de communication, qui se traduit par une modification locale de l'environnement : pour les fourmis c'est le dépôt de la phéromone. Ces échanges d'informations ont rendu possible l'émergence d'une auto-organisation. Par ce terme est définie l'apparition de comportements complexes à partir de comportements individuels simples. Un comportement général complexe n'implique donc pas des stratégies complexes au niveau des individus.

2.7 Conclusion

Les fourmis occupent des niches écologiques très variées pour lesquelles elles se sont spécialisées. Les atouts qui leur ont permis de conquérir les sites de nidification les plus attirants et les plus stables, face aux autres insectes sont leur caractère eusocial et leur capacité d'auto-organisation, cette dernière étant rendue possible par la stigmergie. La capacité à résoudre des problèmes complexes et à s'auto-organiser a poussé les informaticiens à faire le lien entre le monde réel et celui de l'informatique en créant des agents fourmis virtuels. Ces avatars sont utilisés pour concevoir différents algorithmes d'optimisation s'inspirant du mécanisme de coopération des fourmis. C'est ce que nous verrons dans les chapitres suivants.

Chapitre 3

Les fourmis artificielles

Ce second chapitre nous fera découvrir le modèle de la fourmi informatique, créé en s'inspirant du comportement naturel de ces insectes. Nous y trouverons les applications réalisées à l'aide de ces agents virtuels et nous pourrons constater que celles-ci ont été utilisées pour la résolution de problèmes complexes. Nous nous intéresserons également à la manière de simuler la stigmergie pour les fourmis artificielles.

3.1 La fourmi numérique

La fourmi artificielle se présente sous la forme d'un ensemble de procédures qui définissent son comportement. Celui-ci est très semblable à celui de la fourmi naturelle quand elle recherche de la nourriture (Fig. 3.1).

Dans ce cas, une fourmi n'a qu'un rôle assez simple qui consiste à se déplacer du nid jusqu'à la source de nourriture et à y revenir. Le code qui définit leur comportement permet aux fourmis artificielles de se déplacer dans l'espace combinatoire formé par les différents éléments qui peuvent être utilisés pour le problème à résoudre. Pour utiliser un vocabulaire informatique, nous dirons qu'elle construit une solution.

La mémorisation de ces déplacements donne la forme d'une solution où chaque étape est désignée par l'indice de l'élément et où l'ordre de parcours désigne la position des éléments dans la solution.

3.2 La fourmi virtuelle vue comme un agent

L'utilisation des fourmis peut être vue comme celle d'agents communiquants grâce à la stigmergie [105, 60]. En regardant de plus près les définitions on peut constater qu'il est possible d'utiliser également la notion d'agent dans le cadre d'algorithmes comme ceux utilisant le modèle des colonies de fourmis.

Pour commencer il est nécessaire de se poser la question : Qu'est ce qu'un agent ? Cette appellation peut sembler assez imprécise, c'est pourquoi nous prendrons les définitions de Ferber [74].

Définition 1 *On appelle agent une entité physique ou virtuelle :*

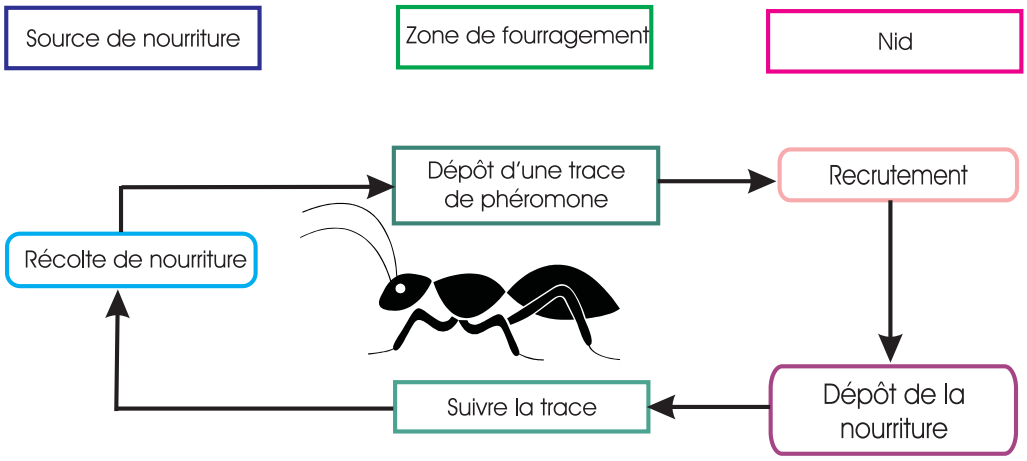


FIG. 3.1 – Comportement d'une fourmi naturelle lors de la recherche de nourriture (dans le cas des fourmis artificielles, il n'y a pas obligatoirement de phase de recrutement.)

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

Définition 2 *Par comparaison avec la définition générale d'un agent donné précédemment, on appelle agent purement communicant (ou agent logiciel) une entité informatique qui :*

1. *se trouve dans un système informatique ouvert (ensemble d'applications, de réseaux et de systèmes hétérogènes) ;*
2. *peut communiquer avec d'autres agents ;*
3. *est mue par un ensemble d'objectifs propres ;*
4. *possède des ressources propres ;*
5. *ne dispose que d'une représentation partielle des autres agents ;*
6. *possède des compétences (services) qu'elle peut offrir aux autres agents ;*
7. *a un comportement tendant à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose et en fonction de ses représentations et des communications qu'elle reçoit.*

En se basant sur ces définitions, il est possible de décrire le comportement des fourmis virtuelles en tant qu'agents :

- les entités informatiques que nous venons de définir sont dites virtuelles car elle n'ont pas d'existence matérielle au contraire des entités physiques qui interagissent dans le monde concret (Des exemples de ces agents physiques sont un robot, un avion, une voiture) ;
- les agents sont capables d'agir : dans le cas des fourmis virtuelles, elles modifient les valeurs de phéromone associées aux différents éléments. Par cette action elles changent leur environnement, ce qui influera sur le choix des autres fourmis à l'itération suivante ;
- les agents sont capables de communiquer : les fourmis utilisent comme on l'a vu précédemment la phéromone comme médium de communication indirecte ;
- les agents sont doués d'autonomie : chaque fourmi a pour but de construire une solution pour un problème donné, se contentant pour cela d'appliquer les règles de sélection qui définissent son comportement, la fourmi utilise la phéromone et parfois des valeurs heuristiques ;
- les agents n'ont qu'une représentation partielle de leur environnement : lors de la construction d'une solution, la fourmi ne connaît à chaque étape que les éléments qu'elle a déjà choisis et les valeurs de phéromone correspondant aux éléments qui pourront l'être. Ces informations donnent à la fourmi une représentation très partielle de son environnement.

Les fourmis numériques dont nous parlerons dans ce mémoire peuvent donc être décrites sous la forme d'agents. Il est toutefois à noter que ces fourmis peuvent être classées dans les agents réactifs car contrairement aux agents cognitifs, elles ne peuvent pas acquérir de nouvelles connaissances et se contentent d'agir/réagir suivant les informations qu'elles trouvent dans leur environnement.

Pour finir ce court survol de la notion d'agent, il semble nécessaire de définir ce que l'on appelle un système multi-agents.

Définition 3 *On appelle système multi-agents (ou SMA), un système composé des éléments suivants :*

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

Le modèle multi-agents est d'abord utilisé pour la simulation du comportement naturel des fourmis pour valider des modèles de comportement [17, 39].

C'est cette formalisation qui est également utilisée en robotique pour décrire le comportement des robots fourmis. Les robots fourmis réussissent à réaliser des actions complexes avec des modes de perception réduits. On distingue deux types de robots inspirés des fourmis :

- les autonomes, qui souvent s'inpirent des capacités individuelles de cet insecte comme la vision [132, 81], la détection de traces chimiques [184], le déplacement [174] ;
- les coopératifs, qui peuvent ramasser des objets dispersés [50] ou explorer des zones inconnues comme la surface de Mars dans un des projets de la NASA.

3.3 Le modèle de coopération

Comme nous avons pu le voir dans le chapitre précédent, les fourmis peuvent résoudre collectivement des problèmes complexes. Un des exemples marquant est celui de la découverte du chemin le plus court dans l'expérience du choix entre les deux arcs d'un pont. Chacune des fourmis, isolément, ne peut trouver la meilleure solution. C'est grâce au mécanisme d'auto-organisation qui émerge de la communication indirecte (stigmergie [105]) que le plus court chemin peut être découvert.

Le modèle de la stigmergie peut être facilement étendu au agents artificiels en associant aux éléments d'un problème des variables d'état spécifique, qui serviront de support à la communication indirecte entre fourmis. Dans le cas de la recherche de nourriture, nous avons vu que les fourmis utilisent la stigmergie en déposant la phéromone sur la piste qu'elles empruntent. La fourmi artificielle pour sa part modifiera les valeurs des variables associées (phéromone) aux éléments du problème qu'elle aura choisis lors de la construction de sa solution.

Dans la nature, les fourmis vont parcourir plus vite le chemin le plus court, ce qui implique que la quantité de phéromone qui y est déposée augmentera plus rapidement. Pour simuler ce mécanisme, il faut intégrer un processus de renforcement positif. Dans le cas des fourmis, plus le chemin est court, plus les fourmis le parcourront vite et plus la quantité de phéromone associée sera élevée. Dans le cas d'un problème, cela revient à dire que les valeurs des variables associées aux éléments appartenant aux meilleures solutions doivent être davantage renforcées que les autres. Afin de pouvoir comparer les solutions, il est nécessaire d'introduire une mesure qui donne l'adéquation de la solution au problème : c'est la fonction qualité (en anglais *fitness*). La valeur de cette qualité est utilisée dans le calcul de la mise à jour des valeurs associées aux éléments du problème lors de la phase de renforcement. Pour la recherche du plus court chemin, on peut prendre tout simplement la distance comme mesure de qualité : plus le chemin est court, plus il sera renforcé.

Une des propriétés de la phéromone est son caractère volatil. Dans le modèle virtuel, un mécanisme similaire sera utilisé afin d'éviter une convergence prématurée (stagnation) due à la découverte d'un optimum local.

Entre ces deux modèles, nous passons d'un univers continu à un univers discret. Ainsi, les fourmis virtuelles sautent d'un élément à un autre, tandis que les fourmis naturelles progressent de façon continue sur le chemin. Cette discrétisation impose que l'évaporation ou la modification des valeurs de phéromone ne puissent se faire en continu. Cet ajustement de valeurs n'est souvent réalisable qu'après la construction de la solution complète. Dans le modèle artificiel, il est également nécessaire de définir

les étapes (éléments) et leur voisinage pour permettre le parcours de la fourmi. Ces définitions sont liées à la nature du problème considéré.

3.4 Les différents domaines d'application

La première application du modèle des fourmis virtuelles a été faite sur un problème de recherche du plus court chemin [104, 51]. Les bonnes performances obtenues ont incité les chercheurs à l'utiliser pour la résolution d'autres problèmes que nous présenterons ensuite.

3.5 Les fourmis artificielles pour la recherche opérationnelle

Les algorithmes de cette section peuvent être regroupés sous le terme de méta-heuristiques d'optimisation à colonie de fourmis (OCF) ou Ant Colony Optimization meta-heuristics (ACO) [62].

Dans les OCF, la colonie comporte un nombre de fourmis artificielles fixe et a pour but de rechercher collectivement de bonnes solutions pour le problème à résoudre. Le rôle d'une fourmi est de construire une solution, en partant d'une position initiale liée à la nature du problème considéré. L'objectif est d'obtenir la solution qui passe par les éléments (étapes) du problème tout en entraînant un coût minimum et en respectant les contraintes définies.

Lors de cette construction, la fourmi accumule des informations sur l'environnement parcouru. Ces renseignements sont utilisés pour modifier la perception que les autres fourmis ont du problème. Chaque fourmi peut travailler de façon indépendante ou en concurrence mais elle coopère toujours. Ces foumis collaborent donc en communiquant de façon indirecte ; les informations qu'elles échangent sont stockées sous forme de taux de phéromone. On peut dire qu'il s'agit de variables stigmergétiques.

Les fourmis construisent leur solution de façon incrémentale. Elles vont choisir à chaque étape un élément dans leur voisinage (parmi les éléments possibles) et ce choix est guidé par les informations laissées par les autres fourmis. Ce sera la direction (l'élément) marquée par la plus grande quantité de phéromone qui aura la plus forte probabilité d'être choisie.

Les fourmis conservent une mémoire du parcours qu'elles ont déjà effectué. Cette connaissance est à la fois utilisée pour définir le voisinage à chaque étape, pour contrôler le respect des contraintes liées au problème, pour évaluer la solution et pour quantifier le renforcement en phéromone.

La forme générale des OCF est donnée dans l'Agorithme 3.1. Les différentes variantes peuvent être caractérisées en décrivant le comportement individuel des fourmis et la procédure de mise à jour de la phéromone.

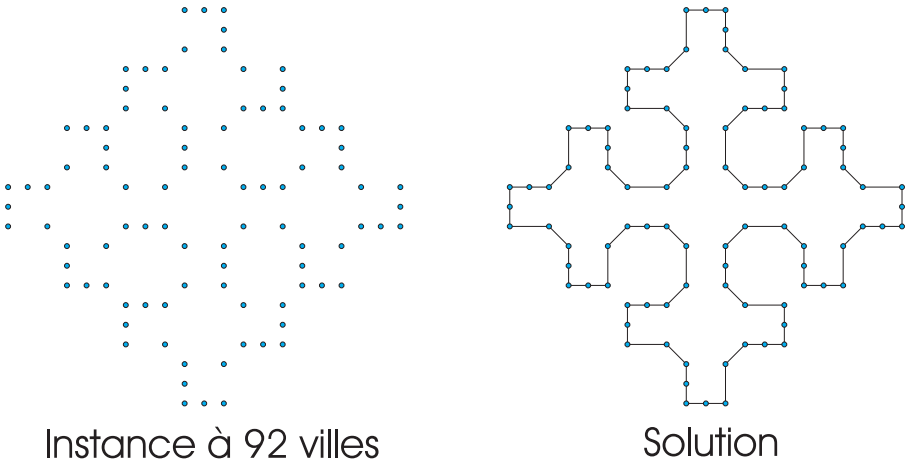


FIG. 3.2 – Exemple d'une instance créée de 92 villes, accompagnée de sa solution

Algorithme 3.1: L'algorithme général des OCF.
L'action du **démon** est optionnelle, et fait référence à des actions centralisées.
OCF()

(1) **tant que** (le critère de fin n'est pas satifait)
(2) génération et activité des fourmis
(3) mise à jour de la phéromone
(4) actions du démon (ex : une deuxième mise à jour) [optionnel]
(5) **fin tant que**

Des stratégies peuvent être ajoutées pour améliorer les performances des systèmes de fourmis. Il s'agit de donner aux fourmis des comportements qui n'ont ici rien à voir avec le modèle naturel. Il est possible d'intégrer un facteur supplémentaire lié à la nature du problème pour guider le choix lors de la sélection de l'étape suivante. Dans le cas de la recherche du plus court chemin, la longueur des chemins peut être combinée avec les valeurs de phéromone. Ces deux informations permettent d'obtenir une table de décision qui a pour but de diriger la recherche des fourmis dans les bonnes régions de l'espace de recherche. Pour le problème de chemin le plus court, les éléments les plus attirant seront donc les arcs qui auront les plus forts taux de phéromone et la plus coute longueur.

Un mécanisme de recherche locale peut également être hybridé avec les fourmis, pour améliorer la solution générée et ainsi augmenter les performances de l'heuristique. Cette étape s'intègre à chaque itération avant la mise à jour de la phéromone.

3.5.1 Le voyageur de commerce

M. DORIGO [66, 58] a été le premier à définir un algorithme à base de colonie de fourmis, il s'agit de Ant-System qui résout des instances du problème du voyageur de commerce (PVC) ou Traveling Salesman Problem (TSP) (fig. 3.2).

Le PVC repose sur un problème concret, celui d'un voyageur de commerce qui doit visiter ses clients. Pour cela, il doit planifier son itinéraire afin d'en minimiser les coûts

de transport. Le but est donc de trouver le parcours qui passera par toutes les n villes où sont situées les clients à visiter, une et une seule fois, en terminant le parcours par le point de départ et en parcourant le minimum de distance. Ce problème $\mathcal{NP}$-dur [91] peut être formalisé comme un graphe $G = (N, A)$ où les villes sont les N nœuds et les connexions entre villes les arcs A : le problème revient à trouver un circuit hamiltonien de longueur minimale.

Ce problème est proche de celui de la recherche du chemin le plus court dans le cas du problème des deux ponts pour les fourmis naturelles. En effet, il s'agit bien de trouver le plus court chemin mais avec les contrainte de villes étapes et avec le fait que la ville de départ est la même que celle de destination. Le problème ne peut pas être réduit à rechercher le plus court chemin entre la position actuelle et l'une des villes restant à visiter, car cette méthode ne fournit pas de bonne solution au PVC.

Notons que le graphe n'est pas obligatoirement entièrement interconnecté. Pour ce qui est des distances entre les villes, deux cas se distinguent :

- si la distance à parcourir entre deux villes est identique quelle que soit la ville de départ, on parlera de PVC symétrique ;
- si les valeurs d'aller et de retour sont différentes on parlera de PVC asymétrique.

Cette différence n'a pas d'influence sur le fonctionnement de la plupart des heuristiques, même si les performances obtenues ne sont pas les mêmes.

Ant-System pour le PVC (AS-TSP) est l'un des trois algorithmes qui furent proposés dans l'article de Colorni *et al.* en 1991 [36] . Celui-ci mettait en avant le comportement coopératif des fourmis pour la résolution du PVC. Nous aborderons uniquement l'algorithme désigné par le nom "ant cycle" qui a donné plus tard naissance à AS-TSP (ou AS) [67].

Comme la majorité des OCF, il peut être décrit suivant deux points : la stratégie de déplacement de la fourmi ou mouvement, et le mode de mise à jour de la matrice de phéromone. Les équations relatives à ces deux points pour cet algorithme ainsi que pour ceux décrits dans ce chapitre sont présentées respectivement dans les tableau 3.1 et 3.2 :

1. Mouvement de la fourmi

   Le fonctionnement de cet algorithme est le suivant : au départ, des fourmis sont positionnées sur les sommets du graphe que forment les villes de l'instance considérée ; les fourmis vont se déplacer de ville en ville en construisant une solution.

   Le choix de la ville suivante respecte la contrainte de ne passer qu'une fois dans chaque ville. Pour cela, une fourmi va mémoriser toutes les villes par lesquelles elle sera déjà passée sous la forme d'une liste (L_k). Ce choix sera guidé par deux facteurs, d'une part la valeur de phéromone à l'instant t qui est associée aux arcs pouvant les amener de la ville actuelle i à une ville j $(\tau_{i,j})$, d'autre part la longueur de ces arcs.

   La probabilité $(P_{i,j})$ de sélection d'un arc est dans ce cas dépendant à la fois du taux de phéromone présent sur cet arc et de son pouvoir attracteur, ce dernier étant obtenu en prenant l'inverse de la longueur. Les deux facteurs ont une action qui peut être modulée grâce à deux paramètres qui sont fixés à l'initialisation. Dans le cas des villes déjà visitées la probabilité de sélection est égale à zéro.

TAB. 3.1 – Equations relatives aux mouvements des fourmis pour les différents algorithmes

Nom de l'algorithme	Mouvement de la fourmi
AS-TSP	$P_{i,j} = \begin{cases} \text{si } j \in L_k(i) & \frac{[\tau_{i,j}]^\alpha \cdot [\upsilon_{i,j}]^\beta}{\sum_{j \in L_k(i)} [\tau_{i,j}]^\alpha \cdot [\upsilon_{i,j}]^\beta} \\ \text{sinon} & 0 \end{cases}$ $L_k(i)$ est la liste tabou pour la fourmi k arrivée au sommet i $\upsilon_{i,j}$ designe l'inverse de la distance entre les villes i et j α et β pondèrent l'influence de la phéromone et de la longueur τ_{ij} taux de phéromone entre les villes i et j
ANT-Q	$j = \begin{cases} \arg\{\max_{l \in J_k(i)}\{[AQ(i,l)]^\alpha \cdot [HE(i,l)]^\beta\}\} & \text{si } q \leq q_0 \\ J & \text{sinon} \end{cases}$ $HE(i,l)$ est une valeur fournie par une heuristique $AQ(i,l)$ donne la valeur de probabilité de choix de l en étant en i (phéromone) α et β pondèrent l'influence des deux mesures q_0 est la probabilité d'utiliser la première équation
AS_{rank}	Identique à AS-TSP
ACS	Semblable à Ant-Q avec $AQ(i,l) = \tau(i,l)$, $HE(i,l) = \nu(i,l)$ et $\alpha = 1$ $\nu(i,l)$ est l'inverse de la distance qui sépare les villes i et l

2. Mise à jour de la mémoire

Quand les fourmis ont fini de parcourir toutes les villes, les solutions qu'elles ont construites vont être évaluées. Dans notre cas la qualité de la solution est la distance parcourue. La mise à jour de la phéromone commence par une phase d'évaporation, qui réduit toutes les valeurs des arcs. Ensuite pour chaque solution, les valeurs de phéromone associées aux éléments la constituant sont renforcées par un facteur proportionnel à l'inverse de la longueur totale de la solution.

La forme générale de AS pour le PVC est présente dans l'Algorithme 3.2. Le nombre de fourmis a été choisi égal au nombre de villes ($m = n$).

TAB. 3.2 – Equations relatives à la mise à jour de la phéromone pour les différents algorithmes

Nom de l'algorithme	Mise à jour
AS-TSP	$\tau_{i,j} = \tau_{i,j}(1-\rho) + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$ $(1-\rho)$ est le facteur d'évaporation $\Delta\tau_{i,j}^{k}$ est l'apport de la fourmi k à l'arc (i,j) $\Delta\tau_{i,j}^{k} = \begin{cases} \frac{Q}{D_k} & \text{si la fourmi } k \text{ est passée} \\ & \text{par l'arc } (i,j) \\ 0 & \text{sinon} \end{cases}$ Q une constante D_k la distance parcourue par la fourmi
ANT-Q	$AQ(i,j) = (1-\rho)\cdot AQ(i,j) + \rho\left(\Delta AQ(i,j) + \gamma \cdot \max_{l\in J_k(i)} AQ(i,l)\right)$ γ est une valeur définie à l'initialisation $\Delta AQi,j = \begin{cases} \frac{W}{D^b} & \text{si } (i,j) \in \text{tour effectué par } k \\ 0 & \text{sinon} \end{cases}$ W est une constante définie à l'initialisation D^b est la distance parcourue par la meilleure fourmi (pour cette itération ou pour l'exécution)
AS$_{rank}$	$\tau_{i,j}(t+1) = \rho\tau_{i,j}(t) + \Delta\tau_{i,j} + \Delta\tau_{i,j}^{*}$ $\Delta\tau_{i,j} = \sum_{\mu=1}^{\rho-1} \Delta\tau_{i,j}^{\mu}$ $\Delta\tau_{i,j}^{\mu} = \begin{cases} (\rho-\mu)\frac{Q}{L_\mu} & \text{si la } \mu^{\text{ème}} \text{ meilleure fourmi est passée} \\ & \text{par l'arc } (i,j) \\ 0 & \text{sinon} \end{cases}$ $\Delta\tau_{i,j}^{*} = \begin{cases} (\rho-\mu)\frac{Q}{L^*} & \text{si la meilleure fourmi est passée} \\ & \text{par l'arc } (i,j) \\ 0 & \text{sinon} \end{cases}$ μ est l'indice dans le classement $\Delta\tau_{i,j}^{\mu}$ augmente le taux de phéromone des arc des μ meilleures solutions $\Delta\tau_{i,j}^{*}$ augmente le taux de phéromone des arc de la meilleure solution ρ nombre de solutions utilisées pour la mise à jour L_μ longueur du tour créé par la fourmi μ L^* longueur du tour créé par la meilleure fourmi
ACS	$\tau_{i,j} = (1-\rho)\tau_{i,j} + \Delta\tau_{i,j}$ $j = \begin{cases} \arg\{\max_{l\in J_k(i)}\{[\tau(i,l)]^\alpha \cdot [\nu(i,l)]^\beta\}] & \text{si } q \le q_0 \\ J & \text{sinon} \end{cases}$ $\Delta\tau_{i,j} = \tau_0$, τ_0 étant une valeur fixée

TAB. 3.3 – Comparaison de l'AS (Ant-cycle) avec d'autres algorithmes sur l'instance de PVC Oliver30

Nom de l'Algorithme	Meilleur
Ant-Cycle	420
Near Neighbour	587
Far Insert	428
Near Insert	510
Space Filling Curve	464
Sweep	486
Random	1212

Algorithme 3.2: L'algorithme AS-TSP.
AS-TSP()

(1) **Initialisation :** $\forall(i,j) \in \{1,...,n\}^2$ $\tau_{i,j} \leftarrow \tau_0$, placer chaque fourmi sur une ville choisie aléatoirement
(2) **pour** $t = 1$ **à** $t = t_{max}$ **faire**
(3) **pour chaque** fourmi k **faire**
(4) Construire un chemin $T_k(t)$ avec la règle de sélection
(5) Calculer la longueur $D_k(t)$ du parcours
(6) **finpour**
(7) Soit T^* le meilleur chemin trouvé et L^* sa longueur
(8) Mettre à jour les valeurs de phéromone
(9) **finpour**
(10) **retourner** T^*, L^*

Comme nous l'avons vu dans la section précédente, AS-TSP s'écarte du modèle des fourmis naturelles. En effet, les fourmis virtuelles possèdent une mémoire du chemin parcouru. Elles perçoivent les distances entre les villes et travaillent avec un temps discret qui leur impose de construire totalement la solution avant de déposer leur phéromone.

Les résultats encourageants (tab. 3.3) ont incité au développement d'autres algorithmes s'inspirant du même paradigme que pour le PVC.

Nous allons aborder les variantes les plus connues de AS-TSP qui ont été proposées.

Ant-Q

Dans "Ant-Q [87] : A Reinforcement Learning Approach to the Traveling Salesman Problem", GAMBARDELLA et DORIGO présentent AS sous la forme d'un algorithme d'apprentissage par renforcement (Q-learning) et en déduisent la forme générale d'une heuristique qu'ils appellent Ant-Q.

À chaque arc sont associées deux valeurs :
- une valeur appelée $AQ(i,j)$ correspondant plus ou moins au taux de phéromone. (Dans cet algorithme on utilise plutôt le terme statistique.)
- une valeur heuristique $HE(i,j)$ indiquant l'intérêt de l'arc (pour le PVC longueur inverse de l'arc).

TAB. 3.4 – Comparaison de Ant-Q avec d'autre méthodes classiques pour cinq instance de 50 villes. Les valeurs présentées sont les erreurs en pourcentage

Instance	EN	SA	SOM	FI	FI 2-opt	FI 3-opt	Ant-Q
1	5,98	5,88	6,06	6,03	5,99	5,90	5,87
2	6,03	6,01	6,25	6,28	6,20	6,07	6,06
3	5,70	5,65	5,83	5,85	5,80	5,63	5,57
4	5,86	5,81	8,87	5,96	5,96	5,81	5,76
5	6,49	6,33	6,70	6,71	6,61	6,48	6,18

Pour cet algorithme, les fourmis sont désignées également par le terme d'agents. L'algorithme est comparable à AS-TSP, les différences viennent de la fonction de mise à jour de la table de phéromone et du mouvement de la fourmi.

1. Mouvement de la fourmi :

   Une liste $(J_k(i))$ de villes candidates est définie, elle donne les villes restant à parcourir à la fourmi (k) placée dans la position (i).

   Le choix d'une ville peut se faire suivant deux méthodes : l'une déterministe où l'arc choisi sera le plus court qui comporte le plus fort taux de phéromone, l'autre probabiliste qui utilise une heuristique pour la sélection (trois heuristiques sont possibles).

   Le choix entre l'application de la méthode déterministe ou heuristique est effectué suivant une probabilité (q_0) définie comme paramètre.

   Pour la méthode heuristique trois variantes ont été testées :
   - une règle pseudo-aléatoire, où la ville est choisie aléatoirement parmi les villes candidates (cette méthode est la plus proche de la règle de choix d'action pseudo-aléatoire du Q-learning) ;
   - une règle proportionnelle pseudo-aléatoire, où la ville est choisie sur le principe de la roulette avec une formule proche de celle de AS-TSP ;
   - une règle proportionnelle aléatoire qui est identique à la règle précédente, à la différence près qu'elle sera la seule utilisée, la probabilité associée à la méthode déterministe étant égale à zéro. Dans ce cas précis, ANT-Q est équivalent à AS-TSP.

2. Mise à jour de la mémoire :

   ANT-Q se différencie d'AS par la mise à jour qui n'emploie que la meilleure solution lors du renforcement, tandis qu'AS utilise toutes les solutions générées.

Les tests indiquent que l'effet de la valeur heuristique est surtout perceptible au début de la recherche. La comparaison sur des instances de PVC symétrique démontre que les résultats (tab. 3.4) obtenus avec ANT-Q sont comparables avec d'autres algorithmes comme Elastic Net, recuit simulé, self organisation map, farthest insertion avec 2-opt, 3-opt ou sans. Tandis que sur des instances de PVC asymétrique, très complexes à résoudre, les performances (tab. 3.5) sont bonnes et même équivalentes à celles d'algorithmes spécialisés. Certes ANT-Q surpasse les résultats obtenus par AS, mais son coût est très élevé (en $o(mn^2)$), ce qui restreint son utilisation à de petites instances (m est le nombre de fourmis et n le nombre de villes).

Instance	FT-92	FT-94	Ant-Q Moyenne	Ant-Q Meilleur
ry48p	14422	14422	14690	14422
43x2	NA	5620	5625	5620

TAB. 3.5 – comparaison de AntQ avec des méthode exactes pour des instance de PVC asymétrique

TAB. 3.6 – Comparaison entre AS, ACS et MMAS pour ry48p

Méthode	Moyenne
AS	14622,24
ACS	14565,45
MMAS	14571,68

$\mathcal{MAX}$-$\mathcal{MIN}$

Une seconde variante fut proposée par STÜTZLE et HOOS, appelée $\mathcal{MM}$AS [209] ($\mathcal{MAX}$-$\mathcal{MIN}$ Ant System). Dans cette méthode, les deux auteurs n'utilisent que la meilleure solution pour la mise à jour de la phéromone afin d'accélérer l'algorithme, ce qui a pour défaut de rapidement piéger l'algorithme dans des optima locaux.

Pour pallier ce problème, les auteurs bornent le taux de phéromone par une valeur inférieure et supérieure, ce qui limite les variations des taux associés aux différents arcs et évite que certains arcs soient totalement délaissés. La valeur du τ_{max} est également utilisée comme valeur initiale. Pour de longues exécutions, où l'apparition d'une stagnation est encore possible, un mécanisme de "smoothing" a été introduit. Ce dispositif consiste à augmenter les taux de phéromone associés aux différents éléments par l'ajout d'une valeur qui est proportionnelle à la différence entre le taux actuel et le maximum.

Cette méthode intègre également une recherche locale qui améliore toutes les solutions ou uniquement la meilleure. Les comparaisons (tab 3.6) effectuées nous montrent que ces modifications augmentent de façon significative les performances obtenues avec AS. Les résultats de $\mathcal{MM}$AS sont même équivalents à ceux de ACS, méthode qui est présentée plus loin.

$\mathbf{AS}_{rank}$

BULLNHEIMER *et al.* [23] ont proposé une version élitiste d'AS nommé ASrank. Dans cette heuristique, les solutions générées sont classées selon un ordre croissant, en fonction de leur qualité (longueur du circuit).

Seules les ρ meilleurs solutions sont prises en compte pour le renforcement de la mémoire de phéromone (ρ : valeur fixée à l'initialisation ; dans le cas des tests elle valait 5).

Cette méthode obtient de bons résultats (tab. 3.7), améliore les performance d'AS et obtient de bons résultats sur les instances étudiées face aux autres algorithmes.

ACS

L'algorithme Ant Colony System (ACS) [64] proposé par M. Dorigo et L. Gambardella tente d'unifier toutes les variantes d'AS qui ont été proposées. Le schéma de

TAB. 3.7 – Comparaison d'AS_{rank} avec AS et une autre variante proposée dans le même article, on trouve également les résultats obtenus avec des méthodes classiques. Les résultats sont présentés pour des instances de tailles différentes.

Nombre de villes	Méthode	Moyenne	Meilleur
30	SA	424,52	423,74
	SA_{nn}	424,26	423,74
	GA	424,42	423,74
	AS	426,24	423,91
	AS_{elite}	426,08	423,74
	AS_{rank}	425,72	423,74
57	SA	924,62	920.08
	SA_{nn}	925,79	920.08
	GA	927,13	920.08
	AS	930,86	924,20
	AS_{elite}	928,00	920,08
	AS_{rank}	926,91	920,08
80	SA	382,41	370,97
	SA_{nn}	380,49	370,97
	GA	380,23	373,51
	AS	380,19	373,36
	AS_{elite}	374,44	370,97
	AS_{rank}	373,74	370,97
96	SA	1101,78	1049,98
	SA_{nn}	1079,89	1043,70
	GA	1074,93	1056,67
	AS	1068,85	1053,26
	AS_{elite}	1055,14	1045,98
	AS_{rank}	1055,00	1043,70
132	SA	1596,09	1558,53
	SA_{nn}	1577,69	1537,23
	GA	1588,99	1543,14
	AS	1568,02	1544,30
	AS_{elite}	1558,15	1537,73
	AS_{rank}	1556,65	1533,54

TAB. 3.8 – Comparaison sur des instances symétriques et asymétriques de l'ACS-3opt avec un algorithme génétique (ATSP), les résultats étant obtenus pour dix exécutions [64].

		ACS-3opt		ATSP	
	instance	meilleur	moyenne	meilleur	moyenne
Symétrique					
	d198	15780	15781,7	15780	15780
	lin318	42029	42029	42029	42029
	att532	27693	27718,2	27686	27693,7
	rat783	8818	8837,9	8806	8807,3
Asymétrique					
	p43	2810	2810	2810	2810
	ry48p	14422	14422	14422	14440
	ft70	38673	38679,8	38673	38683,8
	kro124p	36230	36230	36230	36235,3
	ftv170	2755	2755	2755	2766,1

fonctionnement est globalement le même que celui d'AS.

Il se distingue par plusieurs points :

1. Le mouvement de la fourmi est :
   - soit déterministe de la manière introduite par Ant-Q ;
   - soit aléatoire suivant une probabilité proportionnelle à la phéromone et à l'attraction de l'arc (comme pour AS-TSP).

   Le choix entre les deux techniques est probabiliste.
2. Deux méthodes sont utilisées pour la mise à jour :
   - un renforcement local : il est effectué par les fourmis lors de la création de la solution, on y retrouve l'évaporation et un renforcement par une valeur constante ;
   - un renforcement global classique de la phéromone tel que celui d'AS-TSP, mais ici seule la meilleure solution est utilisée pour la phase de renforcement.

DORIGO et GAMBARDELLA proposent dans le même article ACS-3opt (tab. 3.8), qui hybride l'AS-TSP avec un 3-opt [64]. Cette hybridation consiste en l'intégration d'une méthode de recherche locale, dont le but est d'améliorer les solutions trouvées par les fourmis avant la sélection de la meilleure. La méthode d'optimisation locale est un 3opt restreint (fig. 3.3), il s'agit de permuter 3 arcs mais sans changer l'ordre dans lequel les villes sont visitées. Le but est de conserver la capacité d'ACS à résoudre des instances symétriques et asymétriques du PVC.

3.5.2 Problème d'ordonnancement séquentiel

Le problème d'ordonnancement séquentiel (POS) ou Sequencial Ordering Problem (SOP) avec des contraintes de précédence consiste à trouver un tour hamiltonien ayant un poids minimum dans un graphe orienté dans lequel les poids sont associés aux arcs et aux nœuds en respectant des contraintes de précédence entre des nœuds.

Le POS peut être vu comme une généralisation du problème du voyageur de commerce asymétrique (PVCA). Dans cette représentation $c_{i,j}$ est le poids de l'arc allant de i à j, le poids de $c_{j,i}$ pouvant être différent. La valeur $c_{i,j}$ peut représenter le coût de

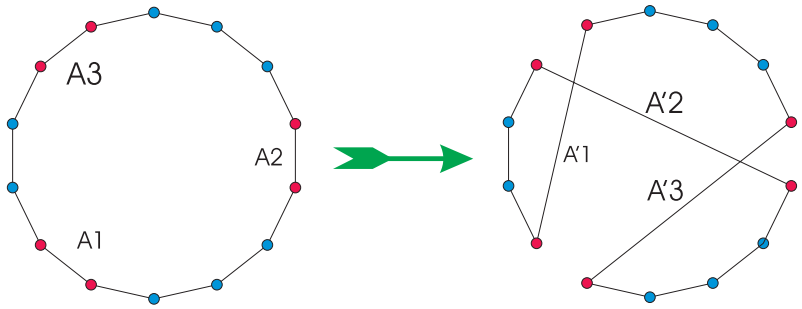


FIG. 3.3 – La méthode d'optimisation locale 3-opt consiste en l'échange de trois arcs : on choisit d'abord un premier arc, puis un second de telle façon que si l'on échange les extrémités de ces arcs la longueur du tour soit diminuée : dans ce cas on réitère cette opération en choisissant un troisième arc de la même manière

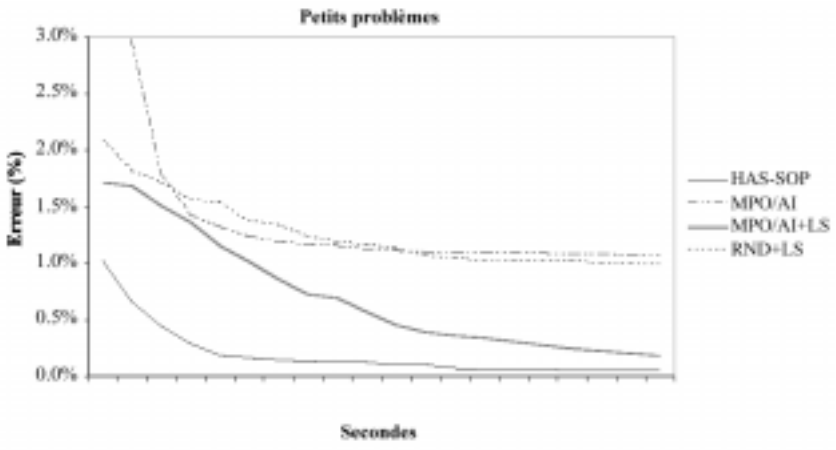


FIG. 3.4 – Résultats obtenus pour de petites instances avec HAS-SOP face à ceux d'autres algorithmes. Les graphiques représentent le pourcentage d'erreur par rapport à la meilleure solution connue en fonction du temps d'exécution. Les résultats sont les moyennes obtenues sur 5 exécutions de 120 secondes

parcours de l'arc si $c_{i,j} \geq 0$; par contre si $c_{i,j} = -1$ il s'agit d'une contrainte, indiquant que l'élément j doit être placé avant i, mais pas nécessairement immédiatement avant ce nœud.

Le POS qui est $\mathcal{NP}$-dur, modélise des problèmes réels comme le problème de routage d'un véhicule avec collecte et contraintes de livraison, la génération de plannings, etc.

GAMBARDELLA et DORIGO ont proposé en 1997 une extension à l'ACS pour le POS nommé HAS-SOP [89]. Cet algorithme diffère de ACS dans sa construction de l'ensemble des nœuds possibles qui tient compte des contraintes de précédence et utilise comme méthode de recherche locale une variante de la méthode 3-opt, modifiée spécialement pour ce problème.

Les résultats de la confrontation d'HAS-SOP avec les meilleures heuristiques sont présentés dans cet article (on trouve deux exemples de comparaison graphique fig. 3.4 et 3.5). Nous pouvons constater que ces résultats sont très bons sur les instances décrites qui sont des problèmes standards. Ceux-ci sont meilleurs que les résultats obtenus avec d'autres méthodes autant en qualité qu'en rapidité. Cet algorithme améliore même

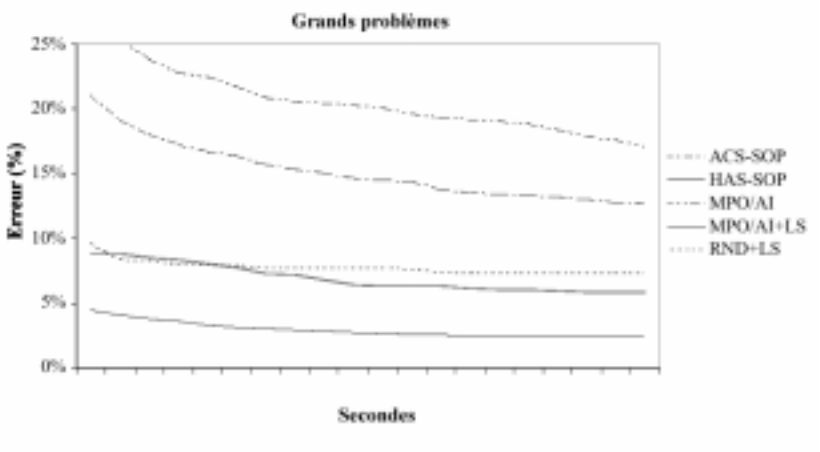


FIG. 3.5 – Résultats obtenus pour de grandes instances avec HAS-SOP face à ceux d'autres algorithmes. Les graphiques représentent le pourcentage d'erreur par rapport à la meilleure solution connue en fonction du temps d'exécution. Les résultats sont les moyennes obtenues sur 5 exécutions de 600 secondes.

nombre des meilleures solutions connues pour les instances présentées.

3.5.3 Détection de graphe hamiltonien

Le problème de détection de circuit hamiltonien dans un graphe consiste à déterminer s'il existe des circuits passant par tous les sommets d'un graphe une et une seule fois.

WAGNER et BRUCKSTEIN [231] ont proposé en 1999 un algorithme à base de fourmis pour résoudre ce problème.

Pour cette résolution, on place dans chaque sommet du graphe deux variables entières μ et τ qui stockent respectivement le nombre de passages de la fourmi et le numéro de l'itération de l'arrivée. L'algorithme VAW (Vertex Ant Walk) définit les règles de déplacement (Algo. 3.3).

Algorithme 3.3: L'algorithme VAW.
VAW()

(1) **répéter**
(2) Choisir le sommet suivant v, voisin du sommet où se trouve la fourmi (noté u) qui a les plus petites valeurs μ et τ ;
(3) Mettre à jour des valeurs μ et τ de u ;
(4) $\mu(u) \leftarrow \mu(u) + 1$ (nombre de passages) ;
(5) $\tau(u) \leftarrow t$ (temps)
(6) Augmenter le temps : $t \leftarrow t + 1$
(7) aller en v
(8) **jusqu'**au moment où le critère de fin est satisfait

Les auteurs montrent dans leur article [231], que si une fourmi trouve un cycle, elle y restera indéfiniment, mais cela ne garantit pas qu'elle le trouve s'il en existe un.

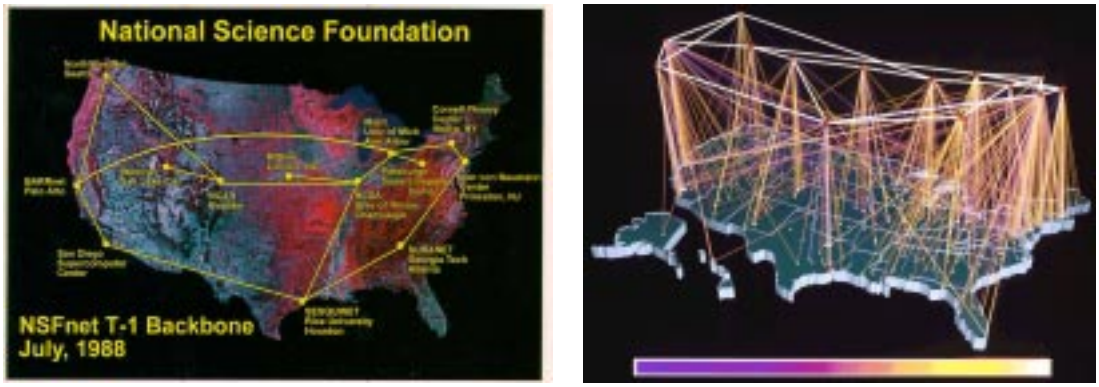


FIG. 3.6 – Schéma d'un réseau américain

Le choix du sommet suivant est déterministe, il sera sélectionné en prenant le nœud voisin qui a la plus forte valeur μ, les ex-aequo étant départagés avec la valeur de τ. Une alternative probabiliste est proposée, qui utilise une fonction de μ, mais aucun résultat ne valide cette proposition. Cette méthode a également fait l'objet d'un second article [232] où les résultats sont plus détaillés. VAW s'écarte nettement du modèle des ACO, car on ne trouve plus vraiment de traces de phéromone.

Les résultats présentés ont été obtenus pour des graphes de connexités différentes comprenant de 100 à 300 nœuds. Ils se montrent assez encourageants, sans toutefois égaler ceux des autres méthodes.

3.5.4 Le problème d'affectation quadratique

Le modèle des algorithmes à colonie de fourmis a également été appliqué à la résolution du problème d'affectation quadratique [206], auquel est consacré le chapitre 5 de ce mémoire.

3.5.5 Le problème de routage dans un réseau non commuté

Le but d'un algorithme de routage est de gérer le trafic en dirigeant les paquets de données de leur source vers leur destination, en maximisant les performances du réseau et en diminuant les coûts des échanges (le réseau peut être l'Internet).

Un réseau peut être décrit comme un ensemble de points, les routeurs, qui sont interconnectés entre eux comme nous pouvons le voir sur deux exemples de réseaux américains présentées en figure 3.6.

Trouver l'algorithme de routage optimal revient donc à résoudre un problème d'optimisation multi-objectifs dans un environnement dynamique. Les mesures des performances sont souvent de deux types : le débit (throughput) et le retard moyen des paquets (average packet delay) ; le premier mesure le nombre de transmissions disponibles en même temps, tandis que l'autre donne un temps moyen de réponse.

Les réseaux sont représentés par un graphe de N nœuds, à chaque arc sont associés la largeur de la bande passante (bandwidth) en bit/s et un délai de transmission.

Les nœuds sont composés de deux tampons, l'un en entrée qui stocke les paquets en attente de routage, l'autre en sortie où ils attendent d'être envoyés, ainsi que d'une

table de routage qui permet de déterminer vers quelle sortie le message va être orienté. Ils comportent également une table qui leur permet de choisir le meilleur aiguillage en fonction de la destination de l'information.

Le processus de routage est distribué : c'est au niveau de chaque nœud que se prennent les décisions, ce qui implique que l'algorithme tienne compte de cette architecture répartie.

Il existe deux types d'algorithmes :

- les algorithmes statiques utilisent une heuristique pour fixer une fois pour toutes l'aiguillage des paquets suivant leur point de départ et celui de destination ;
- les algorithmes dynamiques tentent d'adapter l'aiguillage des paquets aux variables du trafic.

Cette méthode implique d'ajouter des contraintes pour éviter des oscillations dans le choix de certains chemins et la formation de boucles lors de fortes charges.

Pour cet algorithme, on s'écarte du modèle général des OCF, afin de l'adapter à la dynamicité du problème.

L'algorithme AntNet a été proposé par Di Caro et Dorigo [53, 56, 55] pour résoudre ce problème. Cet algorithme utilise des agents fourmis qui sont chargés de modifier les tables de routage afin de les adapter aux fluctuations du réseau. Ces agents se décomposent en deux groupes :

- les "forward ants", dont le rôle est de faire la liaison entre les points du réseau en collectant les informations de son parcours, comme les nœuds visités et les durées de transit. Chaque nœud génère périodiquement une fourmi de ce type et l'envoie vers un site choisi aléatoirement. Au niveau de chaque nœud (fig. 3.7), la fourmi sélectionne l'arc à emprunter, suivant une distribution proportionnelle aux valeurs correspondant à sa destination dans la table de routage (traces de phéromone), ainsi qu'à l'aide d'informations liées au problème. Dans ce cas, il s'agit de l'état des mémoires tampons de sortie vers les nœuds voisins ; cette valeur est inversement proportionnelle à la longueur des informations restant à envoyer (valeur en bits). Si l'agent détecte une boucle, en repérant qu'il a déjà visité un nœud, il supprime les informations concernant ce détour et continue son chemin par une autre voie. Arrivées à destination, ces fourmis donnent naissance aux "backwards ants" et disparaissent. Les nouvelles fourmis ainsi générées ont pour rôle la modification des tables de routage qui ont été visitées.
- Les "backward ants", dont chacune est chargée de modifier les tables de routage en tenant compte des informations récoltées par la fourmi qui l'a créée. Pour cela elle va repartir vers le point de départ de sa créatrice en parcourant tous les nœuds en sens inverse. À chaque étape, elle utilisera ses connaissances pour modifier les probabilités d'aiguillage.

La priorité des agents de retour est plus forte que celle des paquets, afin de propager rapidement l'information quand la charge devient trop forte dans une partie du réseau, et ainsi d'éviter l'engorgement.

L'évaluation a été faite en comparant les résultats de cet algorithme à ceux obtenus en utilisant des méthodes classiques pour le problème de routage qui sont :

- OSPF [167], (official Internet routing algorithm), qui se réduit à définir les tables de routage par un calcul statique des plus courts chemins ;
- BF [15], algorithme asynchrone de Bellman et Ford qui utilise une mesure

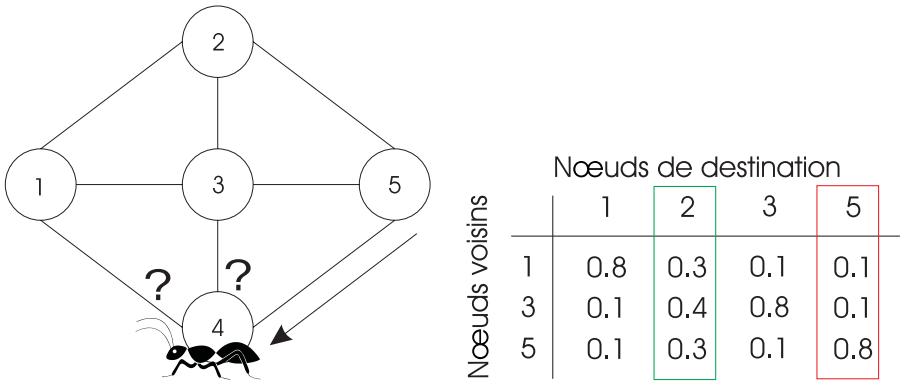


FIG. 3.7 – Exemple d'un problème de routage : Quand une fourmi arrive sur un nœud du graphe, elle va choisir le nœud suivant en se basant sur les valeurs placées dans la table de routage. Ici elle vient du nœud 5 et elle se dirige ves le nœud 2, c'est donc les valeurs de la colonne 2 qui seront utilisées et c'est le nœud 3 qui aura le plus de chance d'être choisi car il a la plus forte valeur.

dynamique des liaisons, et qui a été utilisé pour le protocole RIP (routing Infomation Protocol) de la version unix BSD [140] ;
- SPF [147, 122], prototype qui utilise une mesure dynamique pour l'évaluation des coûts des liaisons.
- SPF-1F, comparable à SPF où le "flooding" est restreint au voisinage ;
- Daemon, c'est un algorithme idéal. Cet algorithme part du principe que le démon est capable de lire l'état des files d'attente de tous les nœuds du réseau et de calculer instantanément le coût réel de tous les parcours.

Les performances obtenues avec AntNet sont comparables ou supérieures à celles des autres algorithmes réels (fig. 3.8). L'algorithme est robuste et trouve rapidement un routage stable qui a de bonnes performances. Un paramètre à considérer est celui de la fréquence de génération des agents : trop faible elle ne permet par la prise en compte rapide de l'état du réseau ; trop forte elle rend l'algorithme sensible aux fluctuations, ce qui fait osciller les tables de routage et dégrade les performances lors de charge élevée du réseau. Ce paramètre doit concilier temps de réaction et insensibilité aux micros variations. Le choix de répartir uniformément le départ des fourmis permet de mettre à jour uniformément le réseau.

Ce modèle d'agents fourmis a également été utilisé pour le routage sur des réseaux de communication commutés (réseaux téléphoniques). SCHOONDERWOERD *et al.* [193] dans leur algorithme ABC (ant-based control) utilisent un seul type d'agent qui va se déplacer entre deux points du réseau en modifiant les probabilités associées à son point de départ dans les tables de routage des nœuds qu'il traverse. BONABEAU *et al.* [18] utilisent un principe similaire à la différence près qu'à chaque nœud les probabilités associées aux nœuds déjà traversés sont modifiées.

3.5.6 Le problème d'optimisation numérique

Dans tous les algorithmes qui ont été présentés dans ce chapitre, les fourmis devaient résoudre des problèmes dans un espace discret. Le passage à la résolution de problèmes dans l'espace continu des fonctions numériques a nécessité le développement

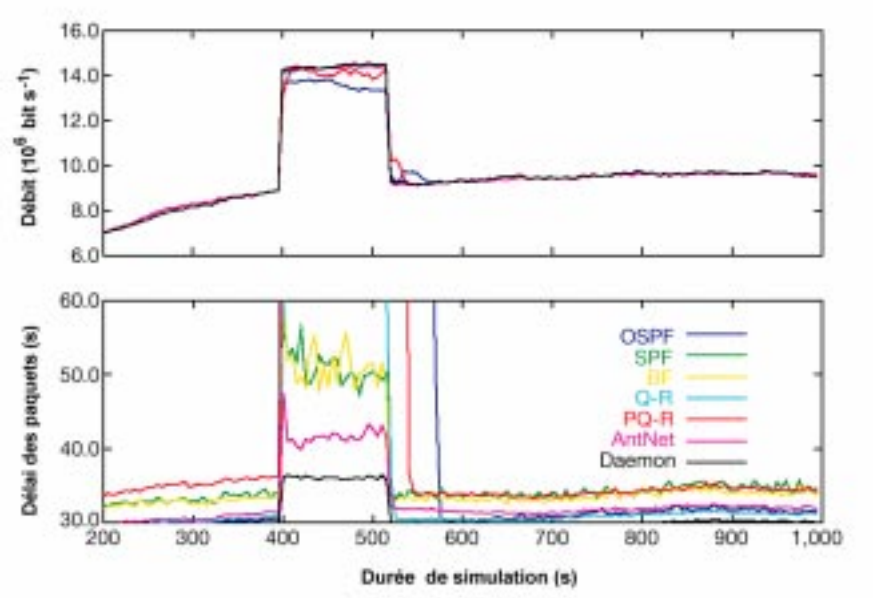


FIG. 3.8 – Résultats obtenus avec AntNet : Pour la comparaison, on simule la saturation d'un réseau pendant 120 secondes, 400 secondes après le début du test. Le graphique du haut donne le débit (le plus large est le meilleur) tandis que le graphique du bas donne le retard moyen dépassant les 5 secondes (le plus bas est le meilleur). AntNet est compétitif avec les meilleurs algorithmes pour le débit et obtient les meilleures performances pour le retard moyen (seul deamon est meilleur, mais il s'agit d'un modèle théorique)

de nouvelles stratégies.

BILCHEV et PARMEE [16] ont développé un système de calcul dynamique qui cherche à minimiser une fonction f à n variables. Cette méthode présentée dans l'algorithme 3.4 s'inspire du fourragement des fourmis. Les fourmis partent du nid et se dirigent vers les emplacements où se trouvent les sources de nourriture. Le parcours de chaque fourmi peut être résumé sous la forme d'un vecteur (fig. 3.9 et fig.3.10) dont les extrémités sont le nid et la source de nourriture. À un instant donné, le nombre de ces chemins est limité. Ce concept s'adapte bien au monde continu, à la différence près qu'il faut trouver une méthode pour placer le nid sur une zone de l'espace de recherche qui semble prometteuse : la méthode proposée est de type algorithme génétique.

Lors de l'exécution, les fourmis quittent le nid, choisissent l'un des vecteurs et

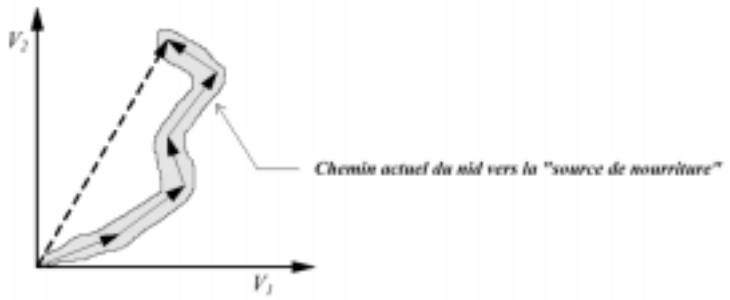


FIG. 3.9 – Le déplacement d'une fourmi peut être mis sous la forme d'un vecteur.

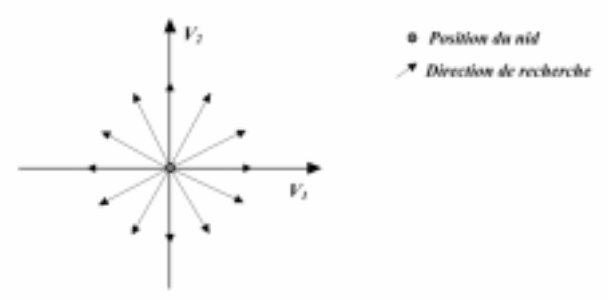


FIG. 3.10 – Les départs du nid : Le nombre de vecteurs (déplacement des fourmis) partant du nid doit être fixé

arrivent à des positions plus ou moins éloignées du nid. Arrivées à destination, elles vont pouvoir explorer les environs, dans les limites fixées par les rayons associés à chaque vecteur. Les vecteurs ne sont pas fixes mais peuvent être modifiés suivant la position des bonnes solutions trouvées. De nouveaux vecteurs peuvent être créés, qui sont le fruit du croisement (comme dans les algorithmes génétiques) de deux vecteurs existants, afin de créer une nouvelle zone de recherche. Le choix du vecteur par une fourmi se fait suivant une probabilité proportionnelle aux taux de phéromone qui y est associé. La modification de ces taux est réalisée proportionnellement à la qualité des solutions trouvées pendant l'exploration.

Algorithme 3.4: L'algorithme d'optimisation numérique.
ACM()

(1) $t \leftarrow 0$
(2) Initialiser $A(t)$ (structure qui représente le nid et son voisinage)
(3) Evaluer $A(t)$
(4) Evaporer $A(t)$
(5) **tant que** la condition d'arrêt est fausse
(6) **faire**
(7) $\quad t \leftarrow t + 1$
(8) $\quad$ Déposer phéromone $A(t)$
(9) $\quad$ Déplacer fourmi $A(t)$
(10) $\quad$ Evaluer $A(t)$
(11) $\quad$ Evaporer $A(t)$
(12) **fin tant que**
(13) **retourner** la meilleur solution trouvée

WODRICH [239] a proposé une heuristique similaire dans laquelle une seconde variété de fourmis remplace l'algorithme génétique pour définir l'emplacement du nid. Ces fourmis sont désignées sous le terme d'agents de recherche globale. Il ajoute également une notion d'âge pour les vecteurs. Cette mesure indique depuis quand ils n'ont pas permis d'améliorer la meilleure solution, les vecteurs trop âgés étant alors abandonnés. MONMARCHÉ, VENTURINI et SLIMANE ont également proposé une méthode apparentée [166, 159] qui intégre :

- un mécanisme de recrutement, permettant, aux fourmis explorant une bonne zone de l'espace des solutions, d'attirer d'autres fourmis sur cette zone ;
- l'hétérogénéité de la population, où les différentes fourmis n'ont pas le même paramètrage ;
- de multiples sites de chasse, qui représentent différentes zones explorées.

L'heuristique API qu'ils proposent s'inspire directement du modèle de fourragement des fourmis Pachycondyla apicalis [82].

3.5.7 Un aperçu des différents domaines d'application des OCF

Les OCF ont été appliquées à de nombreux problèmes d'optimisation. Dans le tableau 3.9, on trouve les principaux algorithmes utilisant ce paradigme qui ont été proposés pour chaque problème.

3.6 La fourmi artificielle et la programmation automatique

Nous avons utilisé le modèle de coopération des fourmis à base de la stigmergie pour concevoir une application de génération automatique de programmes. Cet outil est appelé Ant Programming (AP). Il génère un ensemble de programmes en utilisant la phéromone comme guide. Son objectif est de voir apparaître parmi ces programmes, la solution qui correspondra aux spécifications du problème.

Nous décrirons plus amplement cet algorithme dans la troisième partie où nous le situerons parmi les méthodes existantes pour la génération automatique de programmes.

3.7 Conclusion

Dans la dernière décennie, les informaticiens se sont interessés aux fourmis et à leur inter-coopération. L'engouement qui apparut peu après dans le grand public, renforcé par les romans de Werber [233] et par deux films d'animation, a contribué à la publication dans des revues "grand public" d'articles sur l'informatique inspirée des fourmis, par exemple dans le *New York Times* du 13 septembre 2001, le *Time* "Europe specials" de mars 2001, *Info Science* du 23 octobre 2000, *Scientific American* de mai 2000, *Le Monde* du 10 mai 2000, *New Scientist* du 24 janvier 1998, ...

Ces dernières années ont également vu paraître plusieurs publications, qui font le point sur les applications utilisant le modèle de coopération des fourmis en optimisation, ou encore le tour des différentes propositions pour un problème donné, c'est notamment le cas du PVC [207] ou du PAQ [206].

Plusieurs ouvrages sont également parus : *New Idea in Optimization* [41] replace les algorithmes à base de colonies de fourmis parmi les méthodes d'optimisation récentes, tandis que *Swarm Intelligence : From Natural to Artificial Systems* [17] situe ces heuristiques vis à vis de leur modèle naturel, et fait le point sur les différents domaines qui ont vu des applications de ce modèle. Il est à noter aussi la publication d'un numéro spécial de *FGCS* en 2000 [61], et les travaux sur cette thématique, consacrés aux fourmis faisant suite à *ANTS'2000* [59].

TAB. 3.9 – Algorithmes utilisant le modèle des OCF pour différents problèmes d'optimisation combinatoire

Nom du problème	Auteurs	Année	Nom de l'algorithmes	Références principales
Voyageur de commerce	Dorigo, Maniezzo & Colorni	1991	AS	[58, 66, 67]
	Gambardella & Dorigo	1995	Ant-Q	[87, 63]
	Dorigo & Gambardella	1996	ACS & ACS-3-opt	[64, 65, 88]
	Stützle & Hoos	1997	$\mathcal{MM}$AS	[209, 210, 212]
	Bullnheimer, Hartl & Strauss	1997	AS_{rank}	[23, 26]
	Cordón, *et al.*	2000	BWAS	[40]
Affectation quadratique	Maniezzo, Colorni & Dorigo	1994	AS-QAP	[145]
	Gambardella, Taillard & Dorigo	1997	HAS-QAP	[90, 86]
	Stützle & Hoos	1997	$\mathcal{MM}$AS-QAP	[208]
	Maniezzo	1998	ANTS-QAP	[141]
	Roux, Fonlupt, Robilliard, Talbi	1998	ANTabu	[180, 182, 224]
	Maniezzo & Colorni	1999	AS-QAP	[144]
Affectation généralisée	Ramalhinho, Lourenco & Serra	1998	$\mathcal{MM}$AS-GAP	[175]
Routage réseau (commuté et non commuté)	Schoonderwoerd *et al*	1996	ABC	[193, 194]
	Di Caro & Dorigo	1997	AntNet & AntNet-FA	[53, 56, 55, 54, 57]
	Subramanian, Druschel & Chen	1997	Regular ants	[213]
	White, Pagurek & Oppacher	1998	ASGA	[235, 234]
	Bonabeau *et al.*	1998	ABC-smart ants	[18]
	Heusse *et al.*	1998	CAF	[114]
	Van Der Put & Rothkrantz	1998	ABC-backward	[227, 228]
Routage pour réseaux optiques	Navarro & Smith	1999	ACO-RAP	[169]
Ordonnancement	Colorni, Dorigo, Maniezzo & Trubian	1994	AS-JSP	[37]
	Stützle	1998	AS-FSP	[204]
	Bauer *et al.*	1999	ACS-SMTTP	[13, 12]
	Den Besten, Stützle & Dorigo	1999	ACS-SMTWTP	[13]
	Merkle, Middendorf & Schmeck	2000	ACO-RCPS	[149]
Routage de véhicules	Bullnheimer, Hartl & Strauss	1997	AS-VRP	[22, 25, 24, 21]
	Gambardella, Taillard & Agazzi	1999	HAS-VRP	[85]
Coloriage de graphe	Costa & Hertz	1997	ANTCOL	[43]
Optimisation numérique	Bilchev & Parmee	1995	ACM	[16]
	Monmarche, Venturini & Slimane	1998	API	[166, 159]
	Wodrich	1996	Ants'way	[239]
Plus courte superséquence commune	Michel & Middendorf	1998	AS-SCS	[154, 155]
Sac à dos multiple	Leguizamón & Michalewicz	1999	AS-MKP	[135]
Affectation de fréquence	Maniezzo & Carbonaro	1998	ANTS-FAP	[142, 143]
Ordonnancement séquenciel	Gambardella & Dorigo	1997	HAS-SOP	[89, 84]
Satisfaction de contraintes	Solnon	2000	Ant-P-solver	[200, 199]
Allocation redondante	Liang & Smith	1999	ACO-RAP	[137]
Reconnaissance de graphes hamiltoniens	Wagner & Bruckstein	1999	VAW-H	[231]
recherche dans des graphes dynamiques	Wagner & al	2000	VAW-DG	[232]
Classification non supervisée	Monmarche, Sliman & Venturini	1999	AntClass	[165, 164, 159]

Deux conférences sur les fourmis ont déjà eu lieu à Bruxelles sous la tutelle de DORIGO en 1998 et 2000. D'autres manifestations accueillent les travaux sur ce domaine, principalement sur les thèmes des algorithmes évolutionnaires (*PPSN* [146, 230, 69, 191], *International Conference on Machine Learning* [173]), de la vie artificielle (*European conference on Artificial Life* [229]) et de la simulation des comportements adaptatifs des animaux (*International Conference on Simulation of Adaptive Behavior* [151, 34, 172]).

L'utilisation de la stigmergie a montré son efficacité en informatique, comme l'indiquent les différents algorithmes que nous avons abordés. Cette méthode peut encore être appliquée à de nouveaux problèmes (c'est le cas de la programmation automatique que nous verrons dans la troisième partie de ce mémoire), ou être améliorées par une hybridation qui sera le sujet de notre seconde partie.

Les fourmis ne sont pas les seules à fournir un modèle de coopération : c'est également le cas des abeilles ou des termites. On peut donc qualifier ces algorithmes de méthodes à intelligences d'essaim (Swam Intelligence) ou à intelligence collective [19, 17].

Il est maintenant intéressant de replacer ces algorithmes de "fourmis" face aux autres méthodes de recherche opérationnelle, c'est le sujet du chapitre suivant.

Chapitre 4

Les fourmis face aux autres méthodes

Ce troisième chapitre, sera l'occasion de présenter les différentes heuristiques qui reposent sur des modèles naturels. Toutes ces méthodes, comme d'ailleurs beaucoup d'autres, utilisent une forme ou une autre de mémoire. Pour les caractériser nous proposons une extension de la taxinomie des programmes à mémoire adaptative (AMP) proposée par Taillard. Dans la classification que nous présentons c'est la mémoire qui est le facteur discriminant entre ces différentes heuristiques. Cet outil sans être parfait donne une base pour la description de ces différentes méthodes.

4.1 Introduction

Il est toujours très délicat de vouloir comparer une méta-heuristique comme les OCF à d'autres méta-heuristiques tant les paramètres, les opérateurs de recherche ou encore la forme et le type de mémoire utilisée peuvent être différents. Notre ambition pour ce chapitre est de brosser une représentation rapide des principales méta-heuristiques qui sont utilisées pour les comparaisons.

Afin de proposer une comparaison, entre ces différentes méthodes, nous avons affiné la taxonomie proposée par Taillard. Cette taxonomie nous permet de voir que certaines heuristiques comme PBIL ont par certains aspects de profondes caractéristiques communes avec les OCF.

4.2 Les autres méthodes "naturelles" disponibles

Les dernières décennies ont vu l'émergence de nombreux algorithmes inspirés de la modélisation de l'évolution naturelle. Ces méthodes offrent de bonnes performances pour résoudre des problèmes d'optimisation. La plus grande famille d'heuristiques s'inspire du modèle des algorithmes génétiques que nous aborderons dans la première sous-section ; les autres méthodes existantes feront l'objet de la sous-section suivante.

4.2.1 Les algorithmes génétiques

Ces méthodes utilisent les concepts de la génétique sur lesquels est appliqué le modèle d'évolution proposé par DARWIN. Elles utilisent des populations qui vont évoluer pour s'adapter à leur milieu. En informatique, les individus constituant la population seront les solutions, l'environnement sera le problème à résoudre et l'évolution devrait donner la solution qui résout au mieux le problème.

DARWIN définit dans son ouvrage *De l'origine des espèces* [46], le mécanisme de l'évolution comme reposant sur le hasard et sur la sélection naturelle. Selon cette théorie, le processus ne peut pas produire des modifications importantes ou subites, les variations étant successives, légères et lentes. Cette sélection repose sur le fait que les individus les mieux adaptés à leur environnement ont tendance à devenir prédominants, tandis que les moins adaptés disparaissent.

"Chaque organisme vivant résulte des interactions entre un programme spécifique, qui définit à la fois ses potentialités et ses limites biologiques, et un environnement capable d'induire et de moduler l'expression de ces potentialités" (GALLIEN, [83]) C'est de cette façon que l'on pourrait définir le patrimoine génétique d'un individu et son rôle dans la vie de cet organisme.

Avant de poursuivre, il est nécessaire de préciser quelques termes de vocabulaire :
- **phénotype :** c'est l'ensemble des caractères apparents permettant de reconnaître un individu ;
- **génotype :** c'est l'ensemble de l'information génétique d'un individu ou génome ;
- **gène :** il s'agit d'une information qui est placée sur un chromosome à un emplacement donné appelé "locus" ;
- **mutation :** il s'agit du mécanisme spontané qui altère de façon aléatoire un gène ;
- **recombinaison ou reproduction sexuée :** le premier terme sera utilisé de préférence, car le second peut apporter une confusion entre la notion de reproduction et celle de réplication. Ce mécanisme est celui de la reproduction.

Les lois de MENDEL [148] définissent la transmission des caractères (hérédité) à travers les générations dans le cadre de la reproduction sexuée. Lors de la reproduction de deux êtres (ici, nous ne nous intéressons qu'à ce type de reproduction), un nouvel organisme va être créé, son patrimoine génétique sera constitué des gènes de ses deux parents ; c'est la recombinaison qui permet aux deux parents, dans le cas le plus fréquent, de donner la moitié de leur patrimoine à leur enfant.

En se basant sur ces principes, HOLLAND en 1975 [115] proposa le modèle des algorithmes génétiques. D'autre chercheurs ont étendu ce modèle comme DE JONG [49], GOLDBERG [101], DAVIS [47] et MICHALEWICZ [152]. Pour HOLLAND [115] le codage binaire est le meilleur codage possible pour les chromosomes. Selon lui, plus une représentation est détaillée et plus de similarités entre les chaînes pourront être trouvées. Les solutions sont représentées sous la forme d'ensembles de valeurs. Ces groupes de valeurs sont concaténés sous leur forme binaire en structures pour former les chromosomes.

Plus récemment, d'autres types des représentation sont apparus pour pallier les limitations du codage binaire, trop limitatif ou trop complexe à mettre en œuvre. C'est GOLDBERG qui a introduit l'utilisation d'autres éléments dans les chromosomes (entier,

réel) [102]. Dans le cas du voyageur de commerce, Grefenstette [107] ou Lin [138] utilisent une liste de villes comme chromosome, pour Khuri [185], Reeves [176] ou Falkenauer [73], le chromosome est formé par une liste d'objets à manipuler pour le bin-packing. Dans le cas de la robotique, un codage sous la forme d'un angle et d'un vecteur définissent les déplacements (*cf.* Talbi *et al.* [221]). Dans d'autre cas, il peut s'agir de deux listes pour le placement de composants (*cf.* Cohoon [35], d'une matrice pour Caux [32] ou Schoenauer [190]).

Pour permettre l'évolution, une mesure d'adaptation de la solution au problème est introduite : on parle de fonction "qualité" ou "fitness". Cette mesure permet de trouver les meilleurs individus, ceux qui, selon Darwin, devraient donner les meilleurs enfants. La fonction qualité peut être définie comme suit :

- elle dépend des critères que l'on veut maximiser ou minimiser. Dans le cas de problème de recherche opérationnelle ce sera la distance pour le PVC, le coût des échange pour le PAQ, etc ;
- elle est une boîte noire dont l'entrée est le phénotype et la sortie la valeur de la qualité ;
- elle peut changer de façon dynamique pendant le processus de recherche ;
- elle peut être si compliquée qu'on ne peut calculer que sa valeur approchée, comme dans le cas de la conception d'avion [20] ;
- elle devrait attribuer des valeurs très différentes aux individus afin de faciliter la sélection ;
- elle doit considérer les contraintes du problème. S'il peut apparaître des solutions invalides, la fonction de qualité doit pouvoir attribuer une valeur proportionnelle à la violation des contraintes comme pour le bin-packing [177] ;
- l'environnement peut présenter du bruit dans les évaluations (partielles), c'est le cas pour l'interprétation d'image (comme dans [139]) ;
- la valeur de la fonction de qualité peut être aussi attribuée par l'utilisateur (par exemple, la valeur esthétique [108, 161, 160, 162]).

Le fonctionnement d'un AG est présenté figure 4.1, sa décomposition est la suivante :

- une population d'individus est générée, il s'agit de générer aléatoirement un ensemble de chromosomes ;
- de cette population sera extrait un sous-groupe dont les individus seront appelés "parents" et seront amenés à générer de nouveaux individus ("enfants") par le mécanisme de recombinaison ;
- lors de cette génération peut également être appliquée une mutation qui apportera de nouveaux gènes dans les chromosomes des enfants ;
- un mécanisme de sélection est utilisé pour déterminer, à partir de tous les individus anciens et nouveaux, ceux qui formeront la population de la génération suivante :
  - ⋆ soit la stratégie élitiste où le meilleur individu n'est jamais éliminé de la population ;
  - ⋆ soit une stratégie de renouvellement de la population où tous les individus de l'ancienne population sont remplacés par ceux nouvellement créés ;
  - ⋆ soit une stratégie "un seul à la fois" où après la reproduction les nouveaux individus sont ajoutés à la population existante, plusieurs générations cohabi-

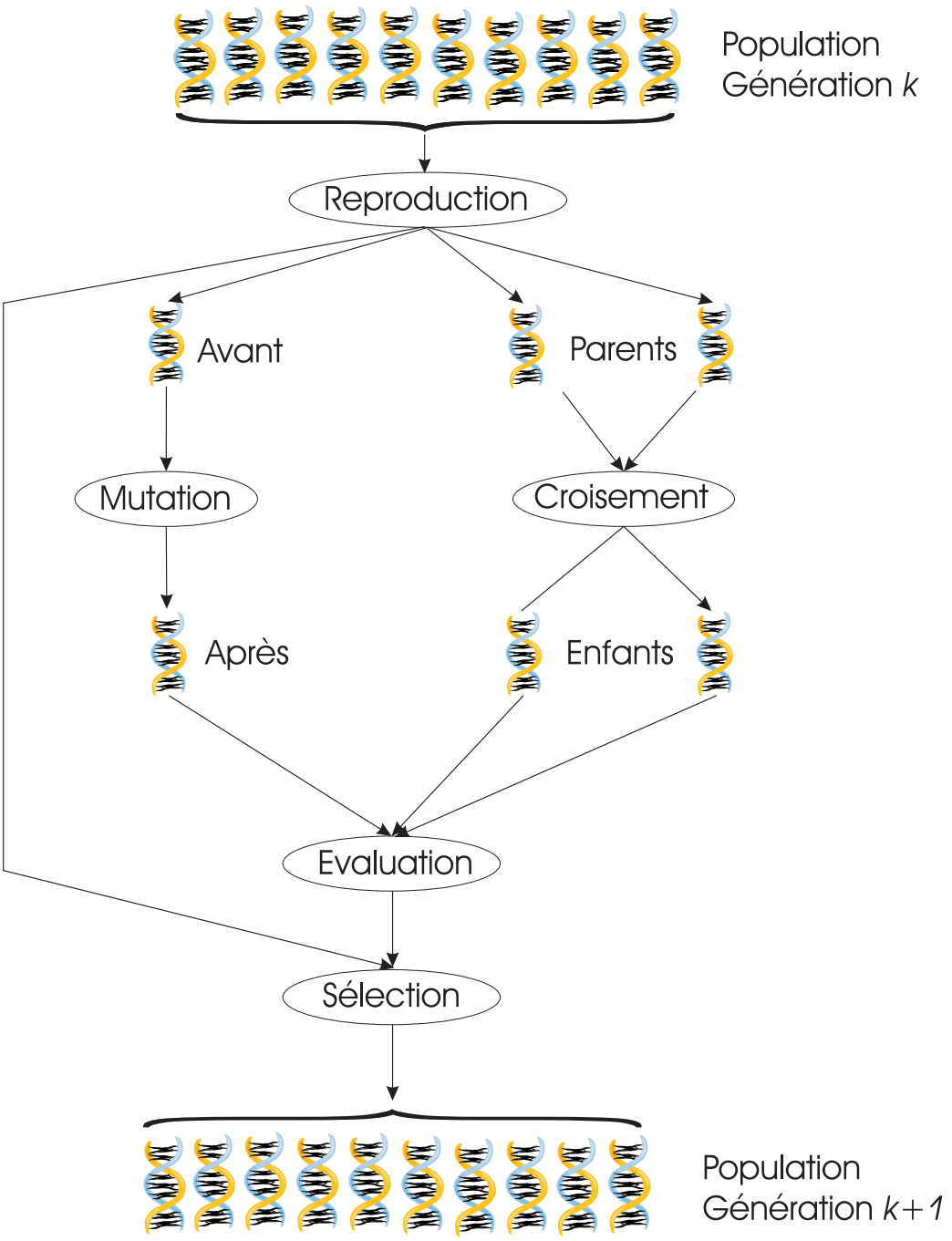


FIG. 4.1 – Schéma général de fonctionnement d'un AG

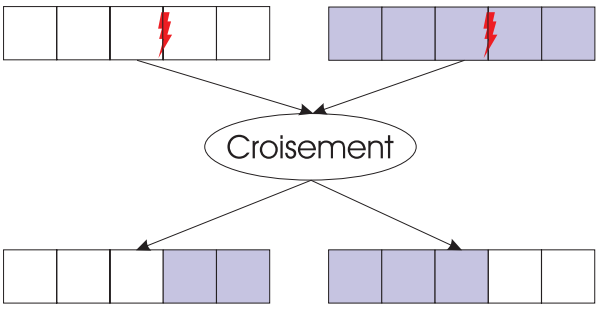


FIG. 4.2 – La recombinaison un point : Les deux chromosomes vont être coupés au même endroit, les éléments ainsi formés vont être échangés pour produire de nouveaux individus

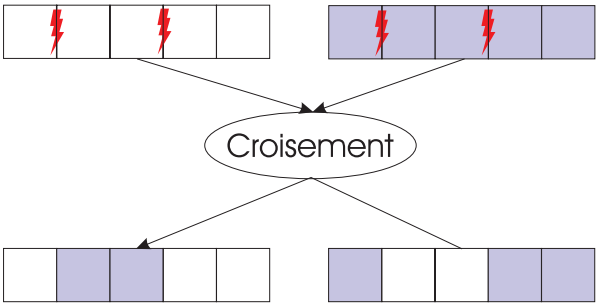


FIG. 4.3 – La recombinaison deux points : Les deux chromosomes sont coupés en trois segments, le segment du milieu est alors échangé pour obtenir les nouveaux individus.

tant. Une solution pour maintenir la taille de la population est de retirer les solutions les plus âgées [152, 33].

– le mécanisme d'évolution sera appliqué sur les populations successives tant qu'un critère d'arrêt ne sera pas rempli.

Parmi les méthodes de recombinaison généralement utilisées se trouve le crossover un-point (fig. 4.2). Celui-ci consiste à choisir aléatoirement un site de croisement (il s'agit d'une valeur comprise entre 1 et $l-1$, où l est la taille du chromosome), et à échanger entre les parents la partie de leur chromosome se trouvant sous le locus sélectionné : il génère donc deux nouveaux individus. D'autres types de crossover existent avec un nombre de sites qui peut être de deux (fig. 4.3), trois, voire plus, ou encore le crossover uniforme [214]. Il est à noter que pour les chromosomes non binaires le processus de recombinaison doit être adapté pour respecter les contraintes liées au problèmes : c'est le cas pour le voyageur de commerce [73, 103, 176]. Certains opérateurs forment des chromosomes non valides qu'il faut alors réparer [152].

Il existe énormément de mécanismes de sélection et de recombinaison, mais également de nombreux paramétrages des différentes valeurs qui contrôlent les algorithmes génétiques. Il serait trop long ici de citer l'ensemble de ces mécanismes ; on peut cependant dans un but d'illustration citer certains réglages qui ont été choisis de façon plutôt empirique : c'est le cas de la règle des 1/5 [119] pour la mutation, de la probabilité de 60% [49] ou de 95% [106] pour la recombinaison. Pour améliorer les performances des AG, de nombreux opérateurs sont proposés pour la recombinaison et la mutation : on trouve la mutation auto-adaptative [195], le point de croisement auto-adaptatif [188] et la recombinaison adaptative dans les algorithmes evolutifs [201].

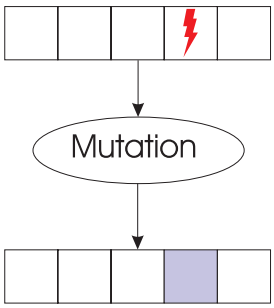


FIG. 4.4 – La mutation : un géne du chromosome est sélectionné, il est alors remplacé par un nouveau choisi aléatoirement

Les enfants pourront également subir une mutation (ce qui correspond à un mécanisme d'exploration).

Dans le cas le plus simple (fig. 4.4), elle sera appliquée avec une probabilité, généralement faible, le choix de cette probabilité dépendant de la taille de la population et de la taille des chromosomes. Cette valeur peut varier afin d'adapter son importance au cours du temps [70].

Il apparaît cependant en conclusion que le mécanisme de l'algorithme génétique canonique (codage binaire, opérateurs de recombinaison et de mutation aveugles) ne permet pas de résoudre des problèmes extrêmement complexes. Les travaux récents [192] montrent que l'ajout d'informations structurelles sur les problèmes à résoudre améliorent de manière significative les performances de l'AG. Cet apport d'informations peut se réaliser par exemple par la création d'opérateurs spécifiques.

Le schéma d'un algorithme génétique simple peut donc se décrire sous la forme de l'algorithme 4.1.

Algorithme 4.1: Algorithme simplifié d'un Algorithme génétique.
AG()

(1) Génération aléatoire d'une population initiale
(2) **Répéter**
(3) Sélection proportionnelle
(4) Appariement
(5) Recombinaison
(6) Mutation
(7) Remplacement générationnel
(8) **Jusqu'**à ce qu'un critère d'arrêt soit vérifié

Notre but n'est pas de faire une présentation exhaustive des algorithmes génétiques mais d'en réaliser un rapide tour d'horizon ; il est possible de trouver de nombreux ouvrages qui traitent très largement du sujet :

- *Genetic algorithms in search, optimization & Machine learning* de GOLDBERG [102]
- *Handbook of genetic algorithms* de DAVIS [48]
- *An introduction to genetic algorithms* de MITCHELL [156]

– *Genetic Algorithms + Data Structures = Evolution Programs* de Michalewicz [153]

4.2.2 Les autres méthodes guidées

D'autres méthodes utilisent des modèles plus ou moins proches de la nature comme fil directeur. La recherche par dispersion est pour sa part très proche des AGs, sans vraiment s'inspirer de la génétique ; la recherche tabou s'apparente au raisonnement humain, et le recuit simulé s'inspire d'un procédé de fonderie. C'est ce que nous détaillerons dans les sections suivantes.

La recherche par dispersion

La recherche par dispersion ou "scatter search" a été introduite par Glover en 1977 [94]. Son principe est similaire à celui des algorithmes génétiques, ce qui est particulièrement visible dans ses implantations récentes. Néanmoins, on a pu remarquer [218] que dans les années qui suivirent cette publication, peu d'articles utilisèrent cette méthode. Cette heuristique fut redécouverte [99, 100, 31] au début des années 90 . La recherche par dispersion est basée sur une population, à l'instar des algorithmes génétiques, bien que, dans ce cas, la taille de la population soit plus petite.

Dans cette méthode, les individus sont des vecteurs d'entiers qui vont évoluer dans le temps grâce à :

– un opérateur de sélection ;
– un mécanisme de recombinaison linéaire de solutions : il a pour but de créer de nouvelles solutions provisoires, celles-ci n'étant pas forcément valides ou acceptables ;
– un opérateur de projection, qui va compléter les solutions temporaires pour les rendre admissibles ;
– un opérateur d'élimination.

Il est donc possible de définir la recherche par dispersion comme un algorithme génétique où les différents éléments ont été modifiés de la façon suivante :

– les individus : les structures binaires sont remplacées par des structures d'entiers ;
– l'appariement : il peut sélectionner plus de deux individus ;
– la recombinaison : il s'agit d'une recombinaison linéaire de vecteurs ;
– la mutation : il s'agit de la projection ; son rôle est de transformer des solutions temporaires obtenues après recombinaison afin de les rendre admissibles.

Récemment, cette méthode a été utilisée pour l'élaboration de tournées [179], pour l'optimisation continue sans contrainte [76], pour la phase d'apprentissage des réseaux de neurones [121], pour le problème d'affectation quadratique [45] et pour l'affectation multi-objectifs [131]. Cette méthode fournit des solutions de très bonne qualité.

Le recuit simulé

La méthode du recuit simulé (*simulated annealing*) est une technique d'optimisation combinatoire dérivée de la méthode de Monte-Carlo qui est utilisée pour la simulation de systèmes thermodynamiques. Métropolis *et al.*, en 1953 [150] ont proposé un modèle

de simulation de l'évolution d'un solide vers son équilibre thermique. En 1982, KIRPATRICK, GELATT et VECCHI (chercheurs chez IBM) [123] montrèrent la similitude entre la recherche de l'équilibre thermique d'un solide et la minimisation d'un critère en optimisation.

Cette méthode repose sur un modèle métallurgique appelé processus de recuit. Lorsque l'on chauffe un métal à haute température, il se liquéfie et peut épouser n'importe quelle forme. Quand la température décroît, le métal prend une forme qui sera de plus en plus difficile à modifier (il est refroidi). Cependant si on désire refaçonner le métal il est nécessaire de le réchauffer (recuit). Lors du refroidissement de l'alliage on choisit la vitesse de refroidissement du solide en fonction des qualités mécaniques voulues. Par exemple, l'acier des objets tranchants est refroidi très rapidement par trempage ce qui donne des cristaux très imparfaits mais très durs. Si on souhaite avoir un métal malléable et peu cassant, il faut que l'alliage ait une structure cristalline la plus parfaite possible, ce qui nécessite des opérations intermédiaires de recuit de manière à éliminer tout défaut du cristal.

En informatique, l'objectif du recuit simulé est de trouver une solution à un problème donné qui minimise sa fonction objectif. Partant d'une solution S donnée qui a une qualité C, on modifie cette configuration aléatoirement pour donner la nouvelle solution S' de coût C'. Cette étape est appelée un mouvement. La nouvelle solution sera acceptée ou refusée suivant la probabilité de Métropolis [150], cette fonction étant inspirée de celle de changement d'état de Boltzmann. Les solutions sont toujours acceptées si elles sont meilleures ; dans le cas contraire c'est une probabilité fonction de la qualité et de la "température" qui permettra de dire si elle sera conservée ou non. Le critère d'acceptation probabiliste des nouvelles solutions est basé sur la variation de la qualité et d'un paramètre de contrôle. Il autorise l'acceptation des solutions moins bonnes, ce qui permet à l'algorithme de sortir de "pièges" constitués par des optima locaux.

Ce procédé peut être formalisé comme un mécanisme statistique dont la probabilité $P(X)$ de visiter l'état X en fonction de son énergie $E(X)$ et de la température T est :

$$P(X) = e^{-\frac{E(X)}{kT}}$$

k est une constante. La méthode suit le schéma de l'algorithme 4.2.

Algorithme 4.2: Algorithme simplifié du recuit simulé.
RECUIT SIMULÉ()

(1) Calculer une solution initiale $k = 0$; et initialiser la température T_0
(2) Modifier la solution courante à l'aide d'une transformation simple.
(3) Calculer son coût Δf
(4) Si $\Delta f > 0$ alors tirer un nombre aléatoire p entre 0 et 1. Si $p > exp(-\Delta f / T_k)$ aller en (7) (la transformation est rejetée).
(5) Adopter la nouvelle solution comme solution courante
(6) Si "l'équilibre statistique est atteint" alors incrémenter k et "abaisser" la température : $T_k = g(T_{k-1})$
(7) Si le système n'est pas "gelé" aller en (2)
(8) FIN

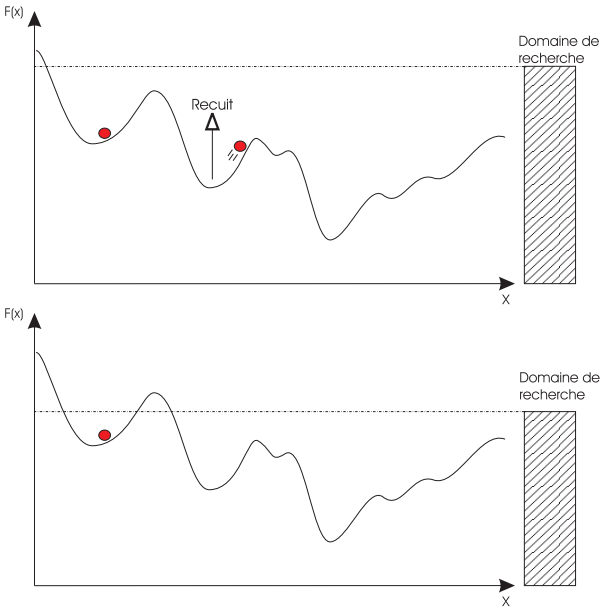


FIG. 4.5 – L'influence de la température pour le recuit : dans le premier schéma la température est élevée (large domaine de recherche), ce qui permet à l'algorithme de facilement sortir d'une vallée contenant un optimum local pour se diriger vers de meilleurs sites. Dans le second cas, la température est plus faible (espace de recherche plus restreint) ce qui bloque la méthode au niveau de cet optimum local.

Au début de l'algorithme, la "température" est élevée, la probabilité est donc proche de 1, ce qui implique que presque toutes les variations de la solution peuvent être acceptées. Au contraire, au fur et à mesure de la diminution de la température, les remontées sont de plus en plus difficiles et seules les faibles détériorations de la valeur de la qualité seront acceptées. Si une solution est rejetée, l'algorithme va tenter d'en choisir une nouvelle. Si aucune ne remplit les conditions requises, la première solution testée est tout de même sélectionnée et la recherche continue sur cette base.

La valeur de température est un paramètre de contrôle de l'heuristique (fig 4.5). La température doit être initialisée à une valeur suffisamment élevée pour permettre de passer les barrières et suffisamment basse pour permettre une exploration des bassins prometteurs. La règle de diminution de la température est importante pour l'algorithme, car elle doit laisser le temps d'explorer le maximum de sites afin de découvrir le plus prometteur (celui qui contient l'optimum global). Le choix de la température initiale permet de fixer le nombre de configurations qui pourront être acceptées. Si cette valeur est trop élevée, toutes les valeurs seront acceptées, ce qui allonge beaucoup le temps de recherche ; par contre si elle est trop basse, l'algorithme risque de rester bloqué au niveau d'un optimum local. Il ne faut pas négliger un critère important qui est la taille des paliers de température. Pour l'heuristique, il s'agit du nombre de configurations à évaluer avant un changement de température. AARTS et VAN LAARHOVEN [1] suggèrent qu'il soit égal au nombre des voisins de la solution courante. D'autres méthodes plus ou moins élaborées ont été proposées par KIRKPATRICK *et al.* [123] ; elles sont toutefois assez difficiles à mettre en œuvre.

On emploie fréquemment cette méthode dans les milieux industriels pour résoudre

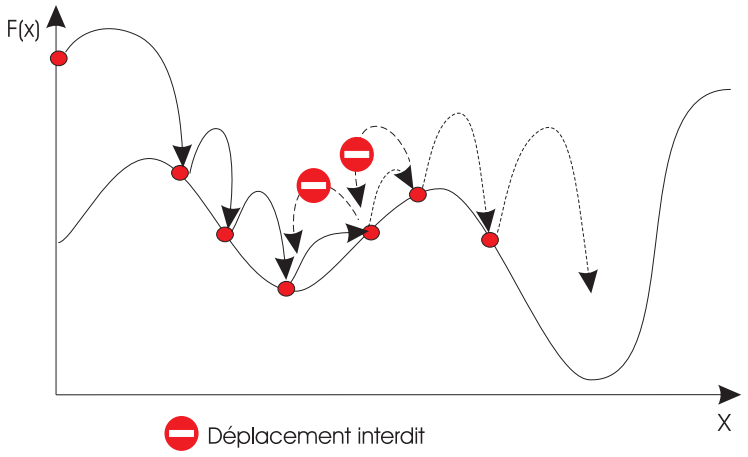


FIG. 4.6 – La recherche tabou : lors du passage de voisin en voisin les retours en arrière et les boucles sont interdits par l'utilisation de la liste tabou.

des problèmes :
- d'optimisation combinatoire (ordonnancement) ;
- de CAO (conception de circuits, placement de composants) ;
- de traitements d'image (restitution d'images brouillées).

La recherche tabou

Cette méthode a été proposée par GLOVER en 1986 [95]. La recherche tabou permet de construire de bonnes ou de très bonnes solutions à des problèmes d'optimisation combinatoire de complexité industrielle comme le montre l'article de GLOVER de 1995 [98] comprenant déjà plus de soixante-dix applications différentes.

L'idée de base de la recherche tabou (fig. 4.6) est l'utilisation d'une liste dans laquelle les mouvements qui ont déjà été effectués seront stockés. Cette liste permet lors de la sélection d'un voisin d'éviter de choisir la solution précédente ou encore de retourner (boucle) en repassant par une solution qui a déjà été visitée, donc d'échapper aux optima locaux. Le nom de “liste tabou” donné par GLOVER vient de l'interdiction de revisiter des solutions récemment visitées. La méthode tabou peut contenir une ou plusieurs listes tabou, selon les besoins. La liste tabou peut être de taille fixe ou le plus souvent de taille variable [217]. La taille de la liste tabou est un facteur important : trop petite, elle ne permettra pas à l'algorithme de s'échapper de certaines zones d'attraction d'optima locaux ; trop grande, la qualité des solutions obtenues sera affectée, le voisinage en chaque point étant très clairsemé [198]. Dans les premières implémentations, les chercheurs [95, 113] préconisaient une taille de 7 éléments.

Algorithme 4.3: Algorithme tabou.
TABOU()

(1) **Initialisation**
(2) Générer une solution initiale x_0 ; $x^* = x_0$; $c^* \leftarrow f(x_0)$; x^* est la meilleure solution rencontrée, c^* est son coût et f la fonction économique (fonction qualité ou *fitness*).
(3) $k \leftarrow 0$, ListeTabou= $\emptyset$
(4) **répéter tant qu'**un critère de fin n'est pas vérifié
(5) Choisir parmi le voisinage de x_k, $V(x_k)$, le mouvement qui minimise f et qui n'appartient pas à Liste Tabou, $meilleur(x_k)$
(6) $x_{k+1} = meilleur(x_k)$
(7) **si** $(c(x_{k+1}) < c^*)$ **alors** $x^* \leftarrow x_{k+1}$, $c^* \leftarrow c(x_{k+1})$
(8) Mise à jour de ListeTabou

Dans cet algorithme (4.3), un mouvement permet de passer d'une solution valide à une autre, qui est dite voisine. Ce voisinage est constitué de toutes les solutions atteignables par une application de l'opérateur choisi.

Plusieurs améliorations au modèle de base ont été proposées. On peut citer :

- la stratégie d'intensification : il s'agit de repérer les éléments faisant partie des meilleures solutions trouvées, qui seront utilisés pour générer de nouvelles solutions, devant être proches de l'optimum. Elle est utilisée dans le problème de placement et d'allocation où est mémorisé le nombre d'entités qui sont placées [44] ;
- la stratégie de diversification : il s'agit de la stratégie inverse ; elle consiste à mémoriser les solutions les plus fréquemment rencontrées, qui seront utilisées pour attribuer des pénalités aux zones déjà explorées, forçant ainsi l'exploration de nouvelles régions [72, 179] ;
- la stratégie d'aspiration [96, 97] : une solution, qui ne serait pas acceptée par les règles du tabou, peut être sélectionnée si elle permet d'atteindre une solution ayant un coût inférieur à la meilleure solution trouvée jusqu'alors ;
- la stratégie de détermination de la taille de la liste tabou [217] : ces stratégies peuvent être statiques (détermination de cette taille en fonction de la nature du problème) ou dynamiques ;
- la stratégie de sélection du meilleur voisin : elle peut se faire avec une stratégie *Best Fit* au cours de laquelle on sélectionne le meilleur du voisinage ; dans le cas du *First Fit*, la solution choisie sera la première qui réponde aux contraintes du tabou. Cette méthode est plus rapide, elle est souvent utilisée dans le cas de voisinage de grande taille.

L'apprentissage incrémental basé sur la population

Cette méthode est plus connue sous son appellation anglaise PBIL "Population-Based Incremental Learning". Cette méthode fut proposée par BALUJA en 1994 [6, 8, 7]. Le but de cette méthode est d'utiliser un vecteur de probabilité pour générer une population de solutions, cette dernière étant ensuite utilisée pour mettre à jour les probabilités. Cette méthode a été appliquée sur des fonctions numériques codées en

binaire, chaque élément du vecteur indiquant la probabilité d'avoir 1 à cette position, la probabilité d'avoir 0 étant obtenue par soustraction. Dans le cas de la maximisation du nombre de 1 dans un vecteur, l'objectif serait d'avoir un vecteur de probabilité de la forme $V = (0,999; \ldots; 0,999)$.

Algorithme 4.4: L'apprentissage progressif basé sur la population.
PBIL()

(1) **Initialisation** du vecteur de probabilité $V = (p_1; \ldots; p_m) = (0,5; \ldots; 0,5)$
(2) **tant que** la condition d'arrêt n'est pas vérifiée **faire**
(3) Générer la population $P = \{s_1; \ldots; s_m\}$ en utilisant V
(4) Evaluer toutes les solutions s_i en utilisant le fonction f
(5) Mettre à jour V en utilisant la meilleure solution s^*
(6) **fin tant que**
(7) **retourner** s^*

La mise à jour se déroule en deux étapes :

1. On sélectionne la meilleure solution en se basant sur les valeurs obtenues à l'aide de la fonction qualité f ;
2. Le vecteur V est mis à jour :

$$\forall i \in \{1 \ldots m\},\ p_i = p_i(1 - TA) + TA \cdot s^*(i),\ i \in]0,1[$$

TA est le taux d'apprentissage (Learning rate) qui fixe la vitesse d'apprentissage de l'heuristique.

Un mécanisme de mutation peut être ajouté à la mise à jour : il a pour but comme pour les AG d'éviter une convergence trop rapide. Il s'agit avec une certaine probabilité de modifier la valeur de p_i en l'augmentant ou en la diminuant légèrement.

Il existe plusieurs variantes de cette méthode, qui peuvent soit utiliser plus d'une solution pour la mise à jour, soit prendre la plus mauvaise solution comme repoussoir : elle apporte alors une contribution négative [8, 178].

Plusieurs extensions ont été proposées pour étendre ce modéle binaire vers l'espace des variables continues [183, 197].

Le mode de fonctionnement de cette heuristique est par certains aspects similaire à celui des OCF comme le montre l'algorithme 4.4 ; il s'en distingue par la phase de mise à jour. En effet, les probabilités (l'équivalent ici de la phéromone pour les "fourmis") sont mises à jour grâce à une méthode inspirée de l'apprentissage compétitif (competitive learning) utilisée pour l'apprentissage de carte de Kohonen (Kohonen's features map) [124]. On peut remarquer que même si la qualité des solutions est utilisée pour sélectionner la meilleure, cette valeur n'est pas employée lors du recalcul des probabilités. Seules les valeurs des bits de la solution sont utilisées par la formule de mise à jour, tandis que pour les OCF, la valeur de qualité pondère l'influence de la nouvelle solution qui modifie la phéromone.

Dans [163, 159], MONMARCHÉ *et al.* comparent cette méthode à deux adaptations d'algorithmes de fourmis que sont AS et ACS appliqués aux problèmes d'optimisation

numérique. Ils montrent qu'ACS et PBIL ont des résultats comparables sur les 10 fonctions proposées par Whitley [236, 237]. PBIL trouve 6 fois le minimum contre 5 fois pour ACS. Il est à noter que l'algorithme des fourmis semble être beaucoup plus sensible au paramétrage.

4.3 Comment les fourmis utilisent-elles leur mémoire ?

Dans la coopération entre fourmis, la phéromone est utilisée comme moyen de communication : c'est la variable stigmergétique. Toutefois, le rôle de ce dépôt n'est pas limité à cette transmission indirecte d'informations. Les fourmis déposent la phéromone sur le sol sous forme de trace. Lorsqu'une fourmi doit faire le choix entre deux chemins, elle sera guidée par la trace olfactive laissée par ses congénères. En empruntant un chemin, cet insecte va renforcer la trace qui s'y trouve, favorisant sa sélection par ceux qui le suivent. La voie qui n'aura pas été empruntée sera donc moins attractive. L'évaporation fera diminuer le taux de phéromone associé à ce dernier arc, accentuant ainsi le phénomène ; il sera même abandonné s'il n'est plus visité.

Ce mécanisme peut être comparé à celui de l'apprentissage et les traces de phéromone à une mémoire collective dans laquelle apparaît la solution du problème.

De même que le chemin le plus court émerge des traces de phéromone qui ont été déposées sur le sol, on espère que la forme de la meilleure solution apparaîtra dans la table de phéromone pour les fourmis artificielles. Les algorithmes des fourmis se placent, comme la plupart des heuristiques, dans la famille des méthodes utilisant une mémoire, ou dans la famille regroupée sous le nom de "programmation à mémoire adaptative" par Taillard [218, 219, 219]. Nous commencerons donc la section suivante par le modèle des AMP, puis nous tenterons de classer les heuristiques suivant l'utilisation de leur mémoire.

4.3.1 Emergence d'un modèle fédérateur : AMP

Taillard *et al.* dans "Adaptative Memory Programming : An unified view of metaheuristics" [219] proposent un modèle qui se veut fédérateur de toutes les heuristiques utilisant une forme de mémoire. Ils le désignent sous l'appellation : "Adaptative Memory Programming" (AMP) ou, en français, "programmation à mémoire adaptative".

Ces algorithmes adoptent le schéma de fonctionnement suivant : après l'initialisation, une solution est construite en utilisant les informations contenues dans la mémoire. La solution obtenue est ensuite améliorée par l'application d'une recherche locale. Enfin, le résultat est utilisé pour mettre à jour la mémoire. Cette boucle est répétée tant que le critère d'arrêt choisi n'est pas obtenu. Ce fonctionnement peut être schématisé par l'algorithme 4.5.

Algorithme 4.5: Forme générale d'un AMP.
AMP()

(1) Initialisation de la mémoire
(2) **répéter**
(3) Construire une nouvelle solution en utilisant l'information contenue dans la mémoire
(4) Amélioration de la solution provisoire à l'aide d'une recherche locale
(5) Mise à jour de la mémoire
(6) **jusqu'à** la satisfaction du critère d'arrêt

En partant de ce modèle, il suffit alors de définir les quatre éléments principaux d'un AMP, pour donner un aperçu d'une heuristique. On peut dire qu'il s'agit des quatre critères discriminants pour la méthode. Ces quatre éléments sont :

(a) la mémoire ;
(b) la procédure de création de solutions provisoires ;
(c) la procédure d'amélioration ;
(d) la mise à jour de la mémoire.

Dans un cas général :
- la mémoire permet de stocker des solutions, ou d'agréger sous forme d'une structure les particularités des solutions obtenues pendant la recherche ;
- les solutions provisoires sont créées en utilisant la mémoire ;
- les solutions provisoires sont améliorées par un algorithme glouton ou une heuristique plus sophistiquée ;
- les nouvelles solutions sont, soit placées dans la mémoire, soit utilisées pour modifier les valeurs de la structure stockant l'historique de la recherche.

4.3.2 Approfondissement du modèle et adjonction de nouveaux critères

Nous avons décidé de reprendre et de compléter ce modèle afin de comparer les systèmes à colonies de fourmis aux autres méta-heuristiques.

Nous allons d'abord préciser la notion de mémoire que peuvent utiliser les différentes méthodes. Nous proposons une distinction suivant trois critères :
- *une mémoire de travail* (mémoire à très court terme) lorsque seules les informations de la génération précédente sont conservées ;
- *une mémoire à court terme* quand des informations de quelques générations sont conservées et utilisées ;
- *une mémoire à long terme* quand on retrouve des traces sous une forme ou une autre de tout le déroulement de l'application.

La dénomination de ces trois catégories de mémoire est directement inspirée de la terminologie utilisée pour la description des différents processus cognitifs chez l'homme [5].

Nous avons également décidé d'ajouter deux éléments suplémentaires pour la caractérisation des heuristiques : il s'agit du critère d'évaluation de la qualité des solutions, et de la sélection des solutions qui participeront à l'itération suivante.

Les éléments à définir pour les méthodes deviennent donc :

(a) la mémoire (*de travail*, *à court terme*, *à long terme*) ;

(b) la procédure de création de solutions provisoires ;

(c) la procédure d'amélioration ;

(d) la méthode d'évaluation des solutions ;

(e) la sélection des solutions ;

(f) la mise à jour de la mémoire.

Pour ce qui est de l'utilisation de la fonction d'évaluation des solutions, on distingue principalement deux cas :

- l'utilisation directe de cette valeur : c'est le cas pour les méthodes à base de population ;
- le calcul de la différence avec la valeur associée à l'ancienne solution : c'est le cas pour certaines méthodes guidées.

4.3.3 Quelques heuristiques décrites sous la forme étendue des AMP

Il est donc possible de définir en six points la particularité d'un algorithme, comme nous allons le voir dans les exemples suivants.

1. Commençons par une méthode classique, qui n'a pas été décrite précédemment, mais qui montre que les méthodes inspirées de modèles naturels ne sont pas les seules à pouvoir être décrites suivant ce schéma.

   La méthode considérée est le Hill-Climber ou méthode dite du gravisseur de colline (GC) :

   (a) la mémoire : *de travail*, il s'agit de la solution courante ;

   (b) la procédure de création de solutions provisoires : elle consite à créer toutes les solutions dites voisines, c'est-à-dire ne s'écartant de la solution courante que par l'application d'un simple opérateur de modification ou de mouvement ;

   (c) la procédure d'amélioration : il n'y en a pas ;

   (d) la méthode d'évaluation des solutions : se fait par une fonction gain, alors que pour le PVC c'est la différence entre la solution courante et la solution temporaire considérée ;

   (e) la sélection des solutions : soit c'est la meilleure qui est choisie (best-fit) soit c'est la première qui améliore la solution courante (first-fit) en définissant un ordre sur le voisinage ;

   (f) la mise à jour de la mémoire : c'est le remplacement de la solution courante par celle sélectionnée.

2. L'AG déjà présenté dans la section consacrée aux algorithmes génétiques peut être décrit sous la forme suivante :

   (a) la mémoire : *à long terme* ; il s'agit d'une mémoire de population, formée par l'ensemble des chromosomes des individus de la génération ;

   (b) le procédure de création de solutions provisoires : c'est la recombinaison ou crossover ;
   (c) la procédure d'amélioration : il est possible d'y placer la mutation, sans que, pour autant, celle-ci ait forcément comme effet d'améliorer le résultat de la fonction qualité ;
   (d) la méthode d'évaluation des solutions : par une fonction qualité ;
   (e) la sélection des solutions : élimination des solutions les plus mauvaises ;
   (f) la mise à jour de la mémoire : il s'agit de la fonction de remplacement qui produit la nouvelle génération.

3. Regardons les méthodes guidées que nous avons abordées. D'abord la recherche par dispersion.
   (a) la mémoire : *à long terme*, c'est la population ;
   (b) la procédure de création de solutions provisoires : c'est l'application des opérateurs de recombinaisons qui forme des solutions provisoires ;
   (c) la procédure d'amélioration : c'est l'application de l'opérateur de projection qui modifie les solutions provisoires pour les rendres valides ;
   (d) la méthode d'évaluation des solutions : par une fonction qualité ;
   (e) la sélection des solutions : on applique l'opérateur d'élimination pour retirer des solutions ;
   (f) la mise à jour de la mémoire : les solutions restantes forment la nouvelle population.
4. Ensuite le recuit simulé.
   (a) la mémoire : *de travail*, c'est la solution courante ;
   (b) la procédure de création de solutions provisoires : il s'agit d'une solution voisine ;
   (c) la procédure d'amélioration : il n'y en a pas ;
   (d) la méthode d'évaluation des solutions : c'est une fonction de calcul de gain ;
   (e) la sélection des solutions : on choisit la solution si elle améliore la solution courante, ou avec une fonction de probabilité dans le cas contraire ;
   (f) la mise à jour de la mémoire : c'est la nouvelle solution courante.
5. Enfin la recherche tabou dans sa version la plus simple.
   (a) la mémoire : deux types sont utilisés, celle *de travail* qui est la solution courante et celle dite *à court terme* présente sous la forme de la liste tabou où sont stockées les dernières solutions visitées ou opérations efféctuées ;
   (b) la procédure de création de solutions provisoires : comme pour le GC, il s'agit des solutions voisines ;
   (c) la procédure d'amélioration : il n'y en a pas ;
   (d) la méthode d'évaluation des solutions : c'est une fonction gain (*cf.* GC)
   (e) la sélection des solutions : la meilleure qui n'est pas interdite par l'utilisation de la liste tabou ;

(f) la mise à jour de la mémoire : c'est la mise à jour de la solution courante.

Les stratégies de diversification et d'intensification introduisent la troisième forme de mémoire qui est celle *à long terme* : il s'agit ici de fréquences d'apparition des différents éléments dans les solutions.

6. Regardons ensuite l'heuristique PBIL
   (a) la mémoire : *à long terme*, sous la forme d'un vecteur de probabilité ;
   (b) la procédure de création de solutions provisoires : les solutions sont générées en utilisant le vecteur de probabilité ;
   (c) la procédure d'amélioration : aucune ;
   (d) la méthode d'évaluation des solutions : calcul d'une fonction de qualité pour chaque solution ;
   (e) la sélection des solutions : seule la meilleure participe à la mise à jour, ensuite les solutions sont détruites ;
   (f) la mise à jour de la mémoire : les probabilités sont changées en accord avec les éléments qui composent la meilleure solution ; on peut également trouver une mutation des probabilités.
7. Nous passons au premier algorithme qui a donné le point de départ des méthodes dites à base de colonies de fourmis, qui est Ant-System :
   (a) la mémoire : *à long terme*, c'est la table de phéromone ;
   (b) le procédure de création de solutions provisoires : les solutions sont créées en utilisant les valeurs de phéromone, soit de façon probabiliste (les éléments étant choisis avec une probabilité proportionnelle à la phéromone), soit de façon déterministe (on prend l'élément qui a le plus fort taux de phéromone) ;
   (c) la procédure d'amélioration : une recherche locale peut être ajoutée ;
   (d) la méthode d'évaluation des solutions : chaque fourmi évalue sa solution et calcule son apport en phéromone ;
   (e) la sélection des solutions : toutes les fourmis participent à l'itération suivante sans conserver de trace de leur passé autre que la mémoire à long terme (table de phéromone) ;
   (f) la mise à jour de la mémoire : les valeurs de phéromone sont diminuées par un phénomène d'évaporation, puis les meilleures fourmis peuvent renforcer leurs éléments dans la table de phéromone.

On peut donc voir que beaucoup de ces méthodes partagent des caractéristiques communes et que les principales différences viennent de la mémoire et de son utilisation. C'est pour cette raison que nous avons insisté dans ce chapitre sur la caratérisation de cette mémoire. Les différents types d'utilisation seront présentés dans la section suivante.

4.3.4 Les différents types d'utilisation de la mémoire

Taillard [218] propose de classer les heuristiques en deux, suivant deux types d'utilisation de la mémoire qui sont :

	mémoire réduite	mémoire de fréquence ou statistique	mémoire répartie ou de population
Gravisseur de colline	√		
AG			√
Recherche par dispersion			√
Recuit simulé	√		
Recherche tabou	√	√	
PBIL		√	
Ant system		√	

TAB. 4.1 – Utilisation qui est faite de la mémoire pour différentes heuristiques

– les heuristiques à mémoire de population ;
– les heuristiques à mémoire statistique.

Ce regroupement peut être affiné en introduisant une troisième catégorie, celle des méthodes à mémoire réduite qui se distinguent des méthodes de population par l'utilisation qui est faite de l'information emmagasinée.

Les techniques à mémoire réduite. Il s'agit de mémoriser des solutions qui ont déjà été trouvées ou des modifications qui ont déjà été effectuées. Ces informations ont le plus souvent comme but de diriger le déplacement de l'algorithme dans l'espace de recherche.

Les systèmes à mémoire de fréquences ou statistique. L'objectif de cette base statistique est de caractériser la structure de bonnes solutions, et c'est dans les variations de ce squelette de solution que l'on espère trouver la meilleure solution.

Les systèmes à mémoire répartie ou de population. Le principe est d'utiliser un ensemble de solutions qui vont servir de base aux différents opérateurs de modifications, ces altérations ayant pour objectif de faire apparaître la solution espérée de la masse en évolution. C'est la pression de sélection qui doit la faire émerger, cette action étant le fruit de la sélection faite grâce à la fonction qualité.

Dans le tableau 4.1 est indiqué le type d'utilisation que les différentes heuristiques abordées font de leurs mémoires.

Dans la majorité des cas, on peut être amené à dire que la mémoire statistique est indirectement une mémoire de population, à la distinction près que dans le cas de la population certains éléments peuvent ne pas y figurer à l'origine ou pendant l'exécution de l'heuristique. Au contraire, dans le cas des statistiques tous les éléments sont normalement représentés même s'ils ont une très faible probabilité d'apparition. On peut dire que la mutation pour la population tend à corriger cette distinction. Pour prendre l'exemple classique de la maximisation du nombre de 1 dans une chaîne binaire, il est possible que dans le cas d'un AG, aucun des chomosomes générés initialement ne comportent la valeur 1 à la position 4. Dans ce cas, l'application seule de l'opérateur de recombinaison ne permettra pas d'obtenir un individu optimal qui ne comporte que des 1. Dans le cas d'une méthode basée sur des probabilités, même si la structure contenant la mémoire associe une faible chance de sélection du 1 en quatrième position, elle existe, et un individu comportant cette caracteristique finira par apparaître. Dans le cas de l'AG, seule la mutation du gène placé au quatrième locus pourra faire apparaître ce 1.

4.3.5 La difficulté d'une comparaison

Afin de comparer les différentes méthodes, il est possible de tester leurs performances sur des instances classiques de problèmes connus. On s'aperçoit que ces méthodes n'ont pas les même performances suivant la nature du problème ou de l'instance. Il nous paraît donc difficile de donner ici une comparaison globale.

Il nous est quand même possible de constater que plus une méthode utilise un mode de mémoire élaboré, plus elle a de chance d'être performante. Prenons l'exemple du tabou : en partant d'une méthode de gradient facilement sujette à être piégée au niveau des optima locaux, on y ajoute une mémoire à moyen terme, la liste tabou. Les performances s'en trouvent nettement améliorées. L'adjonction de mémoires à long terme rend cette méthode très robuste [217].

Les méthodes utilisant des mémoires évoluées semblent être les plus robustes, mais on leur ajoute souvent des méthodes de recherche locale avec des mémoires plus réduites (à court terme). La robustesse d'une heuristique ne viendrait elle pas d'un bon dosage entre les trois variétés de mémoire qui ont été abordées et d'un apport d'informations réalisé par exemple par une recherche locale ?

4.3.6 La difficulté d'une classification

Les performances des différentes heuristiques sont souvent liées à leur adaptation à la nature des problèmes traités. Ainsi, pour certaines méthodes, il n'est pas possible de donner un paramétrage général d'AMP pour les définir. Le modèle ne se définit alors que vis-à-vis d'un type de problème donné. Dans le cas du QAP, qui fait l'objet de la section suivante, une mémoire de population s'ajoute à celle de phéromone dans les différents algorithmes abordés.

Dans le cas des métaheuristiques, la difficulté vient de l'imbrication des méthodes, et il est nécessaire, dans ce cas, d'utiliser un modèle de poupées russes où chaque heuristique sera décrite indépendamment sous une forme d'AMP. De même, il sera difficile de classer ces algorithmes suivant leur type d'utilisation de la mémoire.

Des problèmes de classification peuvent également provenir simplement du paramétrage des algorithmes. Ainsi, si nous prenons un AG sans recombinaison où la population est réduite à un individu, nous sommes en présence d'une technique à mémoire réduite et non plus d'un système à mémoire de population.

Talbi [225] a présenté une intéressante classification des méta-heuristiques permettant de prendre en compte la majorité des cas d'hybridation.

4.4 Conclusion

La majorité des heuristiques utilisées actuellement s'inspirent de modèles naturels pour leur mode de fonctionnement. Elles se différencient des anciennes méthodes par leur mode de conception. Les heuristiques étaient construites sur les propriétés du problème à résoudre, tandis que pour les méthodes inspirées de la nature, leur forme générale est déjà présente. Tout l'art de leur conception réside dans l'introduction de bonnes propriétés et d'informations liées au problème : pour le PVC, c'est la longueur des arcs, ce qui est le cas dans la méthode de recombinaison appelée "edge assembly

crossover" [168] qui tente de conserver les arcs courts, ou dans d'autres heuristiques où sont utilisées des recherches locales spécifiques aux problèmes .

Les algorithmes à colonie de fourmis s'inscrivent dans cette logique. Du modèle biologique, DORIGO *et al.* ont extrait le modèle d'utilisation de la phéromone, ce qui a donné le squelette des ACO. C'est de l'adaptation de ce modèle à la nature des problèmes que proviennent les bonnes performances de cette méthode, comme le dit DORIGO dans [62]. Pour AS, les meilleurs résultats ont été obtenus avec la prise en compte de la distance et de la fonction de recherche locale.

Les méta-heuristiques et heuristiques supportent difficilement un classement, ce dernier étant rendu complexe par leur adaptation au problème considéré. Nous avons proposé une méthode de classification transversale qui s'inspire de la taxonomie d'AMP de TAILLARD. Elle caractérise l'utilisation qui est faite de la mémoire dans la méthode et offre une base de travail intéressante mais insuffisante.

Deuxième partie

Les fourmis pour le PAQ

Chapitre 5

Le problème d'affectation quadratique

Le problème d'affectation quadratique (PAQ) ou en anglais *quadratic assignment problem* (QAP) est un problème classique en optimisation combinatoire. Il est connu pour être très complexe à résoudre. Il a fait l'objet de nombreuses publications et est un cas d'étude académique. Nous aborderons sa définition et ses applications. Nous citerons également les méthodes qui ont déjà été utilisées pour le résoudre, puis nous nous attarderons sur les OCF qui ont été appliqués à ce problème. Nous terminerons enfin ce chapitre en présentant la méthode hybride entre les fourmis et la recherche tabou que nous avons proposée ANTabu.

5.1 Principe

Ce problème présenté pour la première fois par Koopmans et Beckmann en 1957 [125] est considéré comme un modèle mathématique pour le placement d'activités économiques indivisibles. Il consiste à déterminer le meilleur placement d'activités dans un ensemble de positions. Afin de déterminer la qualité d'une solution, plusieurs mesures sont utilisées :

- le coût des échanges qui est fonction du flux (f) entre les activités et les distances (d) parcourues ;
- le coût d'installation (b) de l'activité sur une position donnée.

Ces valeurs sont regroupées dans trois matrices :

- F la matrice des flux, ou l'on trouve les valeurs des flux f entre les différentes activités ;
- D la matrice des distances, ou l'on trouve la distance d entre les positions ;
- B la matrice des coûts d'installation, ou l'on trouve le coût c pour placer toutes les activités sur chacune des positions.

L'objectif du problème est de placer toutes les activités pour que le coût global soit minimum, ce dernier étant égal à la somme des coûts d'échange et d'installation pour toutes les activités dans les positions sélectionnées.

$$\min_{\phi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{i,j} d_{\phi(i),\phi(j)} + \sum_{i=1}^{n} b_{i,\phi(i)}$$

Dans le cas de l'implantation d'activités, les matrices F et D sont symétriques, avec zéro sur la diagonale et aucune valeur négative pour les trois matrices.

Le problème proposé par KOOPMANS et BECKMANN peut se présenter sous la forme d'un ensemble d'activités que l'on doit placer sur un ensemble de positions. Ces deux ensembles étant de même taille (n), une solution est l'ensemble d'affectations qui donne pour un élément (i) la position qui lui est attribuée ($\phi(i)$).

Beaucoup de recherches ont été menées sur ce problème complexe à résoudre. Un problème de taille supérieure à 20 ne peut pas être résolu avec un temps de calcul raisonnable. La résolution complète d'une instance de PAQ demande l'évaluation de toute les possibilités d'affectation qui sont au nombre de $20^{20} = 1,048576 \cdot 10^{26}$, ce qui représente, si on évalue un milliard de solutions à la seconde sur une machine, un temps de calcul de 3,32 milliards d'années.

SAHNI et GONZALEZ [186] ont même démontré qu'il est $\mathcal{NP}$-dur, et qu'en trouver une ε-approximation ne peut être réalisé en un temps polynomial que si $\mathcal{P}$=$\mathcal{NP}$.

Ce problème trouve de nombreuses applications concrètes comme :

- le câblage de tableaux de bord [202] : il s'agit de déterminer le schéma de câblage qui va minimiser les longueurs de fils utilisées ;
- le placement des caractères pour des machines à écrire [30] : il s'agit de déterminer la place des caractères sur un clavier qui minimisera les distances parcourues, donc la durée pour la saisie d'un groupe de textes défini ;
- le placement de services dans un hôpital : ici on cherche à minimiser la distance parcourue par les malades entre les différents services [71, 130] ;
- l'ordonnancement de lignes de production parallèle [92] ;
- l'organisation dans une équipe de course de relais [112] ;
- l'analyse de réactions chimiques pour des composés organiques [226].

Il existe d'autres formalisations mathématiques, qui sont équivalentes pour ce problème, et qui permettent différentes approches de résolution. Toutes ces formulations sont par ailleurs décrites par BURKARD *et al.* dans un article faisant l'état de l'art pour le PAQ [28].

5.2 La version simplifiée

La version simplifiée est celle qui est adoptée pour les différents algorithmes que nous verrons, et qui est également la plus fréquemment utilisée. Il s'agit d'une simplification de la méthode proposée par KOOPMANS et BERKMANN. Dans la version qui nous intéresse, on ne tient pas compte du coût d'implantation, la formule devient donc :

$$\min_{\phi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{i,j} d_{\phi(i),\phi(j)}$$

Dans ce problème, seuls sont considérés les coûts engendrés après l'implantation des activités sur les positions disponibles.

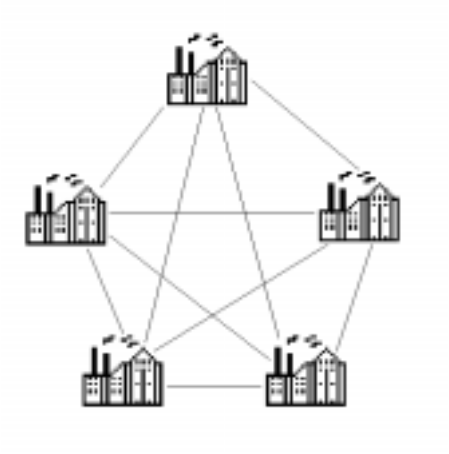


FIG. 5.1 – Schéma représentant l'implantation d'usines

Pour illustrer ce problème regardons deux petits exemples, le premier donne la forme d'un problème concret qui peut se modéliser par une instance du PAQ, le second donne un exemple de codage d'une instance.

5.2.1 L'exemple de l'implantation d'usines

Un exemple de PAQ pour le choix de l'implantation d'usines est présenté en fig. 5.1.

Le problème exposé consiste à implanter n usines sur n sites, une et une seule usine devant être placée sur chaque site. Les distances inter-sites sont connues. La distance entre deux sites quelconques i et j vaut d_{ij}. Les usines une fois implantées devront échanger des produits entre elles lors de leur fonctionnement. Les flux passant d'une usine k à une autre l sont quantifiés par un coût connu (c_{kl}). L'objectif dans ce cas est de déterminer l'implantation qui permettra par la suite de minimiser le coût total de transit entre les usines (ces coûts étant proportionnels à la quantité transportée et à la distance parcourue).

5.2.2 Un exemple pédagogique

Pour cet exemple dont la formalisation est exposée dans figure 5.2, nous prenons quatre objets géométriques. Il est possible de les placer dans quatre emplacements distincts. Ces quatre figures communiquent de façon bidirectionnelle entre elles, chaque objet communique avec celui qui le précède et celui qui le suit, sauf dans le cas du carré et de l'ellipse qui n'ont qu'un voisin. Ce problème peut être formalisé sous la forme des deux matrices présentées : la première est la matrice des flux F et la seconde la matrice des distances D. Dans notre exemple, les distances des diagonales sont les plus longues, afin d'obtenir un placement cyclique des objets. Un exemple d'une solution aléatoire est présenté, le coût des échanges est ici égal à 5. Une des solutions optimales, qui sont au nombre de huit, est également indiquée, son coût étant égal à trois. Dans cet exemple le codage des couples d'affectation est en partie implicite, en effet la position est donnée par l'indice du tableau où est placé l'objet ; par exemple le première objet est à la position 1.

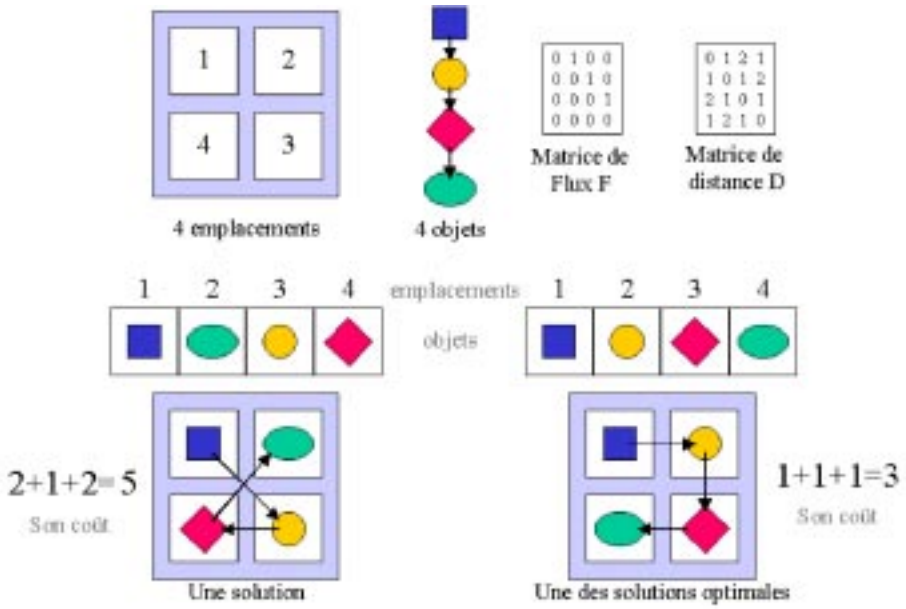


FIG. 5.2 – Schéma représentant un exemple de PAQ

5.2.3 Instances du PAQ

La QAPLIB est un regroupement d'instances du PAQ, qui ont été réunies par BURKARD, KARISCH et RENDL [29]. Cette librairie permet d'avoir un ensemble de problèmes sur lesquels il est possible de comparer les heuristiques. Les instances qui composent la QAPLIB sont désignées par un nom qui se décompose sous la forme d'un radical de trois lettres mis pour les trois premières lettres du nom du premier auteur. Le suffixe est constitué d'une valeur qui représente la taille du problème. Ainsi *bur26a* a été créée par R.E. BURKARD et J. OFFERMANN, et correspond à un problème contenant 26 sites et 26 objets. Dans cet exemple, le nom est complété par une lettre qui sert à différencier les instances de taille identique pour un même auteur.

Des informations sur les instances sont disponibles dans l'article et également sur le site web[1] (ce dernier étant naturellement mis à jour). Il est possible d'y trouver :

- la solution optimale quand elle est connue ;
- la meilleure solution trouvée accompagnée de la méthode qui l'a obtenue ;
- une valeur plancher pour le coût.

Il est à noter qu'elle n'est pas considérée comme très pertinente pour les instances de taille supérieure à 30.

La classification des instances peut se faire en trois catégories distinctes qui se différencient par la nature de leur matrice de flux et de leur matrice de distances :

- les instances uniformes sont générées aléatoirement, elles sont donc très peu structurées ;
- lorsqu'elles sont générées sur une grille, une structure est introduite dans les données du problème ;
- les instances réelles ou assimilées sont très fortement structurées.

Bachelet a proposé une classification de la difficulté des instances du PAQ [3].

[1] http ://www.opt.math.tu-graz.ac.at/qaplib/

5.3 Méthode de résolution pour le PAQ

Il existe quelques algorithmes de résolution exacte pour le PAQ, parmi lesquels on peut citer la méthode *Branch and Bound* [93, 134, 170] et les "cutting plane methods" [14, 171].

De nombreuses heuristiques ont été appliquées au PAQ comme nous le verrons dans la section suivante.

5.3.1 Les heuristiques pour le PAQ

Beaucoup de travaux on été réalisés pour l'élaboration de méthodes exactes de résolution du PAQ, mais elles ne sont pas utilisables pour des instances de tailles supérieures à 20, le temps de calcul étant un facteur rédhibitoire. Pour pallier ce défaut, de nombreuses heuristiques ont été développées, offrant ainsi de bonnes solutions pour un temps de calcul raisonnable.

Les méthodes qui ont été utilisées pour la résolution sont :
- les méthodes constructives ;
- les méthodes d'énumération limitées ;
- les méthodes d'optimisation locale ;
- la recherche tabou ;
- le recuit simulé ;
- les algorithmes génétiques ;
- la méthode GRASP (*greedy randomized search procedure*) ;
- la recherche par dispersion ;
- les méthodes à base de colonies de fourmis.

5.3.2 Les OCF pour le PAQ

Le premier OCF pour le PAQ nommé AS-QAP a été proposé par Maniezzo *et al.* en 1994 [145]. Plusieurs versions améliorées ont ensuite été développées.

Maniezzo et Colornie ont créé une version améliorée du premier algorithme [144]. Plusieurs auteurs ont décrit des méthodes inspirées des OCF en améliorant l'algorithme initial [141, 203, 211, 68, 216] et des heuristiques inspirées des fourmis [86, 224].

On distingue deux classes d'OCF pour le PAQ :
- une classe "constructive" où les fourmis construisent itérativement les solutions, c'est le cas des méthodes inspirées des OCF ;
- une classe "pertubatrice" où les fourmis ont pour fonction de modifier ou de proposer des nouvelles solutions à partir de solutions existantes.

L'utilisation du modèle des OCF impose une représentation du problème sous forme de graphe. Le PAQ peut se définir comme un maillage de positions ou d'objets interconnectés, dans lequel il faut placer des objets, ou des positions, au niveau des nœuds. L'action des fourmis se définit alors comme un déplacement de nœud en nœud, le choix à chaque étape est guidé par la phéromone et dans certains cas par une mesure heuristique. Plus le taux de phéromone est important, et plus l'affectation correpondante est *a priori* intéressante.

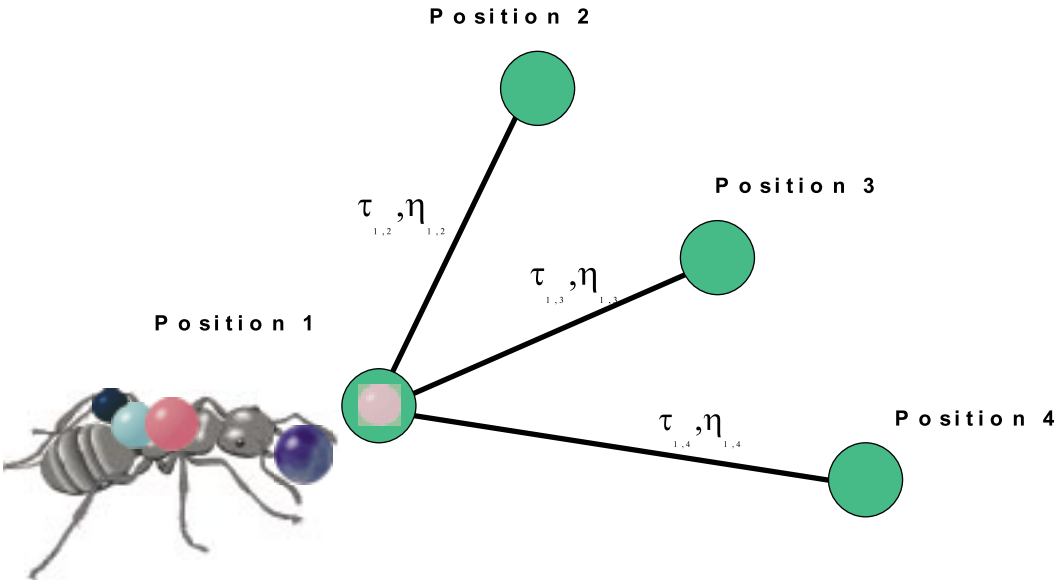


FIG. 5.3 – Le choix de la prochaine affectation de la fourmi : arrivée en position 1, elle choisit l'objet à affecter, puis suivant les valeurs de phéromone (τ) et heuristiques (η), elle fait son choix entre les trois positions disponibles $(2, 3, 4)$

La méthode doit vérifier d'abord que les contraintes du PAQ sont respectées, c'est-à-dire qu'un objet n'est affecté qu'à une et une seule position, et ensuite qu'une position contient exactement un objet.

Une solution pour les fourmis se présente sous la forme de n couples (i, j) correspondant à l'affectation de l'objet i à la position j, ou sous la forme d'une permutation de n entiers où l'objet i situé à la $j^{\text{ème}}$ place de la permutation sera affecté à la position j.

Pour les algorithmes, deux cas peuvent se présenter :
- soit on affecte les objets aux positions ;
- soit on affecte les positions aux objets.

Dans le cas de l'affectation des objets aux positions, qui sera le seul illustré dans la suite de cette section, il est necessaire dans un premier temps de choisir l'objet qui sera affecté, ensuite sera choisie la position où il sera placé.

Pour le choix de l'objet, la phéromone, une valeur heuristique ou encore un choix aléatoire peuvent être utilisés afin de trouver le meilleur ordre d'affectation (cette étape influence évidemment les performances de la méthode).

Dans les heuristiques qui ont été proposées :
- soit l'ordre des objets est fixé tout au long de l'exécution, le choix se faisant suivant une heuristique [145, 144] ;
- soit les objets sont choisis aléatoirement suivant une distribution uniforme au cours de l'exécution [203, 211, 216].

Pour le choix de la position les valeurs de la phéromone et la valeur heuristique peuvent être utilisées (fig. 5.3). Quand les fourmis ont toutes terminé de construire leur solution, elles vont déposer une quantité de phéromone proportionnelle à la qualité de la solution qu'elles ont contruite pour renforcer les affectations choisies (couple

TAB. 5.1 – Méthode de recherche locale utilisée par les OCF

	AS-QAP	AS2-QAP	ANT-QAP	$\mathcal{MM}$AS-QAP	FANT	HAS-QAP	ANTabu
2-opt amélioré	√	√	√	√	√	√	
Recuit simulé	√						
Recherche tabou				√			√

objet/position).

Tous les OCF proposés sont des algorithmes hybrides. Il s'agit de méthodes de fourmis intégrant une méthode de recherche locale. Cette recherche locale est utilisée dans les heuristiques pour améliorer les solutions générées par les fourmis. Les recherches locales sont des variantes du 2-opt, des recuits simulés ou des recherches tabou comme le montre le tableau 5.1 (lorsque deux recherches locales sont indiquées, cela signifie que l'heuristique a été testée avec chacune d'elles).

Nous allons à présent détailler les méthodes utilisant les OCF. Pour leur description, nous nous intéresserons uniquement aux caractéristiques propres à chacune de ces méthodes, laissant de côté les caractéristiques générales que nous avons déjà décrites. Le tableau 5.2 regroupe les caractéristiques qui seront abordées. Il est à noter que tous les OCF décrits dans la section suivante se placent dans le cas d'une affectation des objets aux positions.

5.3.3 *Ant-System* pour le PAQ

Cette heuristique est plus connue sous le nom de AS-QAP [145, 67]. Dans AS-QAP, l'ordre d'affectation est défini par un pré-ordonnancement des objets ; ce classement se fera suivant les valeurs décroissantes des sommes de flux potentiels de chaque objet (échange important avec les autres objets).

À chaque étape, un objet i est placé à une position j : ce choix est fait suivant une probabilité qui favorise les positions ayant le plus fort taux de phéromone et également suivant une information supplémentaire fournie par une heuristique. Le but de l'heuristique est de favoriser le placement des objets ayant une forte somme de flux potentiels sur des positions minimisant leur distance vis-à-vis des autres positions. Pour cela l'heuristique attribue une valeur mesurant l'intérêt de l'affectation de l'objet dans chacune des positions disponibles. La valeur heuristique utilisée pour le choix de l'affectation est inversement proportionnelle au coût attribué à ce placement. Le coût estimé est le produit de la somme des flux pour l'objet (vis-à-vis des autres objets) par la somme des distances pour une position (par rapport aux autres positions). L'ensemble des valeurs heuristiques (pour chaque objet dans chaque position) est déterminé initialement.

A chaque étape les fourmis affectent l'objet suivant à une position qui n'a pas encore été utilisée en se basant sur une probabilité qui est fonction du taux de phéromone associé aux positions disponibles pour cet objet, ainsi que de la valeur heuristique. La formule est identique à celle présentée pour AS-TSP (tab.3.1).

La mise à jour de la matrice de phéromone utilise le même principe que celui de AS-TSP et les mêmes formules (tab.3.2).

TAB. 5.2 – Caractéristiques des différents OCF qui ont été appliqués au PAQ.
Φ désigne l'utilisation de la phéromone et $\mathcal{H}$ l'utilisation de la valeur heuristique

	AS-QAP	ANTS-QAP	$\mathcal{MM}$AS-QAP	FANT	HAS-QAP	ANTabu
Création des solutions :						
– constructrice	√	√	√	√		
– perturbatrice					√	√
Choix de :						
– l'objet	pré-ordre	pré-ordre	aléatoire	aléatoire	NA	NA
– la position	$\Phi + \mathcal{H}$	$\Phi + \mathcal{H}$	Φ	Φ	NA	NA
Heuristique	max. des flux/ min. des distances	GLB & LBD	NA	NA	NA	NA
Mise à jour Φ :						
– fourmis utilisées pour le renforcement	toutes	toutes	la meilleure	la fourmi & la meilleure	la meilleure	toutes
– évaporation	√		√		√	√
Caractéristiques particulières	NA	Branch & Bound	min-max	1 seule fourmi	sélection des solutions	matrice de fréquences modèle parallèle min-max

On peut citer deux améliorations de AS-QAP :
- ANTS-QAP qui fera l'objet de la section suivante ;
- AS2-QAP qui ne change que par le calcul de la valeur heuristique utilisant le GLB [93, 134] à chaque itération, et par la formule de calcul de probabilité d'affectation des objets dans les positions qui est la même que pour ANTS.

5.3.4 L'algorithme ANTS pour le PAQ

ANTS est une méthode améliorée des OCF présentée par Maniezzo [141] et appliquée au PAQ. L'appellation ANTS vient de ***A****pproximate* ***N****ondeterministic* ***T****ree* ***S****earch*. Ce nom est dû au fait que cette méthode se rapproche du *Branch & Bound*. En fait, ANTS est l'évolution d'une méthode exacte à laquelle le principe de coopération des fourmis a été ajouté.

Les fourmis placent de manière itérative un objet sur une position. Ce placement permet de définir une solution partielle. L'heuristique utilisée pour guider le choix des positions s'écarte un peu de celle adoptée pour AS. Elle estime le coût minimum *lower bound* nécessaire pour compléter la solution partielle. Cette valeur est utilisée pour évaluer l'intérêt de l'affectation suivante. La valeur d'intérêt d'une affectation est obtenue en couplant le coût de la solution partielle et celui de la limite inférieure utilisant ce placement. Dans ce cas, plus cette valeur de coût minimum estimé est faible, plus l'affectation est intéressante. Cette estimation offre la possibilité d'écarter des affectations si le coût minimum estimé est plus élevé que celui de la meilleure solution trouvée.

Le calcul de la borne minimum est obtenu par la formule de Gilmore-Lawler [93, 134] appelé GLB. Cette valeur est calculée au démarrage de l'algorithme. Son coût de calcul élevé a incité l'auteur à proposer une version plus légère nommée LBD, qui a une complexité en $O(n)$ contre $O(n^3)$ pour GLB.

Dans cette méthode, le cas de l'affectation des positions aux objets est également considéré. Dans les deux cas les objets et les positions sont ordonnés. Les auteurs montrent l'importance pour leur méthode du choix du sens d'affectation suivant les instances. Mais aucun automatisme n'a été trouvé pour savoir à l'avance celui qui sera le plus performant pour un problème donné. Le classement des positions et des objets se fait en utilisant les valeurs de la GLB.

Ensuite, chaque fourmi a pour rôle de placer un objet à chaque étape dans la position suivante, cela en accord avec l'intérêt associé à chaque objet pour cette position. Cet intérêt est donné par le calcul d'une probabilité basée sur le taux de phéromone et la valeur heuristique.

5.3.5 $\mathcal{MM}$AS pour le PAQ

$\mathcal{MM}$AS-QAP est l'adaptation de l'algorithme Max-Min de Stützle et Hoos [209] pour le PVC. Cette méthode a la particularité de ne pas utiliser de valeur heuristique, puisque ses choix ne sont dirigés que par les valeurs de phéromone. Il s'agit d'une variante de la méthode AS, dont elle se distingue en n'utilisant qu'une seule solution pour la mise à jour qui est :
- soit la meilleure solution de l'itération ;

– soit la meilleure trouvée depuis le début de l'exécution.

Lors de l'exécution, la fréquence d'utilisation de la meilleure solution depuis le début aux dépens de la meilleure de l'itération peut varier. STÜTZLE [205] propose une règle qui augmente cette fréquence durant l'exécution de l'algorithme. C'est cette méthode qui offre les meilleures performances.

Pour éviter une stagnation de l'algorithme, deux valeurs de bornes sont introduites $[\tau_{min}, \tau_{max}]$ qui limitent les valeurs que peut prendre la phéromone. La valeur maximale est utilisée dans ce cas comme valeur d'initialisation afin de favoriser l'exploration [205].

A chaque étape, la fourmi (k) choisit aléatoirement un objet, et décide ensuite de son site de placement suivant une probabilité qui est uniquement fonction des taux de phéromone (la formule est celle de AS-TSP où la valeur heuristique serait inexistante). Les auteurs justifient ce choix d'abord par le gain apporté par la diminution du nombre de paramètres disponibles, donnant ainsi une méthode plus générique, mais également par le fait qu'ils utilisent pour la mise à jour la solution après l'utilisation d'une recherche locale.

Pour le choix de l'affectation, on peut retrouver deux des méthodes définies dans Ant-Q (*cf.* 3.5.1) :

– une règle proportionnelle pseudo aléatoire où on prendra, soit l'affectation qui a le plus fort taux de phéromone, soit l'affectation obtenue au moyen d'une roulette basée sur les taux de phéromone ;
– une règle proportionnelle aléatoire où l'on choisira l'affectation en utilisant une roulette basée sur le taux de phéromone.

5.3.6 L'algorithme FANT

FANT ou *Fast Ant System* est une méthode proposée par Taillard et Dorigo [216]. Cette heuristique comme $\mathcal{MM}$AS-QAP n'utilise pas de valeurs heuristiques pour guider le choix des fourmis, la formule de choix de l'affectation d'un objet sur une position étant simplement guidée par la phéromone. Cette méthode se distingue par le nombre de fourmis utilisées et par le mode de mise à jour de la phéromone.

Cette heuristique a la particulatité de n'employer qu'une seule fourmi, ce qui lui permet de trouver très vite de bonnes solutions améliorées par l'utilisation d'une recherche locale.

La formule de mise à jour de la phéromone tient compte à la fois de la solution courante et de la meilleure solution trouvée, l'influence respective de ces deux solutions étant pondérée par deux variables. Deux mécanismes supplémentaires peuvent modifier le comportement de la fonction de mise à jour :

– si la fourmi améliore la meilleure solution trouvée, la matrice de phéromone sera réinitialisée et le coefficient pondérant l'influence de la solution courante est remis à sa valeur initiale : on se place dans une phase d'intensification ;
– si la fourmi retrouve la meilleure solution, la matrice est réinitialisée et l'influence de la solution courante est augmentée : c'est une phase d'exploration.

5.3.7 HAS-QAP

HAS-QAP (*Hybrid Ant System for the QAP*) a été proposé par Gambardella, Taillard et Dorigo [86].

Cette méthode se distingue principalement des autres techniques pour le PAQ que nous avons décrites par le fait qu'elle ne construit pas totalement ses nouvelles solutions, mais au contraire modifie celles déjà existantes. Dans la mise à jour, on retrouve également le fait comme pour d'autres variantes des OCF, que seule la meilleure solution est autorisée à apporter sa contribution pour la mise à jour de la table de phéromone.

Pour la construction ou, devrait-on dire, la modification, la procédure est similaire à un 2-opt. Il s'agit d'échanger la position de deux objets. Pour cela on choisit tout d'abord le premier objet (i) de manière totalement aléatoire, tandis que l'on va choisir le second objet (j) suivant l'information fournie par la phéromone. Pour cela, on calcule la somme des taux de phéromone associée aux deux couples objets/positions obtenus si on faisait l'échange. La valeur de la somme désigne l'intérêt d'échanger les deux objets. Après avoir calculé les sommes de taux de phéromone après l'échange de l'objet i avec tous les autres, on prend l'échange qui a obtenu la plus forte valeur en règle générale.

Deux mécanismes peuvent s'ajouter pour améliorer les performances de cet algorithme :

- dans la phase d'intensification : la fourmi conserve sa solution pour la génération suivante si cette solution a été améliorée pendant la phase de modification. Cette phase sera déclenchée si au moins une fourmi a modifié la meilleure solution pendant l'itération ;
- dans la phase de diversification : la solution modifiée sera conservée pour l'itération suivante même si elle a vu sa qualité diminuer. Cette phase est déclenchée si la meilleure solution n'a pas été améliorée depuis un nombre fixé d'itérations, et elle commence par la réinitialisation de la phéromone.

5.4 Notre méthode hybride : ANTabu

ANTabu [180, 182, 223, 224] est une hybridation entre un OCF et une recherche locale ; il s'agit dans notre cas d'une recherche tabou [95]. Cette méthode fait partie des OCF et s'inpire pour la construction des nouvelles solutions de la méthode HAS-QAP [86]. En effet, notre méthode, tout comme HAS-QAP, modifie une solution existante, en appliquant des échanges entre objets dans la solution de manière stochastique (pour le choix du premier) et en se basant sur la phéromone (pour le choix du second). Pour ce modèle, les OCF ne sont pas vraiment respectés puisque notre méthode est pertubatrice et non pas constructrice.

Dans ANTabu, toutes les fourmis sont utilisées pour mettre à jour la phéromone quand elles ont terminé la construction (modification) de leur nouvelle solution et ont appliqué la recherche locale.

Nous avons incorporé une puissante fonction de diversification qui permet de déplacer les fourmis dans des zones de l'espace de recherche qui n'ont pas été explorées.

Dans la section suivante, nous allons décrire successivement les différentes particularités de ANTabu, ainsi que le modèle de parallélisation que nous avons développé.

5.4.1 Les éléments clés de ANTabu

Pour ANTabu, les solutions manipulées le sont sous la forme de permutations d'entiers. Pour situer les éléments essentiels de ANTabu, nous nous référons au squelette de l'application (alg 5.1). Il est intéressant de s'attarder sur les quatre éléments principaux que sont la modification des solutions, la recherche locale, la mise à jour de la phéromone et la diversification.

Algorithme 5.1: Algorithme simplifié de l'ANTabu.
ANTabu()

(1) **initialisation** Génération aléatoire d'une population initiale
(2) Appliquer la recherche tabou
(3) Choisir la meilleure solution π^*
(4) Initialiser la matrice de phéromone en utilisant la qualité de π^*
(5) **Répéter** pour chaque fourmi
(6) Modifier la solution en suivant la phéromone (π_m^k)
(7) Appliquer la méthode tabou (π_o^k)
(8) Mise à jour de la phéromone
(9) **si** nécessaire **alors** Diversifier
(10) **Jusqu'**a ce qu'un critère d'arrêt soit vérifié

La construction et la modification des solutions

Dans cette construction, on retrouve une procédure similaire de modification des solutions à HAS-QAP. Dans ANTabu, chaque fourmi va appliquer un nombre m ($m = n/3$, avec n le nombre d'objets du problème) de modifications de la solution de départ basées sur la phéromone. Cette opération a pour but de faire apparaître de bonnes affectations dans la solution manipulée, en utilisant la phéromone. Les affectations (couples) faisant partie de la solution doivent ressortir comme celles ayant le plus fort taux de phéromone. Comme l'expliquent Gambardella *et al.* [86], l'action de ces manipulations a un effet d'exploration entraînant la diminution de la valeur de qualité associée à la solution (fig. 5.4), mais cette diminution est ensuite souvent compensée par l'application de la recherche locale.

La recherche locale

Les méta-heuristiques intègrent généralement des méthodes de recherche locale. Cela s'explique par la constatation suivante : comme l'ont montré Johnson et McGeorch [120] en répétant des recherches locales sur des solutions générées aléatoirement, on obtient de bons résultats. Si l'on part de solutions qui sont déjà proches de l'optimum, on devrait trouver la solution plus rapidement, et c'est là que se situe le rôle de la méta-heuristique qui doit générer de bonnes solutions pour la recherche locale.

Pour notre algorithme, nous avons choisi une recherche tabou. Ce choix s'explique par le fait que cette méthode est très perfomante pour le PAQ et que son utilisation nous pemettra également de comparer les résultats obtenus à ceux d'un tabou parallèle, afin de démontrer l'intérêt de la coopération face à la force brute.

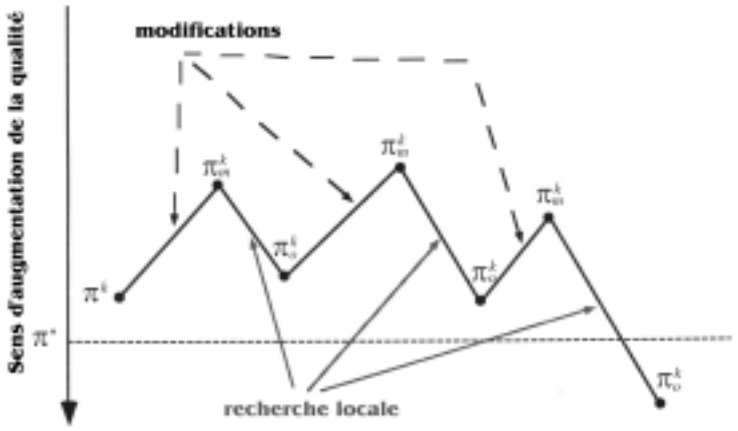


FIG. 5.4 – Exemple d'évolution de la qualité avec une méthode pour le PAQ adoptant une stratégie de modification des solutions. π^k indique la solution de la fourmi k, π^k_m est la solution modifiée et π^k_o celle obtenue après la recherche locale.

La liste tabou représente une fonction de mémoire à court terme, puisqu'elle ne mémorise que les derniers mouvements effectués.

L'algorithme 5.2 représente la forme la plus simple de la recherche tabou.

Algorithme 5.2: La forme la plus simple de la recherche tabou.
TABU()

(1) choisir une solution initiale s
(2) **initialisation**
(3) $s^* \leftarrow s$, s^* représente la meilleure solution obtenue jusque là
(4) $k \leftarrow 0$, k est le compteur d'itérations effectuées depuis la dernière amélioration de s^*
(5) $T \leftarrow \emptyset$, T est la liste tabou
(6) **tant que** le critère d'arrêt n'est pas vérifié **faire**
(7) **si** $N_s - T \neq \emptyset$
(8) **alors** il existe au moins un mouvement effectuable à partir de s et non Tabou
(9) $k \leftarrow k + 1$
(10) Rechercher $s' \in (N_s - T)$ telle que $s' = \min_{x \in (N_s - T)} f(x)$
(11) $s \leftarrow s'$
(12) **si** $f(s) < f(s^\star)$
(13) **alors** $s^\star \leftarrow s$, $k \leftarrow 0$
(14) Mettre à jour la liste T

On peut adjoindre à la recherche tabou un critère d'aspiration. En effet, l'utilisation de la liste tabou peut sembler un peu restrictive et empêcher dans certains cas de choisir de bonnes solutions. C'est pourquoi, on autorise le choix d'un mouvement faisant partie de la liste tabou s'il permet d'obtenir une solution dont la fonction coût dépasse un certain seuil. Ce critère est le plus souvent utilisé sous sa forme réduite. Celle-ci consiste

à ne prendre un mouvement de la liste que s'il améliore la valeur de la meilleure solution trouvée jusqu'alors.

La mise à jour de la matrice de phéromone

Dans ANTabu, à la différence de la plupart des OCF, toutes les fourmis participent à la mise à jour de la matrice de phéromone. Ce choix sera justifié expérimentalement dans le chapitre suivant. Pour chaque fourmi, sa contribution à la matrice de phéromone est proportionnelle à la qualité de la solution trouvée. Cet apport est pondéré par le quotient résultant de la division de la différence entre la qualité de la plus mauvaise solution trouvée et de la solution trouvée par cette fourmi, par la qualité de la meilleure solution.

La formule correspondant à cette mise à jour est la suivante :

$$\tau_i = (1-\alpha)\tau_{i-1} + \frac{\alpha}{f(\pi)} \cdot \frac{f(\pi^-) - f(\pi)}{f(\pi^*)}$$

- τ_i est le taux de phéromone à la i-ème itération ;
- π^- désigne la plus mauvaise solution trouvée ;
- π^* désigne la meilleure solution trouvée ;
- π désigne la solution courante ;
- α est une constante comprise entre 0 et 1.

Le cas où seule la meilleure fourmi est prise en considération pour la modification de la phéromone a été étudié également : nous présenterons les comparaisons dans le chapitre sur les résultats.

De plus, tout comme dans le $\mathcal{MM}$AS, deux valeurs limitent les variations du taux de phéromone : il s'agit d'une borne min et d'une borne max. Dans notre cas : $\tau_{max} = 10$ et $\tau_{min} = 2$. Ces deux facteurs (déf. d'une borne min et d'une borne max et participation de toutes les fourmis à la mise à jour de la matrice de phéromone) évitent la stagnation au niveau d'optima locaux en favorisant la diversité des solutions générées.

La fonction de diversification

La diversification est différente de celle utilisée dans HAS-QAP car celle-ci consistait à générer aléatoirement un nouvel ensemble de solutions initiales pour continuer la recherche.

Le but de notre diversification est de permettre aux fourmis une plus grande exploration, et plus particulièrement d'explorer les zones de l'espace des solutions qui ne l'ont pas encore été. Pour ce faire, une nouvelle matrice a été créée, appelée matrice de fréquences. Cette matrice représente la fréquence d'apparition de chaque affectation dans l'ensemble des solutions. Ces solutions ont été fournies par les fourmis lors des itérations précédentes. Cette matrice est utilisée lors de la diversification, pour générer des solutions comportant les affectations qui ont été les moins choisies. Lors de la génération de la nouvelle solution, on parcourt la matrice de fréquences en recherchant les affectations qui ont été les moins utilisées. Afin d'obtenir des solutions différentes, l'affectation sera choisie avec une équi-probabilité parmi la sélection.

Cette diversification est appliquée si aucune amélioration de la meilleure solution ne s'est produite depuis $n/2$ itérations (n étant le nombre d'objets). Un exemple de déroulement de ANTabu est présenté dans la section suivante.

5.4.2 Exemple de déroulement pour ANTabu

Cet exemple repose sur un jeu de données qui a été créé pour cette occasion.
Les matrices qui définissent le problème pour notre exemple sont les suivantes :

1. D est la matrice des distances et d_{ij} représente un de ses éléments ;
2. C est la matrice de coûts des flux et c_{kl} représente un de ses éléments.

$$D = \begin{pmatrix} 0 & 7 & 1 \\ 7 & 0 & 4 \\ 1 & 4 & 0 \end{pmatrix} \text{ et } C = \begin{pmatrix} 0 & 2 & 9 \\ 2 & 0 & 6 \\ 9 & 6 & 0 \end{pmatrix}$$

D'abord, on génère une solution aléatoire qui va permettre d'initialiser la matrice de phéromone avec τ_0 (valeur arbitraire), comme suit :

$$\pi^0 \ : \ \boxed{3}\boxed{2}\boxed{1}$$

Avec

$$f(\pi^0) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}c_{\pi_i\pi_j} = 118$$

D'où la matrice de phéromone définie arbitrairement :

$$F = \begin{pmatrix} \tau_o & \tau_o & \tau_o \\ \tau_o & \tau_o & \tau_o \\ \tau_o & \tau_o & \tau_o \end{pmatrix} \text{ avec } \tau_0 = \frac{1}{100 \times f(\pi^0)}$$

On génère une solution initiale aléatoire pour chaque fourmi ; dans notre exemple on se limite à deux fourmis a et b.

$$\pi^{a_0} \ : \ \boxed{3}\boxed{1}\boxed{2} \quad \pi^{b_0} \ : \ \boxed{2}\boxed{3}\boxed{1}$$

où π^{x_n} désigne la n-ième itération pour la fourmi x

Chaque individu entre dans la phase de recherche qui comporte imax itérations. Il effectue alors $n/3$ échanges dans la solution, le choix s'effectuant comme suit :
On choisit un index r au hasard, puis un second s avec une probabilité proportionnelle aux valeurs de la trace de phéromone :

$$\frac{\tau^k_{r\pi_s} + \tau^k_{s\pi_r}}{\sum_{r\neq s}(\tau^k_{r\pi_s} + \tau^k_{s\pi_r})}$$

$\tau^k_{r\pi_s}$ désigne le taux de phéromone pour la solution k qui correspond à l'affectation de l'objet qui était contenu à la position s de π à la position r

Dans notre exemple, on n'effectue qu'un seul échange ($n/3$) :

- Pour la fourmi a $r = 2$
  $s = 1$

$$\pi^{a_1} : \boxed{1}\boxed{3}\boxed{2}$$

- Pour la fourmi b $r = 3$
  $s = 2$

$$\pi^{b_1} : \boxed{2}\boxed{1}\boxed{3}$$

Puis on effectue une optimisation locale (tabou), qui n'est pas illustrée dans cet exemple. Ensuite la fonction coût des deux solutions est calculée :

- $f(\pi_{a_1}) = 178$
- $f(\pi_{b_1}) = 112$

Les résultats sont centralisés.

Si aucune fourmi ne fournit de solution améliorant π^*, la solution précédente est à nouveau utilisée pour l'itération suivante. Ce mécanisme sera répété à chaque itération jusqu'au moment où une au moins des solutions fournies sera meilleure que π^* ; sinon chaque fourmi prendra sa nouvelle solution comme point de départ pour la suite.

Dans le cas où une des solutions est meilleure que π^*, elle la remplace. On met à jour la matrice de phéromone avec toutes les solutions trouvées :

$$F = \begin{pmatrix} \tau'_0 + \tau_b & \tau'_0 + \tau_a & \tau'_0 \\ \tau'_0 + \tau_a & \tau'_0 & \tau'_0 + \tau_b \\ \tau'_0 & \tau'_0 + \tau_b & \tau'_0 + \tau_a \end{pmatrix}$$

avec

$$\tau'_0 = (1 - \alpha)\tau = 7,5 \cdot 10^{-5}$$

$$\tau_a = \frac{\alpha}{f(\pi^{a_1})} \cdot \frac{f(\pi^-) - f(\pi^{a_1})}{f(\pi^*)} = 0$$

$$\tau_b = \frac{\alpha}{f(\pi^{b_1})} \cdot \frac{f(\pi^-) - f(\pi^{b_1})}{f(\pi^*)} = 5,26 \cdot 10^{-4}$$

Ici :

- π^- désigne π^{b_1}
- π^* désigne π^{b_1}

Enfin, si $n/2$ itérations sont exécutées sans que π^* soit modifiée, on effectue une diversification qui consiste à repartir d'un ensemble de nouvelles solutions comprenant π^* , la matrice F est réinitialisée à l'aide de π^*, ce qui termine l'itération courante.

Après la première itération, la matrice de fréquences a la forme :

$$D = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Si une diversification est effectuée à ce stade, la seule solution générée sera :

$$\boxed{3}\boxed{2}\boxed{1}$$

5.4.3 Implantation parallèle

Afin de gagner en efficacité, notre algorithme a été implanté sous la plate-forme PVM, celle-ci permettant d'exploiter la puissance de calcul des réseaux actuels.

PVM

PVM 3 est un logiciel qui permet d'utiliser un réseau de machines UNIX hétérogènes comme s'il n'était qu'un ensemble de processus communicants. Ainsi, de grands problèmes de calculs peuvent être résolus par l'utilisation d'agrégation de plusieurs ordinateurs.

Le développement de PVM a débuté pendant l'été 1989, au Oak Ridge National Laboratory et est maintenant un projet de recherche en cours. Ce logiciel est distribué gratuitement, dans le but de "faire avancer la science", précise l'Oak Ridge National Laboratory.

Sous PVM, un utilisateur définit une collection d'ordinateurs séquentiels, parallèles ou vectoriels qui apparaîssent comme un grand ordinateur à mémoire distribuée. Le terme de machine virtuelle désigne cet ordinateur logique à mémoire distribuée, et un hôte (ou host) désigne un des ordinateurs membres de la collection.

PVM fournit la fonction qui démarre automatiquement une tâche dans la machine virtuelle et lui permet de communiquer et de se synchroniser avec les autres tâches déjà présentes.

Une tâche est définie comme une unité d'exécution, comme le sont les processus UNIX. C'est souvent un processus UNIX, mais pas nécessairement.

Les applications, qui peuvent être écrites en Fortran 77 ou en C, peuvent être parallélisées par le système de passage de messages. C'est une construction courante pour beaucoup d'ordinateurs à mémoire distribuée.

Par l'envoi et la réception de messages, les multiples tâches d'une application peuvent coopérer et se synchroniser pour résoudre un problème en parallèle.

PVM supporte l'hétérogénéité dans l'application, les machines et les réseaux. PVM permet la disposition de tâches suivant l'architecture la plus adaptée à la solution choisie afin d'optimiser les résultats. PVM s'occupe de toutes les conversions de données qui peuvent être requises si deux ordinateurs utilisent des représentations différentes d'entiers ou de réels (empaquetage XDR). PVM autorise la machine virtuelle à être interconnectée par une variété de réseaux différents.

Le système PVM est composé de deux parties :

- la première partie est un démon, qui réside sur tous les ordinateurs composant la machine virtuelle. Quand un utilisateur veut démarrer une application PVM, il crée d'abord une machine virtuelle en démarrant PVM. L'application PVM peut alors être démarrée depuis n'importe quel hôte. De multiples usagers peuvent configurer en même temps des machines virtuelles sur les mêmes hôtes, et chaque usager peut exécuter plusieurs applications PVM simultanément.
- La seconde partie du système est une librairie contenant les routines de l'interface PVM (libpvm3.a). Cette librairie contient des routines auxquelles l'utilisateur peut faire appel pour le passage de messages, pour la création des processus, pour synchroniser des tâches, et pour configurer ou modifier la machine virtuelle. Les programmes d'application doivent être liés avec cette librairie pour utiliser PVM.

Modèle utilisé

Si on regarde la forme générale des OCF (3.1), on s'aperçoit que les actions des fourmis sont indépendantes et donc intrinsèquement parallélisables. Plusieurs stratégies

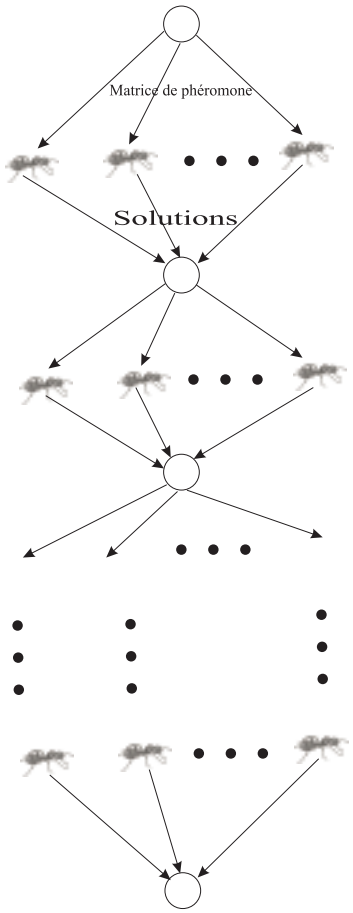


FIG. 5.5 – Echange d'informations pour l'ANTabu.

ont d'ailleurs été proposées comme dans [27], où les auteurs discutent de l'intérêt de ne mettre à jour la matrice de phéromone qu'après plusieurs itérations de la fourmi, afin de diminuer les communications sur le réseau. Notre choix pour ANTabu s'est porté sur une mise à jour après toutes les itérations (cf. fig. 5.5), le surcoût des échanges étant assez faible.

Le schéma de parallélisation est présenté dans la figure 5.6.

L'algorithme ANTabu possède un parallélisme intrinsèque. La parallélisation consiste à faire exécuter la boucle principale par des processus indépendants (les fourmis) qui ne communiquent entre eux que par l'intermédiaire de la mémoire(modèle maître/esclaves), celle-ci se traduisant dans notre cas par la matrice de phéromone. Le nombre réduit de mises à jour de la matrice permet de limiter les échanges entre le maître qui gère la mémoire et les travailleurs qui effectuent l'optimisation ; le passage d'informations n'étant effectué qu'au moment de la modification de la matrice. Le maître reçoit les solutions des travailleurs, et applique la formule de mise à jour, puis renvoie la matrice modifiée.

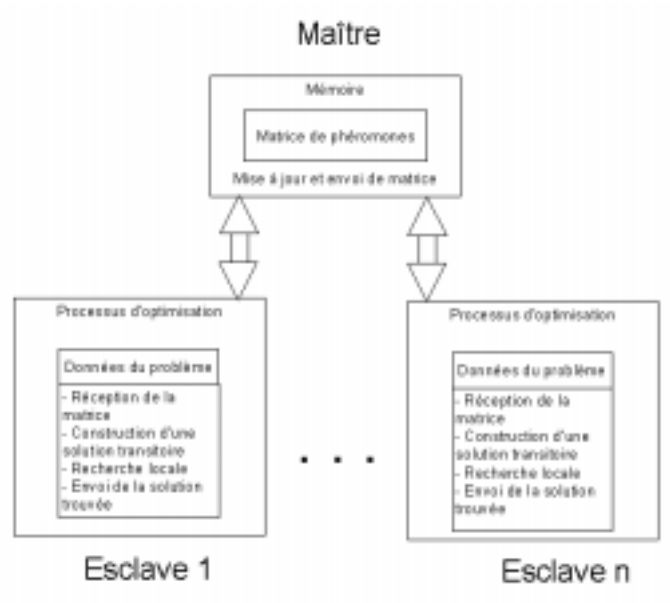


FIG. 5.6 – Modèle parallèle de ANTabu.

5.5 Conclusion

Le problème de l'affectation quadratique est connu pour être extrêmement difficile à résoudre. Comme nous l'avons vu, beaucoup de méthodes ont été utilisées pour tenter de résoudre ces instances avec plus ou moins de réussite. Cette réussite est souvent liée à la nature même des instances, puisque c'est en fonction de celles-ci que les heuristiques sont plus ou moins performantes. Il est d'ailleurs intéressant de noter que ce découpage n'est pas suffisant : une heuristique très performante sur un groupe d'instances peut être très décevante sur d'autre instances. Pour pallier cette incertitude plusieurs mesures ont été introduites : parmi celles-ci on peut citer les mesures statistiques et topologiques sur le paysage du PAQ proposées par BACHELET et TALBI [3, 220].

Les méthodes se basant sur les OCF obtiennent de très bons résultats pour le PAQ. C'est dans ce contexte que nous avons développé notre méthode hybridant les fourmis avec une recherche locale performante : la recherche tabou.

Dans le chapitre suivant, nous allons aborder les résultats obtenus avec notre algorithme et nous les confronterons à ceux des autres méthodes.

Chapitre 6

Etude expérimentale d'ANTabu

Ce chapitre est consacré aux résultats expérimentaux obtenus avec ANTabu. Dans un premier temps, ils seront utilisés pour démontrer l'intérêt de cette méthode et des choix effectués pour ses différents constituants. Ensuite, nous regarderons ces résultats comparativement à ceux obtenus par d'autres méthodes utilisées pour la résolution du PAQ.

6.1 Introduction

Dans ce chapitre nous introduisons un certain nombre de termes caractérisant la topologie des instances du PAQ. Voici une brève définition de ces termes :

- le flot de dominance (ou dominance de flux) : il est utilisé pour indiquer la complexité d'une instance suivant la nature de la matrice des coûts de flux (il est égal à 100 fois l'écart type divisé par la moyenne des flux) ;
- une instance régulière ou peu structurée : c'est classiquement une instance qui a un flot de dominance supérieur à 1,2. Son paysage est plutôt plat et rugueux (fig 6.1) ;
- une instance irrégulière ou structurée : est celle qui a un flot de dominance inférieur à 1,2. Son paysage est constitué de régions de grande taille à des niveaux bien distincts
- la distance de dominance : elle a été proposée pour tenter d'affiner la mesure de complexité des instances ; l'équivalent du flot de dominance appliqué à la matrice de distance a été introduit.

Pour toutes les comparaisons, nous avons utilisé ANTabu avec une population comprenant 10 fourmis. Toutes les instances sont extraites de la QAPlib[1], et la sélection des instances présentées dans ce chapitre est celle de l'article de Gambardella *et al.* [86].

Dans les comparaisons, nous nous sommes attachés au fait que l'objectif d'un tel algorithme est d'obtenir un résultat proche de l'optimum dans un laps de temps le plus court possible.

[1] http ://www.opt.math.tu-graz.ac.at/qaplib/

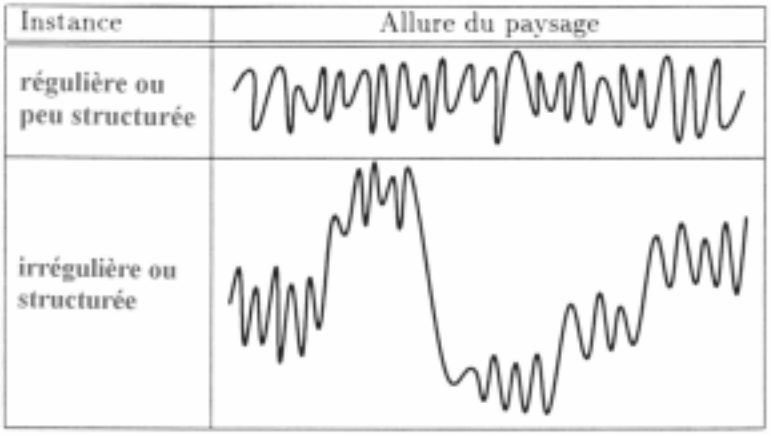


FIG. 6.1 – Illustration de la notion de structure d'une instance par forme de son paysage

6.2 Validation des choix pour ANTabu

Dans cette section, nous allons tenter de montrer l'intérêt des modifications que nous avons apportées par rapport au modèle général des OCF pour le PAQ. Après avoir examiné les performances de notre algorithme pour de petites exécutions, nous discuterons plus particulièrement de l'intérêt de notre fonction de mise à jour de la phéromone et de notre diversification.

6.2.1 ANTabu pour de courtes exécutions

12 problèmes de la QAPlib [29] ont été sélectionnés. Certains appartiennent à la classe des problèmes irréguliers (`bur26d`, `bur26b`, `chr25a`, `els19`, `kra30a`, `tai20b`, `tai35b`), d'autres à la classe des problèmes réguliers (`nug30`, `sko42`, `sko64`, `tai25a` et `wil50`).

Les paramètres utilisés pour l'évaluation sont : 10 itérations pour chacune des 10 fourmis, avec une méthode de recherche tabou qui est limitée à $5n$ itérations où n désigne la taille du problème considéré, et un coefficient d'évaporation de $\alpha = 0.1$.

Pour cette série d'instances (tableau 6.1), l'algorithme obtient de bonnes performances malgré un nombre d'itérations relativement faible, puisque l'on ne fait que $10 \cdot (5n+1)$ évaluations. On peut voir que le taux d'erreur pour cette série représentative d'instances des classes régulières et irrégulières est toujours très inférieur à 1%, voire même inférieur à $0,1\%$ pour une majorité.

6.2.2 Validation du paramétrage du système de fourmis

ANTabu se distingue principalement de la plupart des systèmes de fourmis par sa diversification et par le renforcement de la phéromone. Quel est donc l'apport de ces modifications ?

Nous allons examiner successivement le cas de la diversification et celui de la mise à jour de la phéromone.

TAB. 6.1 – Résultats de l'ANTabu sur de petites instances et de courtes exécutions (10 itérations). Pour chaque instance, le résultat est présenté sous la forme d'un écart vis-à-vis de la meilleure solution connue et exprimée en pourcentage

Nom du problème	Meilleure solution connue	ANTabu (%)
bur26b	3817852	0,018
bur26d	3821225	0,0002
chr25a	3796	0,047
els19	17212548	0
kra30a	88900	0,208
tai20b	122455319	0
tai35b	283315445	0,1333
nug30a	6124	0,029
sko42	15812	0,076
sko64	48498	0,156
tai25a	1167256	0,843
wil50	48816	0,66

La diversification :

Pour valider la fonction de diversification, nous l'avons comparée à la stratégie de base qui consiste à générer de nouvelles solutions aléatoires. Pour cela, nous avons comparé les résultats obtenus avec notre méthode de diversification face à cette méthode de génération aléatoire.

Trente trois instances comprises entre 30 et 56 ont été utilisées pour la comparaison : esc32a, esc32b, esc32c, esc32d, esc32e, esc32f, esc32g, esc32h, kra30a, kra30b, lipa30a, lipa30b, lipa40a, lipa50a, lipa50b, nug30, sko42, sko49, sko56, ste36a, ste36b, ste36c, tai30a, tai30b, tai35a, tai35b, tai40a, tai40b, tai50a, tai50b, tho30, tho40, et wil50. Les deux programmes commencent à la première diversification avec les mêmes solutions ; ensuite pour ANTabu est appliquée la fonction de diversification dans les zones non explorées (*cf.* 5.4.1), tandis que pour l'autre méthode c'est la génération aléatoire qui est utilisée.

Les résultats considérés sont les moyennes calculées sur trente exécutions (6.2). Les résultats sont supérieurs ou égaux à la génération aléatoire dans 85% des cas. Dans seulement 15% des cas, on assiste à une dégradation des performances. Les contre-performances de notre méthode sont pour les instances irrégulières : bur26b, kroa30a, tai30b, tai35b, tai40b.

Pour tenter d'expliquer cette différence de performances entre la résolution des instances régulières et irrégulières, nous pouvons examiner les résultats obtenus par ANTabu pour 20 instances irrégulières de taille comprise entre 19 et 80.

Dans le tableau 6.3, sont présentées en plus de la moyenne obtenue pour les instances irrégulières, la valeur de la meilleure solution et de la plus mauvaise. On peut remarquer que dans pratiquement tous les cas la meilleure solution est au moins trouvée une fois sur les 10 exécutions. On peut donc émettre l'hypothèse que les mauvaises performances

TAB. 6.2 – Comparaison entre une diversification aléatoire et notre diversification. Seules sont présentées les statistiques trouvées lors de la comparaison. Les temps de calcul sont donnés dans le tableau 6.8

	nombre d'instances	proportion (pourcentage)
Résultat meilleur que la version aléatoire	8	24,24
Résultat égal à la version aléatoire	20	60,61
Résultat inférieur à la version aléatoire	5	15,15

moyennes sont dues à la présence de nombreux optima locaux, regroupés sous la forme de massifs. Dans ce cas précis, la diversification présentée aurait plutôt tendance à placer les nouvelles solutions loin de ces zones ; cela est sans doute le prix à payer pour une bonne exploration de l'espace de recherche.

Pour pallier ce défaut, il serait intéressant d'avoir une mesure qui adapterait la fonction de diversification à la nature de l'instance. Pour cela nous avons examiné trois mesures classiques qui sont la taille du problème, le flot de dominance et la distance de dominance. Pour quantifier l'intérêt de ces mesures, nous avons regardé s'il y avait une corrélation entre les moyennes obtenues sur les 33 instances précédemment citées et ces mesures (fig. 6.2). Il est clair que ces valeurs ne semblent pas donner une indication sur la performance de ANTabu.

La mise à jour de la phéromone :

Nous avons comparé une version de référence (où seule une fourmi participe au renforcement et où la mise à jour de la table est effectuée par la formule classique introduite dans HAS-QAP (*cf.* 5.3.7)) avec deux versions :

- une version utilisant toutes les fourmis et la mise à jour de HAS-QAP ;
- une version utilisant une fourmi et utilisant notre formule de mise à jour.

Pour cette comparaison, nous avons étudié 25 instances de la QAPlib ayant une taille comprise entre 30 et 40 : `kra30a`, `kra30b`, `lipa30a`, `lipa30b`, `lipa40a`, `lipa50a`, `lipa50b`, `nug30`, `sko42`, `sko49`, `sko56`, `ste36a`, `ste36b`, `ste36c`, `tai30a`, `tai30b`, `tai35a`, `tai35b`, `tai40a`, `tai40b`, `tai50a`, `tai50b`, `tho30`, `tho40`, et `wil50`.

Les statistiques présentées ont été obtenues sur la base de la moyenne des résultats de trente exécutions par instance. Les résultats sont présentés dans le tableau 6.4.

En ce qui concerne la participation de toutes les fourmis, les résultats sont supérieurs ou égaux dans 88% des cas. Pour la formule de mise à jour, les résultats sont supérieurs dans 80%.

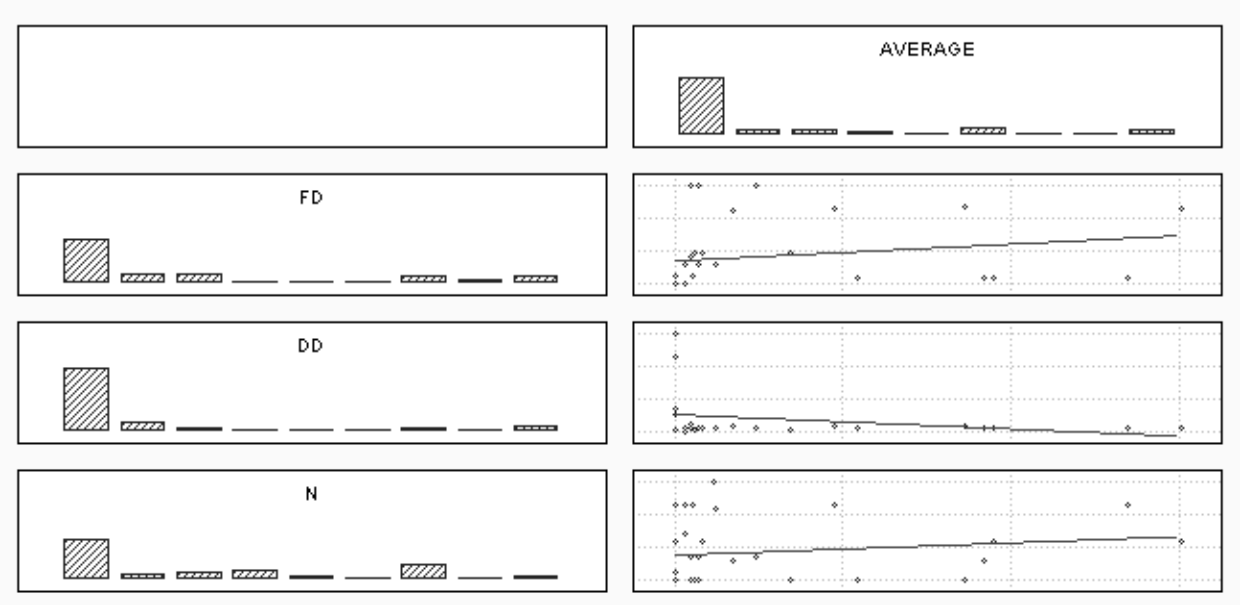
Les résultats obtenus valident totalement ces choix effectués. En effet, dans la grande majorité des cas, soit les résultats obtenus ont été améliorés, soit la meilleure solution a déjà été trouvée.

TAB. 6.3 – Performances de ANTabu sur des instances irrégulières. Les valeurs présentées sont celles de la meilleure et de la plus mauvaise solution trouvée pour chaque instance. Ces valeurs ont été obtenues pour dix exécutions. *Les valeurs donnent l'écart par rapport à la meilleure solution connue. La meilleure solution est toujours trouvée sauf dans un cas.*

Instance	Moyenne	Secondes	Plus mauvaise solution trouvée	Meilleure solution trouvée
bur26a	0,0000	50	0,0000	**0,0000**
bur26b	0,0169	50	0,1693	**0,0000**
bur26c	0,0000	50	0,0000	**0,0000**
bur26d	0,0000	50	0,0000	**0,0000**
bur26e	0,0000	50	0,0000	**0,0000**
bur26f	0,0000	50	0,0000	**0,0000**
bur26g	0,0000	50	0,0000	**0,0000**
bur26h	0,0000	50	0,0000	**0,0000**
chr25a	0,8957	40	4,5838	**0,0000**
els19	0,0000	20	0,0000	**0,0000**
kra30a	0,2677	76	1,3386	**0,0000**
kra30b	0,0000	86	0,0000	**0,0000**
tai20b	0,0000	27	0,0000	**0,0000**
tai25b	0,0000	50	0,0000	**0,0000**
tai30b	0,0000	90	0,0000	**0,0000**
tai35b	0,0408	147	0,2212	**0,0000**
tai40b	0,4640	240	2,6239	**0,0000**
tai50b	0,2531	480	0,9075	**0,0000**
tai60b	0,2752	855	1,5608	**0,0000**
tai80b	0,7185	2073	1,8549	0,0019

TAB. 6.4 – Comparaison des deux versions de l'application comportant chacune une des deux modifications avec une version qui en est dépourvue. Seules sont présentées les statistiques trouvées lors de la comparaison

	Toutes les fourmis		Notre formule	
	nombre d'instances	proportion (pourcentage)	nombre d'instances	proportion (pourcentage)
Résultat meilleur que la version de référence	11	44,0	9	36,0
Résultat égal à la version de référence	11	44,0	11	44,0
Résultat inférieur à la version de référence	3	12,0	5	20,0



Les résultats présentés sont les coefficients de corrélations

	Moyenne de ANTabu
Taille du problème	0,17
Flot de dominance	0,21
Distance dominance	-0,23

FIG. 6.2 – Corrélations entre les performances de ANTabu et la taille, le flot de dominance et la distance dominance des instances

TAB. 6.5 – Comparaison des résultats du PATS et de l'ANTabu. Les meilleurs résultats sont en **gras.**

	tai100a	sko100a	sko100b	sko100c	sko100d	wil100
QAPlib						
Meilleure solution connue	21125314	152002	153890	147862	149576	273038
PATS						
Meilleure solution trouvée	21193246	152036	153914	147862	149610	273074
Gap	0,322	0,022	0,016	0	0,023	0,013
Durée (en mn)	117	142	155	132	152	389
ANTabu						
Meilleure solution trouvée	21184062	152002	153890	147862	149578	273054
Gap	**0,278**	**0**	**0**	**0**	**0,001**	**0,006**
Durée (en mn)	139	137	139	137	201	139

6.3 Coopération contre force brute

Nous avons confronté nos résultats à ceux obtenus avec PATS afin de valider le choix de la coopération [222] et de vérifier si tous les bons résultats obtenus ne sont pas simplement dus à la présence de la recherche locale (la recherche tabou).

PATS consiste en un ensemble de Tabous tournant de manière distribuée sur un réseau de stations de travail (incluant des PC Intel, des stations Sun et Alpha). La charge de ces stations est mesurée et les tabous sont automatiquement placés sur les machines qui sont inutilisées. Lorsque les machines ne sont plus disponibles les recherches tabous sont alors retirées par l'intermédiaire de la plate-forme MARS [109].

Pour cette comparaison, nous avons étudié des instances appartenant à deux classes de problèmes :

- une instance avec des matrices de distances et de flux uniformes : `tai100a` ;
- un ensemble d'instances avec une matrice de flux aléatoires : `sko100a`, `sko100b`, `sko100c`, `sko100d`, `wil100`.

Parmi les instances proposées dans [222], nous avons pris le parti de ne nous intéresser qu'aux problèmes de grande taille dont PATS n'a pas réussi à trouver la meilleure solution. Les résultats obtenus par ces deux méthodes sur ces instances sont présentés dans le tableau 6.5.

La meilleure solution trouvée pour PATS et ANTabu est la meilleure solution pour dix exécutions. La durée est donnée en minutes, et correspond à la moyenne pour les dix exécutions sur un réseau de 126 machines pour PATS et sur un réseau de 10 machines (SGI INDY) pour ANTabu.

L'implantation de PATS sur une plate-forme adaptative rend difficile la mesure de la puissance disponible. En effet, celle-ci dépend de la charge des machines et de leur occupation. Même si l'on considère qu'en moyenne la moitié de la capacité totale était

disponible, on remarque bien que la puissance de calcul devrait favoriser PATS face à ANTabu qui n'utilise que 10 machines.

Les résultats montrent que ANTabu trouve de meilleurs résultats tout en utilisant moins de ressources de calcul (*cf.* configuration décrite dans le paragraphe précédent). Notons également que la meilleure solution est trouvée pour trois des quatre instances **sko100**.

Pour le système PATS, les Tabous utilisent des solutions générées aléatoirement, qu'ils vont tenter d'améliorer tout au long de leurs exécutions. Les fourmis quant à elles utilisent également au départ des solutions aléatoires, puis mettent en commun leurs solutions de façon à trouver un "squelette" de bonnes solutions qui servira à modifier leur solution. Dans ces deux algorithmes, le paramétrage du tabou en lui-même est semblable.

Cette différence de performances ne peut s'expliquer que par l'utilisation de la coopération entre les fourmis et cela par l'intermédiaire de la matrice de phéromone.

6.4 Comparaisons avec ANTabu

Cette section se décompose en deux parties.

D'abord nous allons comparer ANTabu à HAS-QAP, ce dernier étant l'OCF le plus proche dans son comportement de ANTabu, ensuite nous examinerons plus précisément les résultats de notre algorithme face aux autres méthodes.

6.4.1 Comparaison avec HAS-QAP

Les paramètres utilisés sont identiques dans les deux cas. La recherche tabou dans ANTabu est limitée à $5n$ itérations (n étant la taille du problème). Cette limitation est mise en place afin d'avoir le même nombre d'évaluations dans la recherche locale pour les deux méthodes afin d'évaluer leurs performances. Pour cette comparaison, nous avons étudié les 12 instances suivantes :

- 5 instances irrégulières : `bur26d, chr25a, els19, tai20b`, et `tai35b` ;
- 5 instances régulières : `nug30a, sko42, sko64, tai25a`, et `wil50`.

Les résultats présentés correspondent aux moyennes obtenues pour dix exécutions des deux algorithmes. Les résultats de HAS-QAP sont directement issus de l'article [90]. Pour chaque instance, les résultats des deux algorithmes sont présentés sous la forme d'une moyenne des écarts vis à vis de la meilleure solution connue.

Pour les instances présentées dans le tableau 6.6, ANTabu obtient toujours de meilleurs résultats que HAS-QAP. On peut se demander si ces résultats ne sont pas dus uniquement à l'utilisation de la recherche tabou (la recherche tabou étant connue comme une méthode de recherche locale très puissante).

Toutefois, les résultats présentés dans la comparaison avec PATS nous permettent de conclure que ces bons résultats sont bien le produit de l'hybridation des deux méthodes et non pas le seul fruit de l'application de la recherche locale.

6.4.2 ANTabu face aux autres méthodes

Dans cette section nous comparons 6 algorithmes qui sont les suivants :

TAB. 6.6 – Comparaison des résultats du HAS-QAP et de l'ANTabu. Les meilleurs résultats sont en **gras.**

Instance	Meilleure solution connue	HAS-QAP	ANTabu
bur26b	3817852	0,106	**0,018**
bur26d	3821225	0,002	**0,0002**
chr25a	3796	15,69	**0,047**
els19	17212548	0,923	**0**
kra30a	88900	1,664	**0,208**
tai20b	122455319	0,243	**0**
tai35b	283315445	0,343	**0,1333**
nug30a	6124	0,565	**0,029**
sko42	15812	0,654	**0,076**
sko64	48498	0,504	**0,156**
tai25a	1167256	2,527	**0,843**
wil50	48816	0,211	**0,066**

- la recherche tabou réactive de BATTITI et TECCHIOLLI : RTS [11] ;
- la recherche tabou robuste de TAILLARD : TT [217] ;
- la méthode génétique hybride de FLEURENT et FERLAND : GH [75] ;
- le recuit recuit simulé de CONNOLLY : SA [38] ;
- le systeme de fourmi de GAMBARDELLA *et al.* : HAS-QAP [90] ;
- notre methode : ANTabu [182].

Taillard a montré dans [215] que les différentes méthodes n'ont pas les mêmes performances suivant qu'elles tentent de résoudre une instance régulière ou une instance irrégulière. Les instances qui ont un flot de dominance supérieur à 1,2 sont les instances régulières, les autres sont les instances irrégulières ou structurées.

Dans le cas des instances régulières qui sont présentées dans le tableau 6.7, ANTabu est la méthode qui trouve le plus souvent les meilleurs résultats.

Nous avons également observé les performances des différents algorithmes sur un jeu de 20 instances irrégulières qui comprennent entre 10 et 80 positions. Les résultats sont présentés dans le tableau 6.8. Pour ces instances, c'est HAS-QAP qui obtient les meilleures performances, il trouve la meilleure moyenne dans 16 cas, ANTabu ne la trouvant que dans 13 cas. Ces résultats un peu inférieurs sont certainement dus comme nous l'avons vu dans la section 6.2.2 à notre forte fonction de diversification ; fonction qui semble moins performante sur les instances irrégulières.

6.5 Conclusion

Comme les résultats présentés le montrent, ANTabu obtient de très bonnes performances. Les rares fois où notre méthode obtient une moyenne inférieure, si l'on regarde plus en détail, on peut remarquer que la meilleure solution est trouvée la plupart du temps. Cette contre-performance en moyenne peut s'expliquer par l'utilisation de

TAB. 6.7 – Comparaison des résultats pour des instances régulières avec une durée de calcul identique. *Les meilleurs résultats sont en caractères gras. Les valeurs représentent les écarts moyens avec la meilleure solution connue, obtenus pour 10 exécutions et sont exprimés en pourcentage.*

Nom de l'instance	n	Meilleure solution connue	TT	RTS	SA	GH	HAS-QAP	ANTabu	Secondes
nug20	20	2570	**0**	0,911	0,070	**0**	**0**	**0**	30
nug30	30	6124	0,032	0,872	0,121	0,007	0,098	**0**	83
sko42	42	15812	0,039	1,116	0,114	0,003	0,076	**0**	248
sko49	49	23386	0,062	0,978	0,133	0,040	0,141	**0,038**	415
sko56	56	34458	0,080	1,082	0,110	0,060	0,101	**0,002**	639
sko64	64	48498	0,064	0,861	0,095	0,092	0,129	**0,001**	974
sko72	72	66256	0,148	0,948	0,178	0,143	0,277	**0,074**	1415
sko81	81	90998	0,098	0,880	0,206	0,136	0,144	**0,048**	2041
sko90	90	115534	0,169	0,748	0,227	0,196	0,231	**0,105**	2825
tai20a	20	703482	0,211	0,246	0,716	0,268	0,675	**0**	26
tai25a	25	1167256	0,510	**0,345**	1,002	0,629	1,189	0,736	50
tai30a	30	1818146	0,340	0,286	0,907	0,439	1,311	**0,018**	87
tai35a	35	2422002	0,757	0,355	1,345	0,698	1,762	**0,215**	145
tai40a	40	3139370	1,006	0,623	1,307	0,884	1,989	**0,442**	224
tai50a	50	4941410	1,145	0,834	1,539	1,049	2,800	**0,781**	467
tai60a	60	7208572	1,270	**0,831**	1,395	1,159	3,070	0,919	820
tai80a	80	13557864	0,854	**0,467**	0,995	0,796	2,689	0,663	2045
wil50	50	48816	0,041	0,504	0,061	0,032	0,061	**0,008**	441

TAB. 6.8 – Comparaison des résultats pour des instances irrégulières avec une durée de calcul identique. *Les meilleurs résultats sont en caractères gras. Les valeurs représentent les écarts moyens avec la meilleure solution connue, obtenus pour 10 exécutions et sont exprimés en pourcentage.*

Nom de l'instance	n	Meilleure solution connue	TT	RTS	SA	GH	HAS-QAP	ANTabu	Secondes
bur26a	26	5426670	0,0004	-	0,1411	0,0120	**0**	**0**	50
bur26b	26	3817852	0,0032	-	0,1828	0,0219	**0**	0,0169	50
bur26c	26	5426795	0,0004	-	0,0742	**0**	**0**	**0**	50
bur26d	26	3821225	0,0015	-	0,0056	0,0002	**0**	**0**	50
bur26e	26	5386879	**0**	-	0,1238	**0**	**0**	**0**	50
bur26f	26	3782044	0,0007	-	0,1579	**0**	**0**	**0**	50
bur26g	26	10117172	0,0003	-	0,1688	**0**	**0**	**0**	50
bur26h	26	7098658	0,0027	-	0,1268	0,0003	**0**	**0**	50
chr25a	25	3796	6,9652	9,8894	12,4973	2,6923	3,0822	**0,8957**	40
els19	19	17212548	**0**	0,0899	18,5385	**0**	**0**	**0**	20
kra30a	30	88900	0,4702	2,0079	1,4657	**0,1338**	0,6299	0,2677	76
kra30b	30	91420	0,0591	0,7121	0,1947	0,0536	0,071	**0**	86
tai20b	20	122455319	**0**	-	6,7298	**0**	0,0905	**0**	27
tai25b	25	344355646	0,0072	-	1,1215	**0**	**0**	**0**	50
tai30b	30	637117113	0,0547	-	4,4075	0,0003	**0**	**0**	90
tai35b	35	283315445	0,1777	-	3,1746	0,1067	**0,0256**	0,0408	147
tai40b	40	637250948	0,2082	-	4,5646	0,2109	**0**	0,4640	240
tai50b	50	458821517	0,2943	-	0,8107	0,2142	**0,1916**	0,2531	480
tai60b	60	608215054	0,3904	-	2,1373	0,2905	**0,0483**	0,2752	855
tai80b	80	818415043	1,4354	-	1,4386	0,8286	**0,6670**	0,7185	2073

notre méthode de diversification basée sur une matrice de fréquence. En effet, dans certains cas, elle va permettre l'amélioration de la meilleure solution si celle-ci se trouve éloignée de la zone qui était explorée. Mais, dans d'autres cas, cette diversification risque d'éloigner les fourmis du massif de solutions. Notre étude a validé tous les choix que nous avons faits, tant pour la diversification, que pour la participation de toutes les fourmis à la mise à jour de la phéromone et pour le choix de la formule de cette mise à jour. Nous n'avons pas tenté de valider le choix de l'approche modificatrice de ANTabu, les bonnes performances de HAS-QAP et de ANTabu la validant totalement.

La comparaison avec PATS nous a permis de montrer que la coopération est plus profitable que l'utilisation de la force brute, et que les performances de ANTabu ne sont pas uniquement le fruit de la recherche tabou même si c'est une méthode de recherche locale puissante.

Lors de la publication en 1999 des résultats présentés ici, ANTabu était sans doute la meilleure heuristique connue pour le PAQ.

Troisième partie

Les fourmis et la programmation automatique

Chapitre 7

La programmation automatique

Dans ce chapitre nous allons présenter les différentes méthodes de programmation automatique, la plus connue la programmation génétique (PG) ayant été proposée par Koza. Nous introduisons ensuite une nouvelle méthode de génération automatique de programmes "Ant Programming" qui s'inspire du modèle des OCF.

7.1 Introduction

L'homme a toujours rêvé de construire une machine intelligente. En informatique, ce rêve peut se concrétiser sous la forme d'un programme capable d'accumuler des connaissances, de les analyser et d'en déduire de nouveaux comportements. Ces nouveaux comportements peuvent eux-mêmes prendre la forme de programmes. Sans aller jusqu'à la machine intelligente, il est intéressant d'avoir un outil qui soit capable de générer automatiquement de nouveaux programmes à l'aide d'une base d'exemples.

C'est le principe de la programmation automatique, qui recherche dans l'espace des programmes une solution répondant au mieux au problème posé, en utilisant les exemples pour évaluer la qualité des solutions rencontrées. L'espace de recherche des programmes possibles est défini principalement par la grammaire utilisée pour coder les programmes solutions. Cette grammaire est constituée d'instructions qui seront utilisées lors de la construction des solutions ; à chaque type de problème correspond un jeu d'instructions différent. Ainsi, pour une équation mathématique on aura des fonctions mathématiques, pour une équation logique des opérateurs logiques, etc. L'espace dans lequel la recherche est effectuée est donc délimité par deux facteurs, la taille de la grammaire considérée et la taille des solutions que l'on autorise. Plus ces valeurs sont réduites, plus l'espace sera petit et plus la recherche sera facilitée. Mais si l'espace est trop réduit, la solution recherchée peut avoir une qualité faible ; par contre s'il est trop grand la recherche risque d'être très longue. Les différentes techniques existantes forment ce que l'on appelle la programmation automatique et s'intègrent dans le cadre général de l'apprentissage ("Machine learning" [158]). À notre connaissance, les différents algorithmes de programmation automatique existants reposent sur le modèle darwinien de l'évolution.

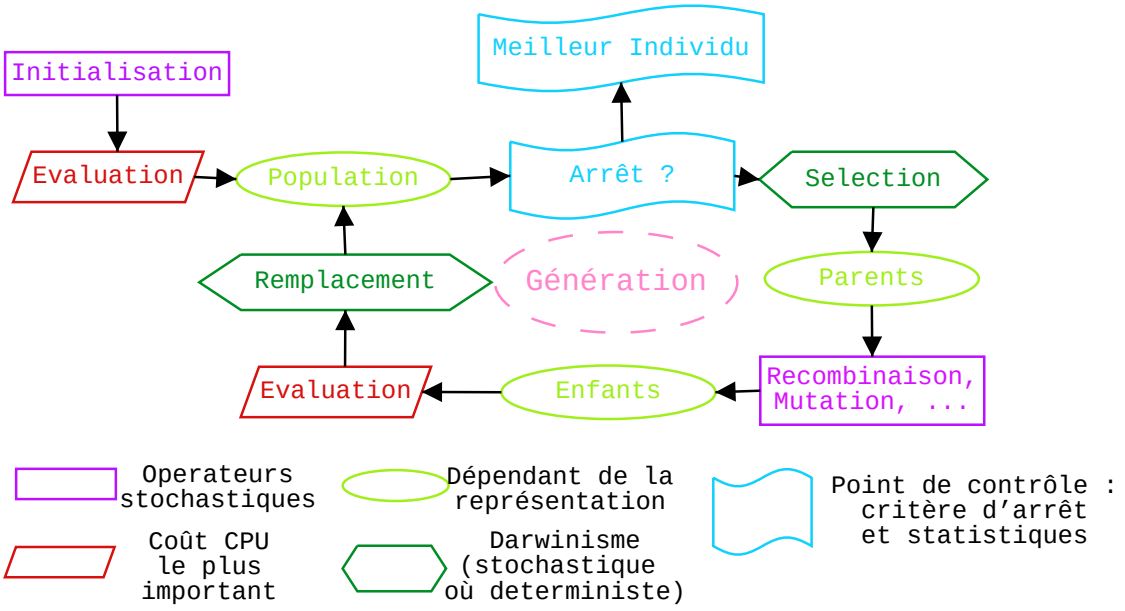


FIG. 7.1 – Schéma d'évolution de la programmation génétique
(Extrait du tutoriel d'EO, http ://geneura.ugr.es/~jmerelo/EO.orig.html)

7.2 La programmation automatique

Afin de resituer les algorithmes de programmation automatique, faisons un rapide retour sur le modèle d'évolution darwinien déjà évoqué dans le chapitre 2. Son schéma de fonctionnement (Fig 7.1) se décompose en 6 principaux points :

1. Génération des solutions de la première génération et évaluation
2. Sélection des individus les mieux adaptés
3. Application d'opérateurs génétiques (mutation, recombinaison, recopie, ...) sur les individus sélectionnés à l'étape 2
4. Calcul de la qualité des solutions
5. Remplacement des anciens individus
6. Retour à l'étape 2 si le critère d'arrêt n'est pas satisfait.

Les fonctions utilisées sont liées à la nature même des problèmes à résoudre, comme le montrent les exemples présentés dans le tableau 7.1.

La première méthode de programmation automatique reposant sur le concept d'évolution a été présentée par FOGEL dans les années 60 : elle a pour nom "evolutionnary programming" (EP) ou programmation évolutionnaire. Les programmes qui sont manipulés sont des automates finis [78, 77]. Ils sont générés aléatoirement pour former la première population et sont modifiés uniquement avec le mécanisme de mutation.

KOZA a proposé en 1992 dans son livre "Genetic Programming" [126] une autre méthode appelée programmation génétique (PG). Cette méthode est inspirée du modèle des algorithmes génétiques (AG) de HOLLAND [116] : la PG manipule pour sa part des arbres codant des programmes. Cette méthode est actuellement la plus utilisée, c'est pourquoi elle nous sert de référence pour nos comparaisons. Son fonctionnement est détaillé dans la section suivante.

TAB. 7.1 – Tableau donnant pour quelques problèmes classiques [126] la fonction de calcul de la qualité d'un programme

Problème	Fonction qualité
Problème de centrage d'une voiture	Somme des durées, sur les cas d'apprentissage, pour centrer la voiture
Problème de la fourmi artificielle	Nombre de boulettes de nourriture trouvées
Problème de régression simple	Somme, pour les cas d'apprentissage, des valeurs absolues des différences entre les valeurs obtenues et celles attendues
Problème de multiplexeurs	Nombre de valeurs obtenues qui correspondent à celles attendues
Problème de classification	Nombre d'éléments bien classés

D'autres méthodes ont été ensuite développées ; parmi celles-ci, en 1997, SALUTOWICZ et SCHMIDHUBER ont proposé PIPE [187] (Probabilistic Incremental Program Evolution) qui est une méthode inspirée par le modèle de la PG et celui de PBIL [6]. Elle utilise un vecteur de probabilité pour la génération de solutions (arbre codant des programmes), il se comporte comme une mémoire statistique qui mémorise la forme de certaines solutions tout au long de l'exécution. Ce vecteur est modifié par la meilleure solution qui va augmenter la probabilité de choix de ses composants, tandis que la plus mauvaise solution va diminuer ces probabilités.

PIPE se distingue de EP et PG par le fait qu'à chaque génération la population est entièrement générée grâce aux probabilités. Les solutions de la génération précédente ne sont pas utilisées. PIPE peut malgré tout être vu comme faisant partie des sytèmes évolutifs. En effet, il travaille avec une population, la recombinaison étant collective au travers de l'arbre de fréquences et la mutation étant directement appliquée à cette structure.

7.3 La programmation génétique

La représentation utilisée pour implanter la PG par KOZA est l'arbre, c'est aussi celle du langage Lisp. Ce modèle de graphe comprend deux types de nœuds :

- les nœuds internes que l'on peut regrouper sous le terme générique de "fonction". Il peut s'agir de fonctions booléennes, arithmétiques, transcendantes, à effet de bord (assignation de variables, déplacement de robot, ...), fonctions implantant des structures de contrôle : alternative, boucle, appel de routines, ... Elles peuvent être caractérisées entre autres par leur arité qui indique le nombre de nœuds qui pourront leur être attachés, nœuds qui fournissent les paramètres de ces fonctions.
- les feuilles qui représentent les terminaux. Il s'agit classiquement de pseudo-variables contenant les entrées du programme, de constantes qui sont fixées par une connaissance préliminaire du problème ou générées aléatoirement (*random ephemeral constant*), de fonctions sans argument mais avec effet de bord, et de

variables ordinaires.

Pour illustrer l'utilisation de la structure d'arbre et l'évaluation de la qualité du programme qui lui correspond, nous prenons l'exemple cité par M. MITCHELL dans "An introduction to genetic algorithms" [157].

Ce problème est défini par le jeu d'apprentissage donné dans le tableau 7.2. Dans cet exemple, le but est d'obtenir une fonction mathématique qui donne la période de révolution d'une planète en fonction de sa distance au soleil. Cela se traduit par la recherche de la formule qui nous donne la valeur placée dans la seconde colonne à partir de la valeur correspondante de la première colonne. La grammaire utilisée ici, se compose des fonctions $\{+, -, *, /, \sqrt{}\}$ et du terminal A.

TAB. 7.2 – Durée d'une révolution P (en années terrestres) en fonction de la distance A de la planète au soleil (la distance est exprimée en fonction de la distance terre/soleil)

Planéte	Valeur de A	Valeur de P
Venus	0.72	0.61
Terre	1.00	1.0
Mars	1.52	1.84
Jupiter	5.20	11.9
Saturne	9.53	29.4
Uranus	19.1	83.5

Lors de l'application de la PG, un ensemble de solutions sera généré de façon aléatoire, 5000 pour notre exemple. La meilleure solution obtenue lors de cette première génération aléatoire, est présentée figure (7.2). La qualité de cette formule est évaluée grâce aux valeurs données du jeu d'apprentissage, cette qualité est basée sur la somme des écarts entre les résultats obtenus et ceux attendus. Plus cette somme est faible, meilleure est la qualité. La valeur de cette solution est de $91,4$, et son expression simplifiée est $f(A) = A$.

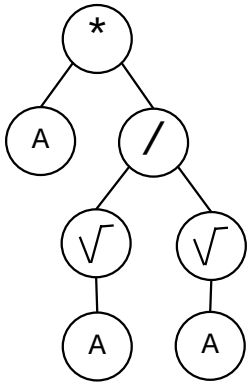


FIG. 7.2 – Arbre généré par PG à l'initialisation correspondant à $A\frac{\sqrt{A}}{\sqrt{A}}$

Pour les arbres représentant les programmes, les opérateurs de mutation et de recombinaison sont classiquement implantés de la façon suivante :

- La recombinaison se réalise par l'échange d'un sous-arbre entre deux parents, ces parties d'arbre se situent au dessous de points de coupure qui sont choisis aléatoirement (fig. 7.3).

– La mutation d'un programme consiste à remplacer un sous-arbre par un nouveau, ce dernier étant généré aléatoirement (fig. 7.4).

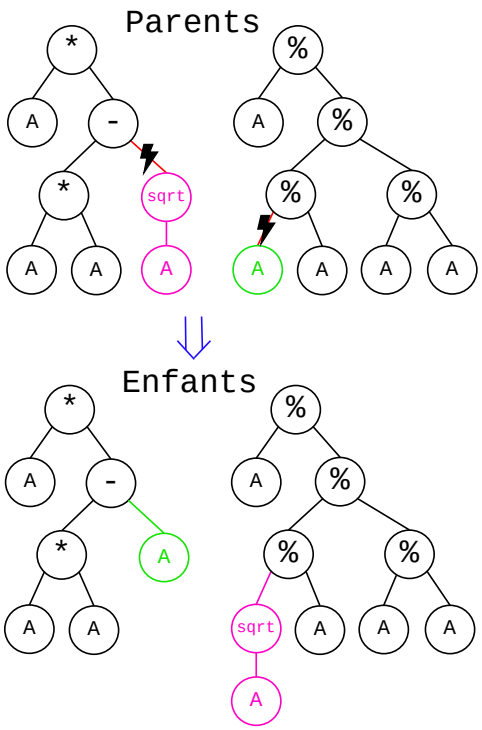


FIG. 7.3 – Recombinaison de deux programmes parents et génération de deux nouveaux programmes fils. Ici la recombinaison se réalise par l'échange d'un sous-arbre entre les parents, le point de coupure est indiqué par un éclair noir

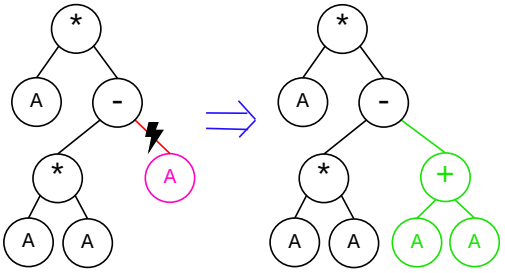


FIG. 7.4 – Mutation d'un programme, il s'agit de remplacer un sous-arbre par un nouvel arbre, généré aléatoirement

Ces opérateurs sont donc appliqués aux solutions de la génération initiale, et de meilleures fonctions apparaîssent dans la génération suivante, par exemple $f(A) = ((A - A) + (A - A)) - ((A/A) - (A + A)) = 2A - 1$.

C'est la pression de la sélection qui va faire émerger de meilleures solutions au fil des générations, comme il est possible de le voir dans la figure (7.5). Dès la troisième génération, on obtient une fonction qui correspond à nos attentes ($f(A) = A\sqrt{\sqrt{A^2}} = A\sqrt{A}$). Elle correspond à la troisième loi de Kepler qui est $P^2 = cA^3$ avec $c = 1$. On peut noter qu'à la génération 98 une fonction simplifiée est aussi trouvée sous la forme de $f(A) = \sqrt{A^3}$.

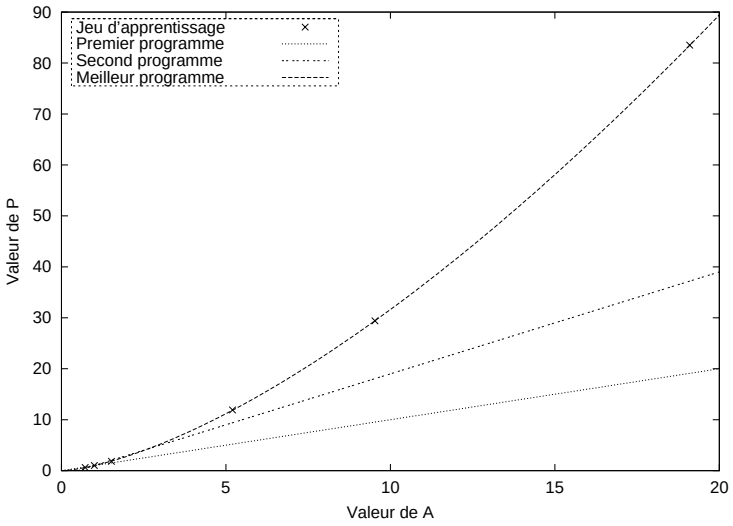


FIG. 7.5 – Représentation de trois solutions obtenues avec PG et des valeurs du jeu d'apprentissage

La PG obtient des bons résultats sur les différents problèmes classiques sur lesquels elle a été appliquée [126], mais aussi sur des problèmes réels. On peut citer, pour ces derniers, la recherche de la concentration en phytoplancton à partir de la couleur de l'eau [80], la classification de séquences d'ADN [111], la création de stratégies boursières [136], la génération automatique de circuits éléctroniques [129], etc.

7.4 Améliorations actuelles de la PG

BANZHAF *et al.* [10] ont montré dans une expérience très instructive que la recombinaison a généralement un effet destructif, comme le montre l'analyse comparée entre la qualité des parents et celle des enfants fruits de la recombinaison (fig. 7.6).

Les travaux actuels visent à modifier la recombinaison pour éviter son effet destructeur, et à lui intégrer une prise en compte de la nature du problème à résoudre afin d'améliorer ses performances. C'est dans ce cadre que l'on trouve tous les développements visant à optimiser le placement des points de coupure utilisés pour la recombinaison.

Pour LANGDON [133] la recombinaison doit limiter le développemnt de la taille des arbres, c'est pour cela que les points de coupure dans les deux arbres doivent se placer sur des positions homologues, c'est-à-dire que la distance entre ces deux points doit être limitée. Cela évite le développement exponentiel des arbres générés. Pour D'haseleer [52] cette limitation du développement passe par l'échange de sous-arbres qui doivent avoir des structures semblables. Dans le livre de Banzhaf *et al.* [10], les auteurs développent ce principe et font le parallèle avec le modèle biologique. Dans la nature la recombinaison a un effet limité, les échanges se faisant entre gènes codant les mêmes caractéristiques : on ne trouvera donc pas un échange entre les pigments définissant la couleur des yeux et ceux de la peau ou des cheveux. En s'appuyant sur ces constatations, les auteurs proposent une recombinaison basée sur des ressemblances structurelles et fonctionnelles.

Une autre amélioration apportée à la programmation génétique est l'intégration

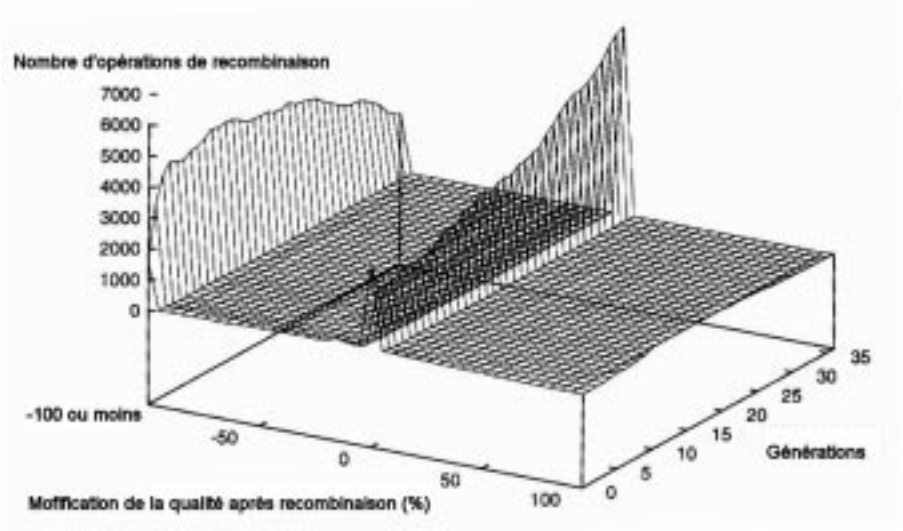


FIG. 7.6 – Effet de la recombinaison. La décroissance (en nombre d'individus) de la qualité des enfants par rapport aux parents est sur la partie gauche, l'amélioration sur la partie droite

d'informations liées à la connaissance de la nature du problème pour la construction de l'opérateur de recombinaison, principe qui est hérité des algorithmes génétiques.

L'intégration d'informations dans la recombinaison ou dans l'algorithme de PG peut se faire en rendant la PG *fortement typée*. Cet ajout d'informations peut être intégré par l'utilisation d'une grammaire à contexte libre (CFG *Context Free Grammar*). L'utilisation d'une telle grammaire ne permet pas le respect de la propriété de clôture. Elle implique la modification des opérateurs de recombinaison qui doivent contrôler le respect des types acceptés en paramètre par les opérateurs. L'incorporation des ADF (*Automatically Defined Functions*) et autres ADM (*Automatically Defined Macros*) [126, 127, 128] permettent également de faire apparaître d'une manière co-évolutionnaire une connaissance propre au problème traité dans la programmation génétique. La figure 7.7 présente les principes de la programmation génétique fortement typée.

De même, on a amélioré le processus de la mutation. Comme pour la recombinaison, la mutation peut intégrer un facteur afin de limiter la taille de l'arbre produit dans le cas de la mutation de sous-arbre.

Les meilleures méthodes de mutations sont, comme pour la recombinaison, celles qui comportent une connaissance du problème. Parmi les mutations qui ont été proposées on peut citer la mutation des constantes numériques proposées par SCHOENAUER [196] utilisant une gaussienne et un opérateur qui effectuent une recherche sur toutes les constantes numériques disponibles.

7.5 L'algorithme de programmation par fourmis (AP)

L'algorithme que nous proposons [181], Ant Programming ou AP consiste à créer à chaque génération une nouvelle population de programmes, en utilisant une mémoire inspirée des systèmes à base de phéromone. Les programmes qui sont créés ont une structure d'arbre identique à celle que nous avons décrite précédemment.

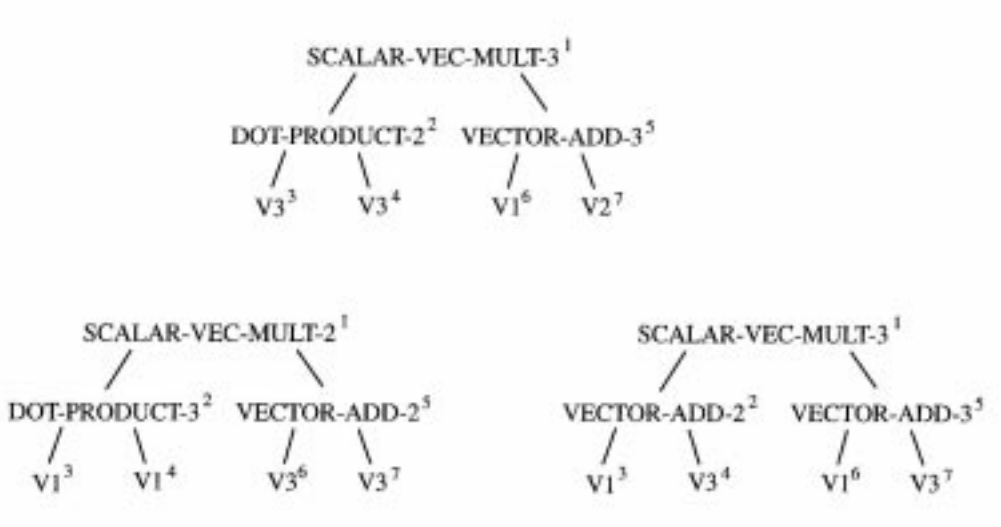


FIG. 7.7 – Représentation de la programmation génétique fortement typée. Chaque variable et constante a un type. Les constantes ont le type SCALAR, les variables peuvent avoir le type VECTOR-N (vecteur à N dimensions) ou MATRIX-N-N (matrice de taille N x N). De même chaque fonction possède un type pour chacun de ses arguments et pour sa valeur de retour. V1 et V2 sont de type VECTOR-3, V3 est de type VECTOR-2. L'arbre du dessus est un arbre valide et retourne un vecteur de type VECTEUR-3. Par contre, les deux arbres suivants sont invalides, le premier parce que la valeur retournée par la racine est invalide et le second parce que V3 n'est pas du bon type.

7.5.1 Le détail de notre algorithme

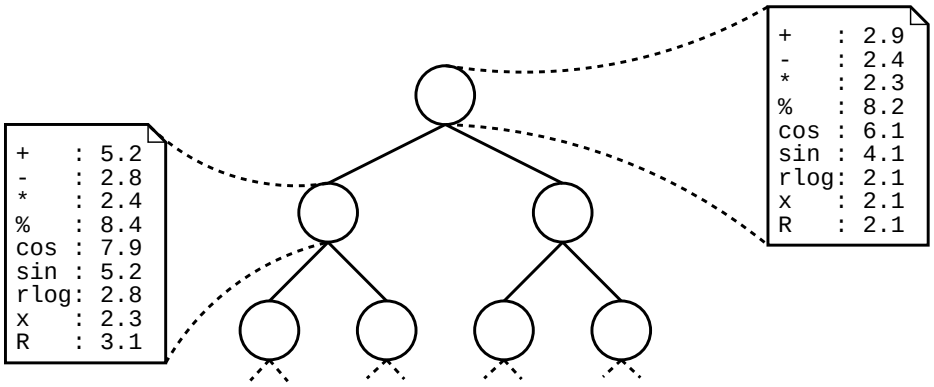


FIG. 7.8 – Arbre de phéromone : Dans chaque nœud est placée la quantité de phéromone associée à chaque fonction ou terminal possible.

Le mode de fonctionnement d'AP reprend celui des algorithmes à base de colonies de fourmis. L'utilisation de la phéromone dans notre méthode peut être rapprochée de l'utilisation de la forme de mémoire statistique qu'utilise PIPE, tout comme il est possible de rapprocher la phéromone des OCF de l'utilisation des vecteurs de probabilité de PBIL.

Les OCF se caractérisent par l'utilisation de la phéromone qui est stockée dans une structure. Pour AP, cette structure est un arbre de phéromone représenté figure (7.8).

La forme est la même que celle des solutions mais, pour chaque nœud on trouve un tableau qui donne la quantité de phéromone associée aux différentes fonctions et terminaux possibles.

Algorithme 7.1: Algorithme AP.
AP()

(1) initialisation de l'arbre de phéromone
(2) profondeur_max := profondeur_initiale
(3) **faire**
(4) **pour** chaque fourmi **faire**
(5) générer un arbre en se basant sur la phéromone
(6) évaluer le programme sur le jeu d'apprentissage
(7) Simuler l'évaporation de la phéromone
(8) **pour chaque** programme
(9) simuler le renforcement de la phéromone selon la qualité du programme
(10) **si** aucune amélioration du meilleur de l'exécution depuis MAX itérations **alors**
(11) profondeur_max := profondeur_max + 1
(12) **jusqu'**au critère de terminaison
(13) retourne le meilleur programme de l'exécution comme solution

Le schéma de fonctionnement d'AP est donné dans l'algorithme 7.1. Le nombre de fourmis est donc le nombre de programmes qui seront manipulés à chaque itération.

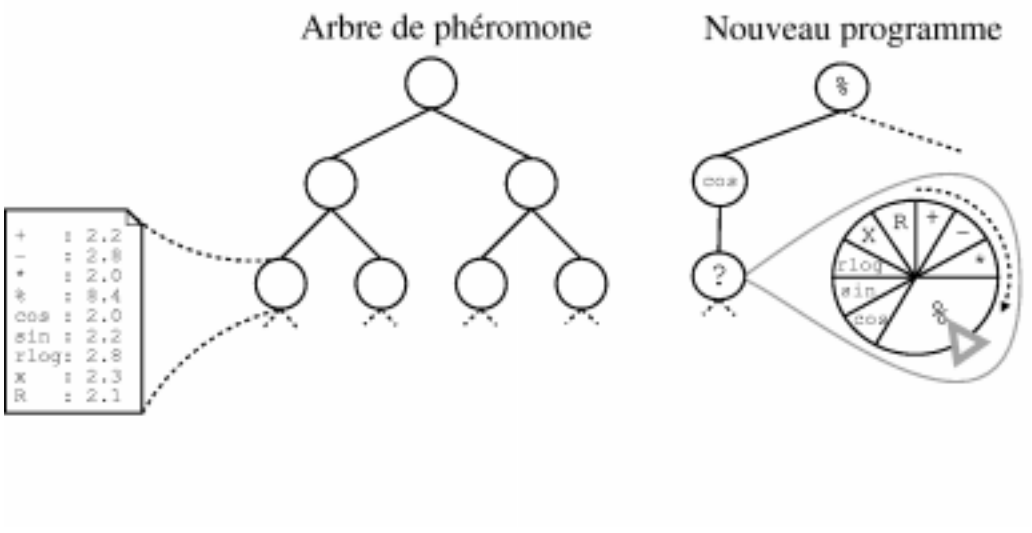


FIG. 7.9 – Sélection d'un nœud lors de la génération d'un programme par AP. Cet élément est choisi grâce à une méthode de roulette, c'est-à-dire que la sélection se fait avec une probabilité proportionnelle au taux de phéromone associé aux différents composants possibles correspondant à la position de ce nœud.

Chacune de ces fourmis construit une solution en utilisant la phéromone. Le mécanisme

est le suivant : elle crée un nœud, ensuite elle choisit le terminal ou la fonction qu'elle y placera. Ce choix se fait grâce à une méthode de roulette (fig. 7.9), c'est à dire avec une probabilité proportionnelle au taux de phéromone associé aux différents éléments, ces valeurs étant stockées au nœud correspondant dans l'arbre de phéromone. Formellement, cette probabilité de choix d'un symbole est donnée par la formule :

$$P_{s,n} = \frac{\tau_{s,n}}{\sum_{\sigma \in S_n} \tau_{\sigma,n}}$$

- n désigne le nœud courant,
- s désigne le symbole,
- S_n est l'ensemble des symboles possibles pour le nœud n,
- $\tau_{s,n}$ est le taux de phéromone associé au symbole s pour le nœud n.
- $\sum_{\sigma \in S_n} \tau_{\sigma,n}$ donne la somme des taux de phéromone pour l'ensemble des symboles S_n du noeuds n

Pour la première itération, tous les taux de phéromone sont égaux : le mécanisme de création peut donc être assimilé à une génération aléatoire, aucun symbole n'étant favorisé sur aucun des nœuds. Si l'élément associé au nœud est une fonction, la création se poursuit en profondeur. On génère un fils par argument nécessaire à la fonction, et on réitère récursivement cette phase de création/sélection pour générer tout l'arbre. Par contre, s'il s'agit d'un terminal, la branche sur laquelle on se trouve est achevée. Lors de la génération, les arbres ne peuvent excéder une profondeur maximale fixée ; lorsqu'elle est atteinte, les seuls noeuds qui peuvent être sélectionnés sont les terminaux.

Quand les solutions ont été générées, on passe à la phase d'évaluation, grâce au jeu d'apprentissage, comme pour la PG.

En utilisant les valeurs de qualité, les fourmis vont pouvoir mettre à jour l'arbre de phéromone. Ce mécanisme se décompose en deux phases :

1. D'abord, l'ensemble des taux de phéromone va subir une évaporation, ce qui se traduit par une diminution des valeurs en accord avec la formule :

   $$\tau'_{s,n} = (1 - \alpha)\tau_{s,n}$$

   où α désigne une constante (pour notre algorithme $\alpha = 0.1$), s désigne un des symboles pour le nœud n, $\tau_{s,n}$ est le taux de phéromone du symbole s pour le nœud n, et τ' est la valeur de phéromone après évaporation.

2. Ensuite, toutes les fourmis vont renforcer le taux de phéromone qui est associé à chaque élément de l'arbre/programme qu'elle a construit : ce renforcement se fait de façon proportionnelle à leur valeur de qualité de ce programme. La formule utilisée est la suivante :

   $$\tau''_{s,n} = \tau'_{s,n} + \frac{\exp^{f(p)*\log(100)}}{100}$$

   où s est le symbole placé dans le nœud n du programme p, $f(p)$ est la qualité de p, prenant sa valeur dans l'intervalle $[0, 1]$ (0 correspond à un très mauvais programme et 1 à un programme parfait sur le jeu d'apprentissage), τ' est la valeur courante de phéromone pour le symbole s du noeud n, et τ'' est la valeur après le renforcement par le programme p.

TAB. 7.3 – Valeurs d'apprentissage de l'exemple

Cube	Coté cm	Volume cm^3
1	1	1
2	3	27
3	5	125
4	7	343
5	11	1331

Pour assurer une bonne diversité des programmes générés, le principe du $Min-Max$ introduit par STÜTZLE et HOOS [209] et présenté dans la section 3.5.1 est utilisé pour borner les valeurs de phéromone. Dans notre cas, nous avons repris la formule qu'ils ont proposée : $Min = 2 * max\_val$, $Max = 5 * Min$.

max_val désigne la qualité maximale d'un programme : il est égal à 1 pour notre algorithme, ce qui donne comme borne aux taux de phéromone les valeurs $Min = 2$ et $Max = 10$. Le choix de la valeur d'initialisation pour la phéromone s'est porté sur la valeur médiane (6).

Afin d'étendre progressivement l'espace de recherche, un mécanisme permet d'augmenter la profondeur des arbres/solutions au cours de l'exécution. A chaque fois que le meilleur programme trouvé n'a pas évolué depuis MAX itérations, la profondeur maximale autorisée est incrémentée de 1, les nœuds nouvellement créés sont initialisés à la valeur médiane (6).

Les paramètres qui devront être spécifiés pour l'exécution de notre algorithme sont les suivants :

- la profondeur maximum des arbres
- le nombre d'itérations
- le nombre de fourmis
- la grammaire

7.5.2 Exemple de déroulement pour AP

Dans cet exemple, l'objectif est de retrouver la formule donnant le volume d'un cube en fonction de son côté.

Le tableau 7.3 définit le jeu d'apprentissage pour AP. Il donne le volume pour 5 cubes de dimensions différentes :

Pour la génération des équations, AP pourra utiliser les symboles suivants :

- comme fonction : $+$, $-$, $*$, % (% division protégée) ;
- comme terminal : c qui est la longueur d'un côté du cube.

Dans un souci de simplification, seules deux fourmis seront utilisées, et les arbres manipulés auront comme profondeur maximale 3.

D'abord, on initialise la matrice de phéromone avec la valeur initiale $\tau_0 = 6$ comme le montre la figure 7.10

Ensuite, les deux fourmis génèrent leur solution en se basant sur la phéromone. Pour

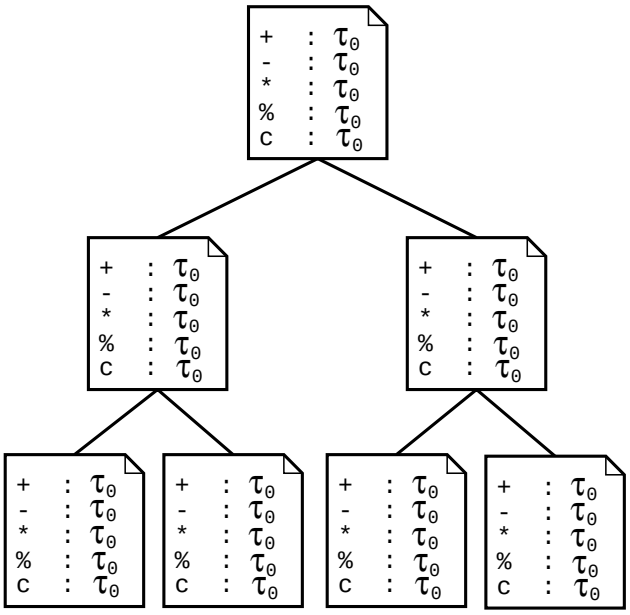


FIG. 7.10 – L'arbre de phéromone après initialisation

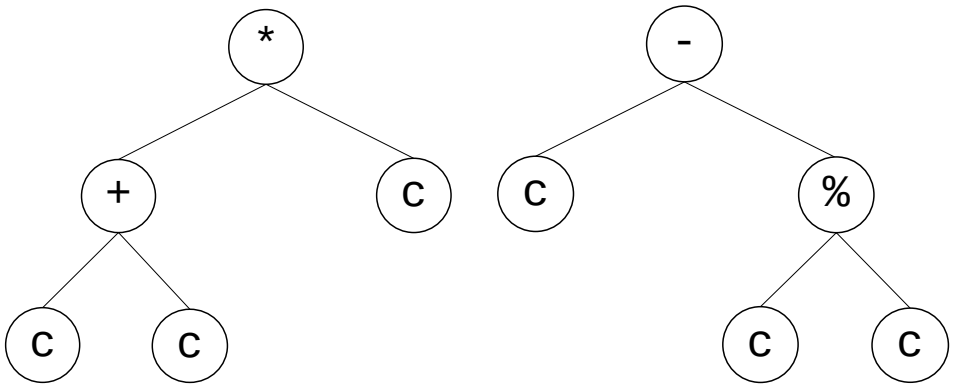
cette première itération toutes les valeurs sont identiques, cela équivaut à générer des solutions aléatoirement. Deux solutions possibles sont présentées dans la figure 7.11.

On applique la phase d'évaporation à toutes les valeurs de phéromone :

$$\tau_0' = (1 - \alpha) \cdot \alpha$$

On calcule la qualité des solutions obtenues en trois étapes :

1. on calcule la somme des valeurs absolues des différences entres les volumes attendus et ceux trouvés en utilisant l'équation (E_k) générée par la fourmi (k) ;
2. on ajoute 1 ;
3. on prend l'inverse.



arbre de la fourmi a arbre de la fourmi b

FIG. 7.11 – L'arbre de phéromone après initialisation

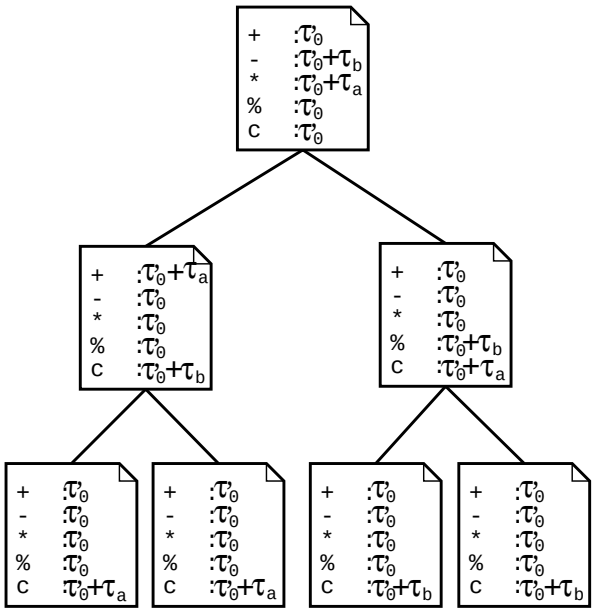


FIG. 7.12 – L'arbre de phéromone après la première itération

Le but ici est d'obtenir une valeur de qualité comprise entre 0 et 1, 0 étant la plus mauvaise qualité de solution et 1 la meilleure. Les valeurs de qualité pour les deux solutions sont donc :

$$f(a) = \frac{1}{\sum_{n=1}^{5} \mid (V(cube_n) - E_a(cube_n) \mid +1} = \frac{1}{1419+1} = 0,0007$$

$$f(b) = \frac{1}{\sum_{n=1}^{5} \mid (V(cube_n) - E_b(cube_n) \mid +1} = \frac{1}{1804+1} = 0,0006$$

Ensuite, on met à jour la matrice de phéromone avec toutes les solutions trouvées. L'apport de chaque fourmi est respectivement égal à :

$$\tau_a = \frac{\exp^{f(a)*\log(100)}}{100} = 0,0739$$

$$\tau_b = \frac{\exp^{f(b)*\log(100)}}{100} = 0,0744$$

On regarde pour chaque symbole de chaque nœud de l'arbre si la valeur de phéromone respecte le critère de max-min. Cela revient à tester :

– si la valeur τ_1 obtenue après la mise à jour est inférieure à τ_{min}, alors $\tau_1 = \tau_{min}$;

– si la valeur τ_1 obtenue après la mise à jour est supérieure à τ_{max}, alors $\tau_1 = \tau_{max}$;

Après la première itération, la matrice de fréquence a la forme présentée dans la figure 7.12.

A chaque étape de l'exécution, l'arbre de phéromone est modifié suivant les solutions trouvées par les fourmis. On peut y voir émerger la forme d'une solution optimale, composée des éléments qui ont le plus fort taux de phéromone. Dans notre exemple, au cours de l'exécution, il est possible d'obtenir un arbre de phéromone semblable à celui présenté figure 7.13. Dans cet arbre, on voit nettement apparaître la solution $c * c * c$

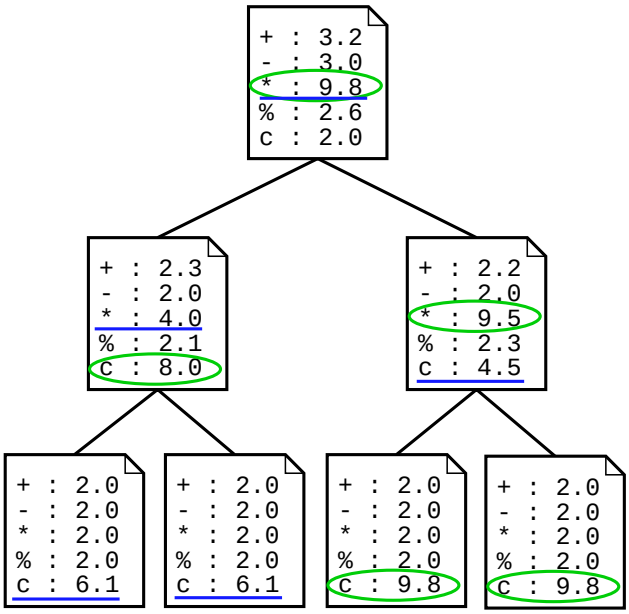


FIG. 7.13 – L'arbre de phéromone après n itérations : la formule du volume y apparaît (cercle), on peut également trouver une autre représentation de cette formule (souligné)

(cercle) qui est l'équation que l'on recherche. L'équation du volume recherché a deux représentations équivalentes sous forme d'arbres. La seconde forme apparaît également dans l'arbre de phéromone (souligné) avec des valeurs inférieures à l'autre solution, cette différence peut s'expliquer par le fait qu'au cours de l'exploration les éléments de la première solution ont été choisis plus fréquemment.

7.6 Conclusion

La programmation automatique offre un bon outil pour la résolution de problèmes complexes. Il nous a paru intéressant de tester l'adaptation du modèle de coopération des fourmis sur ce type d'algorithme dont l'exemple le plus répandu est la PG.

AP se distingue de la PG par une approche totalement différente :

- contrairement à la PG, il n'existe pas de mécanisme permettant l'échange direct d'informations de structure entre individus de la population (pas de mécanisme de recombinaison) ;
- la mémoire dans la PG est contenue dans le patrimoine génétique de chaque individu. Par contre pour AP, elle se présente sous la forme d'une mémoire globale (arbre de phéromone), les solutions sont détruites à chaque génération et de nouvelles sont créées en se basant sur cette mémoire.

Le principe utilisé pour AP est celui des algorithmes à colonies de fourmis, sa forme peut faire penser à celle de l'algorithme PIPE, tant par son comportement que par sa structure. Cependant, AP s'en distingue nettement par les points suivants :

- son mécanisme de mise à jour est beaucoup plus simple que le calcul de probabilité de PIPE : une modification pour un nœud donné ne se propage pas dans le sous-arbre dont il est racine ;
- il n'existe pas de mécanisme de mutation destiné à la recherche locale, contraire-

ment à PIPE où l'on trouve une phase de mutation de l'arbre de probabilité. Une différence secondaire apparaît dans l'utilisation des solutions pour la mise à jour : pour AP chaque solution générée apporte sa contribution contrairement à PIPE où seul le meilleur résultat est pris en compte.

Les remarques précédentes attestent de la simplicité des mécanismes mis en jeu pour la gestion de l'arbre de phéromone, comparée à la gestion de l'arbre de probabilité de PIPE. Reste la question de savoir si AP ne serait pas une forme simple d'un modèle très complexe d'algorithme guidé par les probabilités. Dans cette éventualité tous les algorithmes à colonies de fourmis ne seraient-ils pas dans le même cas ?

Dans la section suivante nous allons examiner les résultats obtenus avec AP et les comparer avec ceux de la PG.

Chapitre 8

Résultats d'Ant Programming

Ce chapitre a pour but de valider l'approche mise en œuvre dans AP. Pour cela, nous comparerons les résultats de AP avec ceux de la GP, sur des problèmes classiques proposés par Koza.

8.1 Les problèmes

Pour évaluer les performances de AP nous avons comparé les résultats obtenus sur des problèmes classiques avec ceux produits par l'algorithme de programmation génétique décrit par Koza.

Les comparaisons ont été réalisées sur des problèmes classiques, qui sont souvent utilisés comme base de comparaison pour les algorithmes de programmation automatique.

Les problèmes présentés ici sont les suivants :

- le problème du multiplexeur à 11 bits ;
- les problèmes de parité à 3, 4, 5 ou 6 bits ;
- deux problèmes de régression ;
- le problème des deux spirales ;
- le problème de la fourmi artificielle dans le cas du parcours " Santa Fe trail ".

Pour chacun de ces problèmes, nous allons exposer son principe ainsi que les différents paramètres utilisés. Le tableau 8.1 donne les fonctions et les terminaux qui sont utilisés pour la génération des programmes.

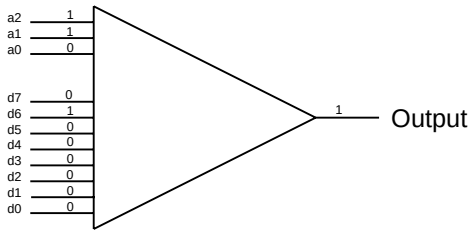


Fig. 8.1 – Multiplexeur 11 bits avec comme entrée 11001000000 et comme sortie 1

Le problème du multiplexeur à 11 bits consiste à trouver le programme qui

TAB. 8.1 – Ensemble des fonctions et des terminaux

Problèmes		
Multiplexeur 11 bits	fonctions	and, or, not, if
	terminaux	$a_0, a_1, a_2, d_0, \ldots, d_7$
n-parité	fonctions	$and, or, nor, nand$
	terminaux	$d_0, d_1, \ldots$
Fonctions de regression	fonctions	$+, -, \%, \sin, \cos, \exp, \log$
	terminaux	x, R
Spirales entrelacées	fonctions	$+, -, *, \%, \sin, \cos, ifmtz$
	terminaux	R, x, y
Fourmis artificielles	fonctions	if-food-ahead, progn2, progn3
	terminaux	left, right, move

Remarque : *dans la colonne des symboles : R désigne une constante générée aléatoirement entre -1 et 1 ; $ifmtz$ désigne une fonction d'arité 3 qui retourne la valeur de la seconde expression si le résultat de la première expession est positif, sinon le résultat de la troisième expression est retourné ; x, y désignent les coordonnées du point dans le repère cartésien.*

sélectionne parmi les 8 bits de données (d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7) la valeur du $j^{\text{ème}}$ bit, celui-ci étant désigné par une adresse codée sur 3 bits (a_0, a_1, a_2). Pour ce problème, la qualité d'un programme est le nombre de bonnes sorties trouvées sur le jeu de test qui en comprend 2^{11}. La figure 8.1 présente un exemple où la valeur $(01000000)_2$ est placée en entrée pour les données et $(110)_2$ pour l'adresse, ces valeurs correspondant respectivement à 64 et 6 en décimale. Le programme doit donc sélectionner en sortie le $6^{\text{ème}}$ bit qui est égal à 1.

Les problèmes de parité à n bits consistent à trouver un programme qui retourne Vrai (1) si le nombre de 1 en entrée est pair et Faux (0) dans le cas contraire. La qualité des programmes est évaluée sur les 2^n entrées possibles.

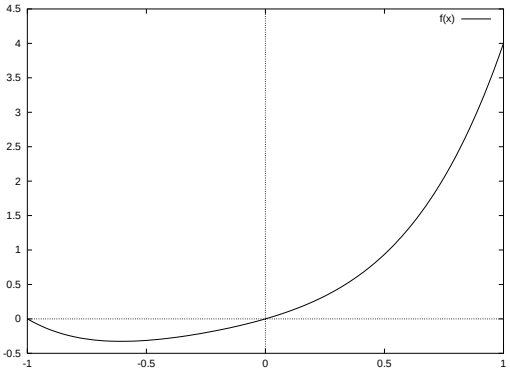


FIG. 8.2 – Problème de régression : $f(x) = x^4 + x^3 + x^2 + x$

Les problèmes de régression consistent à retrouver une fonction mathématique à partir d'un ensemble de valeurs discrètes données. La valeur de la qualité est la

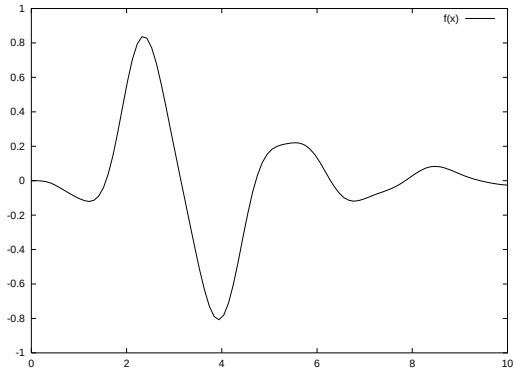


FIG. 8.3 – Problème de régression : $f(x) = x^3 \cdot e^{-x} \cdot cos(x) \cdot sin(x) \cdot (sin^2(x) \cdot cos(x) - 1)$

moyenne des écarts entre les 200 valeurs attendues (générées de manière uniforme) et celles produites par la fonction cherchée (i.e. $f(p) = 1/n * \sum_{i=1}^{n} \mid C_{\text{calculé}} - C_{\text{attendu}} \mid$). Les équations recherchées qui ont généré les valeurs de travail sont $f(x) = x^4 + x^3 + x^2 + x$ dans l'intervalle [-1,1] (Fig. 8.2) et $f(x) = x^3 \cos(x) \sin(x)(\sin^2(x) \cos(x) - 1)$ dans l'intervalle [0,10](Fig. 8.3).

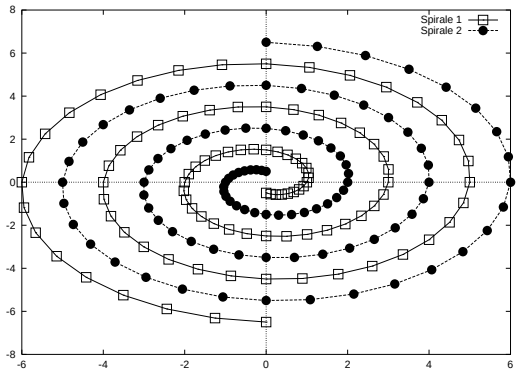


FIG. 8.4 – Representation du problème des deux spirales entrelacées

Le problème des deux spirales est un problème de classification. Il s'agit en partant d'un ensemble de points de pouvoir dire s'ils font partie d'une première spirale ou d'une seconde, ces spirales étant entrelacées comme le montre la figure 8.4. L'ensemble à classer comprend 194 points : ceux désignés par un losange blanc font partie de la première spirale, tandis que ceux désignés par les points noirs font partie de la seconde.

Le problème de la simulation d'une fourmi artificielle. Une fourmi artificielle doit trouver un chemin partiellement couvert de boulettes de nourriture, en effectuant un nombre de pas limités. Ce problème a été résolu par KOZA en utilisant la PG. La fonction qualité compte le nombre de boulettes trouvées lors du parcours. La figure 8.5 présente ce problème (appelé " Santa Fe trail "). Les 89 boulettes sont présentées sous la forme de points noirs, les points blancs complètent les cases manquantes pour former le parcours idéal.

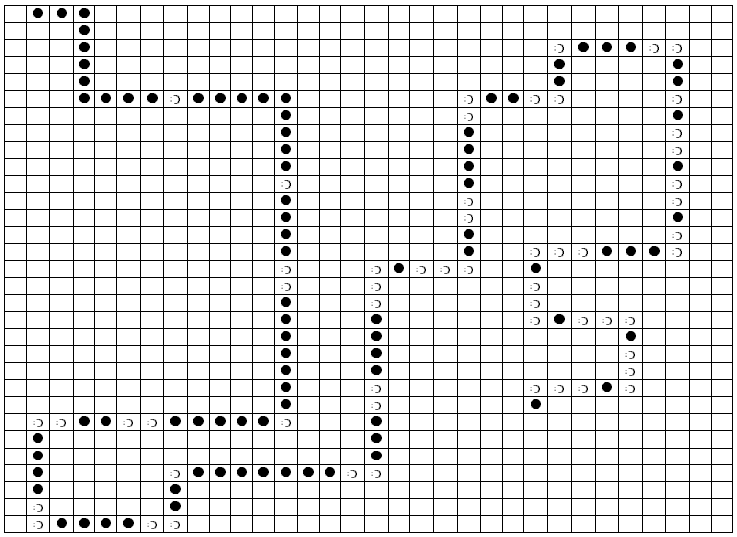


FIG. 8.5 – Representation du "Santa Fe Trail Problem"

8.2 Paramétrage

Les résultats présentés sont obtenus pour 40 exécutions indépendantes. Nous avons utilisé un paramétrage standard pour la PG (probabilité de recombinaison de 0,85 ; probabilité de mutation de 0,05 ; probabilité de recopie de 0,10). Nous avons octroyé un nombre d'évaluations de la fonction qualité semblable pour AP et la PG (c'est cette étape qui représente le coût de calcul le plus élevé). Les résultats présentés pour la PG ont été obtenus en utilisant la librairie lilGP [240] de l'université du Michigan, (sans utiliser la définition automatique de fonctions (ADFs) [127]).

8.3 Comparaison avec la PG

Pour chaque problème, les résultats présentés (tab. 8.3) sont d'une part, la qualité de la meilleure solution obtenue sur les 40 exécutions de AP et la PG, et d'autre part la moyenne de toutes ces exécutions. Le critère d'arrêt pour chaque problème est le nombre d'évaluations de la fonction qualité identique pour chacun des deux algorithmes. Le paramétrage de l'algorithme de la PG est présenté dans le tableau 8.2.

Il apparaît clairement, au vu des résultats que le problème du multiplexeur 11 bits est très difficile pour les deux algorithmes. La meilleure valeur de qualité trouvée par ces deux méthodes n'est que de 1408, pour une valeur maximale possible de 2048.

Pour les problèmes de n-parité, une population de 16000 individus ainsi qu'un nombre de 50 générations (comme le préconise Koza) est utilisé pour la PG. La profondeur des arbres générés est fixée à 17 pour les deux méthodes. AP et la PG obtiennent des performances moyennes pour ces problèmes. Les résultats de la PG dépassent légèrement ceux de AP[1].

Pour le problème des deux spirales entrelacées, la meilleure solution trouvée par AP permet de classer correctement 145 points sur les 194 possibles. La PG pour sa part ne

[1]Nous n'avons pas retrouvé pour la PG des performances aussi bonnes que dans [126]

TAB. 8.2 – Paramétrage de l'algorithme de la PG

Problème	Population	Crossover (%)	Recopie (%)	Mutation (%)	Profondeur maximale
Multiplexeur 11 bits	200	85	10	5	17
Parité 3 bits	16000	85	10	5	17
Parité 4 bits	16000	85	10	5	17
Parité 5bits	16000	85	10	5	17
Parité 6 bits	16000	85	10	5	17
Première fonction de régression	5000	85	10	5	17
Seconde fonction de régression	5000	85	10	5	17
Spirales entrelacées	10000	85	10	5	10
Fourmis artificielles	1000	85	10	5	10

TAB. 8.3 – Résultats : comparaison de AP avec la PG

Problème	Meilleure Solution		Moyenne		Meilleure	Nombre
	AP	PG	AP	PG	possible	d'évaluations
multiplexeur 11 bits	1408	1408	1309,4	1312,7	2048	10000
parité 3 bits	7	8	6,6	6,0	8	800000
parité 4 bits	11	12	10,2	11,4	16	800000
parité 5 bits	18	20	18,0	19,0	32	800000
parité 6 bits	34	36	33,3	35,5	64	800000
Première fonction de régression	0,00	0,00	8,49	3,49	0,00	250000
Seconde fonction de régression	11,98	20,66	32,13	28,08	0,00	250000
Spirales entrelacées	145	129	140,24	118,5	194	500000
Fourmis artificielles	79	89	63,87	59,3	89	10000

permet de classer que 129 points. De plus la moyenne des solutions obtenues par AP est légèrement supérieure à celle de la PG.

Dans le cas des fourmis artificielles pour le problème "Santa Fe Trail", les résultats de la PG (1000 individus) sont meilleurs que ceux de AP, elle trouve même la solution optimale par deux fois. Étrangement, nous pouvons constater l'inverse pour les moyennes où c'est AP qui dépasse la PG (63,8 contre 59,3).

Dans le cas des fonctions de régression, AP réussit à trouver la solution optimale pour le premier problème et obtient de très bons résultats sur le second. Ces deux problèmes sont connus comme étant difficiles à résoudre. La figure 8.6 présente la fonction attendue$(x^3 \cdot e^{-x} \cdot cos(x) \cdot sin(x) \cdot (sin^2(x) \cdot cos(x) - 1))$ pour le second problème ainsi que la meilleure solution trouvée par AP. Il est clair que ces deux courbes sont très proches.

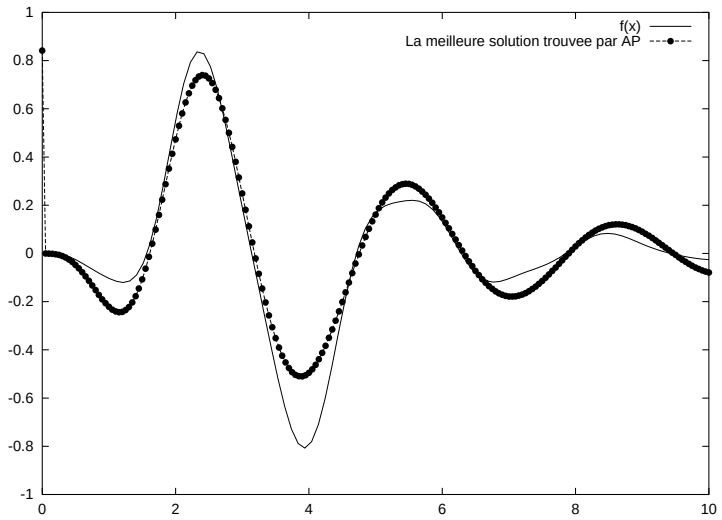


FIG. 8.6 – $f(x) = x^3 \cdot e^{-x} \cdot cos(x) \cdot sin(x) \cdot (sin^2(x) \cdot cos(x) - 1)$ ainsi que la meilleure solution trouvée par AP

Il est intéressant de noter qu'AP a tendance à générer de plus petites solutions que la PG, comme le montre le tableau 8.4. On voit que la moyenne des profondeurs d'arbres trouvés par AP est en général inférieure à celles de la PG. Lorsqu'AP trouve une solution meilleure que celle de la PG, sa profondeur est égale (c'est le cas des 2 spirales) ou inférieure (pour la 2^{nde} fonction de régression) à celle de la PG.

8.4 Extensions

Deux extensions sont actuellement en cours d'implantation, la première vise à intégrer un mécanisme d'intensification, la seconde vise à améliorer la gestion des constantes dans AP.

8.4.1 La génération partielle des individus

Il est toujours tentant d'intégrer dans une heuristique une méthode de recherche locale afin d'améliorer ses performances. Une possibilité est de remplacer une partie

TAB. 8.4 – Comparaison de la profondeur des solutions générées par AP et la PG

Problème	Taille de la meilleure solution		Taille moyenne	
	AP	PG	AP	PG
multiplexeur à 11 bits	4	5	5,6	5,85
parité 3 bits	7	9	5,7	7,4
parité 4 bits	5	6	5,12	5,2
parité 5 bits	4	5	5,4	6,27
parité 6 bits	4	6	3,6	6,95
première fonction de régression	6	7	4,84	15,4
seconde fonction de régression	7	7	5,71	10,8
spirales entrelacées	5	5	5,04	4,98
fourmis artificielles	6	10	5,03	8,88

de l'arbre (sous-arbre) qui code le programme par un nouveau sous-arbre. Dans AP, la création du sous-arbre s'apparente à la création d'un nouvel arbre, la différence résidant dans le choix de l'emplacement du point de départ de cette nouvelle branche pour l'arbre de phéromone : cela ressemble à la mutation de sous-arbres dans la PG.

La sélection de ce sous-arbre peut être le fruit d'un tournoi entre plusieurs candidats choisis aléatoirement. Le tournoi désigne le plus mauvais sous-arbre (un critère de choix pouvant être la qualité de celui-ci). Pour la génération du nouveau sous-arbre, on se placera au nœud équivalent de l'arbre de phéromone et la sélection des nouveaux nœuds utilisera la formule précédemment citée. On trouve une illustration de cette technique dans la figure 8.7.

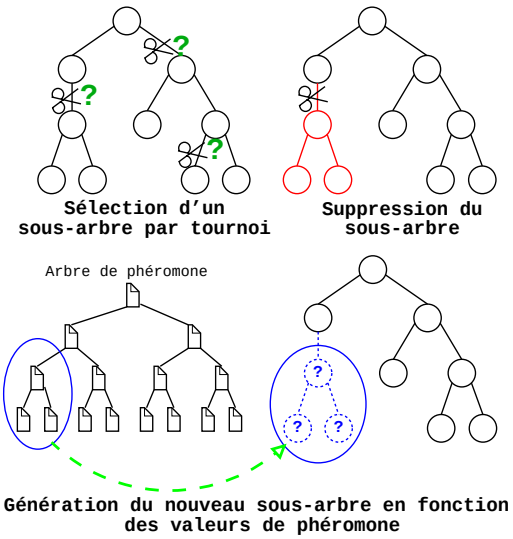


FIG. 8.7 – Génération d'un sous-arbre par utilisation de la phéromone

8.4.2 La gestion des constantes

Pour améliorer les performances d'AP, il semble intéressant d'intégrer un mécanisme plus évolué de gestion des constantes. En effet, si une valeur trouvée fait partie d'une bonne solution. Il est clair qu'un mécanisme conservant les constantes trouvées pour les générations futures augmenterait les performances de AP. Une alternative se présente entre deux solutions envisageables :

- la première possibilité est de placer une mémoire au niveau de chaque nœud de l'arbre de phéromone, constituée de valeurs candidates à la sélection ; cette liste comprendra les m meilleures constantes (la qualité d'une constante (taux de phéromone) étant dictée par la solution qui l'a engendrée) trouvées à cette position. Lors de la génération de l'arbre, AP aura alors le choix entre une valeur aléatoire et une valeur déjà générée suivant une probabilité prédéfinie.
- la seconde possibilité consiste à créer une mémoire indépendante où seront placées les p meilleures constantes (la qualité d'une constante sera évaluée comme dans l'hypothèse précédente). Lorsqu'AP choisira de placer une constante dans un nœud, elle aura le choix entre générer aléatoirement une constante, ou en choisir une de la liste, comme le montre la figure 8.8.

Dans les deux cas, le choix dans la liste pourra se faire suivant une roulette proportionnelle à la qualité des constantes.

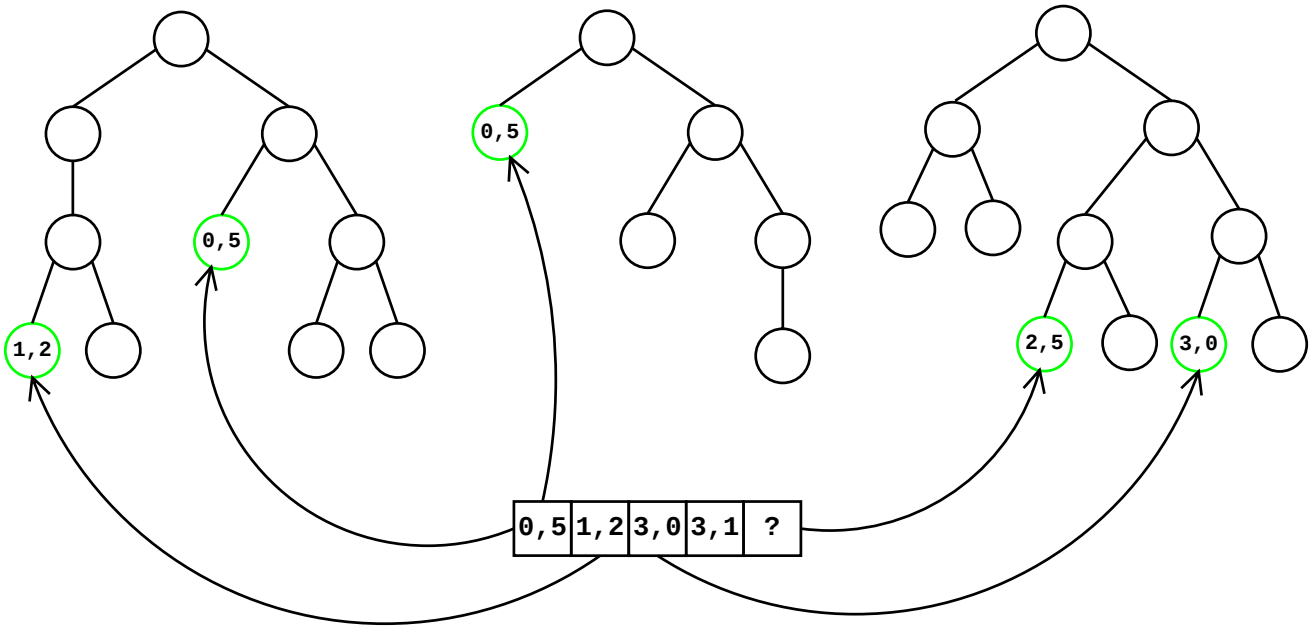


FIG. 8.8 – Choix des constantes. On peut choisir de générer une nouvelle constante (la case ?) ou sélectionner avec une certaine probabilité (proportionnelle à sa qualité) une des quatre déjà utilisée

La seconde méthode devrait faire apparaître des constantes qui sont globalement bonnes, tandis que la première méthode apporte une information supplémentaire dans le choix de la localisation de la constante.

8.5 Conclusion

AP s'inspire de mécanismes naturels, utilisés par les fourmis pour la recherche de nourriture. On peut se demander quels aspects du phénomène réel sont vraiment utiles pour le modèle informatique, et notamment quel est l'apport de l'évaporation de la phéromone à l'algorithme. D'autres techniques qui n'ont pas cette inspiration biologique peuvent aussi être utilisées. Dans notre cas, nous avons par exemple utilisé le concepte du $Min - Max$, qui définit une borne supérieure et inférieure et permet d'avoir un bon dosage entre l'exploration et l'exploitation.

Comme le montrent les résultats présentés dans ce chapitre, AP est une méthode de programmation automatique intéressante, dont les performances peuvent rivaliser avec celles du modèle de la programmation génétique.

Notons que notre algorithme dans sa forme actuelle intègre des limitations, c'est notamment le cas dans la gestion des constantes. En effet, quand une constante est placée dans un programme, sa valeur est initialisée de façon totalement aléatoire. Il serait donc possible comme nous l'avons vu d'intégrer un mécanisme de mémorisation de ces valeurs, afin de permettre l'utilisation ultérieure des constantes faisant partie de bonnes solutions.

Il semble que les solutions générées par AP soient plus compactes en comparaison de celles de la PG. Un complémént d'étude reste nécessaire pour savoir si cette taille est due au principe d'une recherche avec mémoire globale (ici la phéromone) ou au procédé utilisé pour faire grandir les arbres. Ces petites solutions ont des chances d'être plus généralisables, suivant le principe du "rasoir d'Occam".

Quatrième partie

Les fourmis et après ?

Chapitre 9

Conclusion

Dans ce mémoire nous avons vu comment, grâce à un mécanisme de coopération indirecte appelé stigmergie, les fourmis étaient capables de résoudre des problèmes complexes. De la description de ces insectes eusociaux ont émergé des modèles comportementaux en biologie, et des modélisation d'agents-fourmis pour la réalisation de robots ou d'heuristiques. Ce mode de communication qui utilise un marqueur chimique, la phéromone, permet par exemple aux fourmis naturelles de trouver le chemin le plus court dans l'expérience des deux ponts. La similarité de ce problème avec l'opération de base du problème du voyageur de commerce a donc conduit à développer une famille d'algorithmes que l'on regroupe sous le nom d'OCF, pour méthode d'Optimisation à Colonie de Fourmis.

Ces OCF obtiennent de bonnes performances pour le PVC, ce qui a incité les chercheurs à trouver de nouveaux champs d' application. Une méthode simple pour savoir si les OCF sont directement applicables à un problème donné est de regarder s'il est possible de l'exprimer sous la forme d'un graphe. Les agents-fourmis simulent alors leur comportement naturel, se déplaçant d'étape en étape, guidées par la phéromone. Ce déplacement encode au final une solution pour le problème considéré : circuit pour le PVC, affectation pour le PAQ, ...

Il est toutefois intéressant de noter que la majorité des OCF s'écartent du modèle naturel en intégrant, d'une part une composante heuristique spécialisant ainsi la méthode au problème à résoudre, et d'autre part une méthode de recherche locale nécessaire à l'obtention de bonnes performances. Une exception notable est fournie par les heuristiques de routage qui considèrent souvent les fourmis comme de vrais agents se déplaçant dans le réseau de communication.

Il est incontestable que les OCF permettent de développer des méta-heuristiques performantes pour de nombreux problèmes. Il est intéressant de tenter de situer ces méthodes parmi celles déjà existantes. Pour cela, nous avons pris le parti de nous placer du point de vue de l' utilisation de la mémoire et de proposer une taxinomie. Cette comparaison reflète bien les différences de performance entre les différentes heuristiques, les meilleures étant celles qui ont la gestion la plus évoluée de leur mémoire ou qui utilisent plusieurs formes de mémoire à la fois.

Pour valider l'intérêt de la coopération face à la force brute, nous avons développé une heuristique nommée ANTabu, hybridant les OCF avec une recherche locale puissante (la recherche tabou). Après avoir validé les différents choix d'implantation de notre méthode, nous avons comparé notre heuristique avec PATS (un tabou parallèle).

Malgré une différence de puissance de calcul très largement en défaveur de notre algorithme, les résultats montrent nettement le gain obtenu par la coopération face à l' utilisation de la force brute. Notre heuristique a été également confrontée à d'autres méthodes utilisées pour le PAQ, démontrant ainsi ses très bonnes performances.

Nous avons également étudié l'adaptation à la programmation automatique du modèle de la coopération entre fourmis. La conception de notre algorithme Ant Programming (ou AP) s'éloigne au minimum du modèle naturel des fourmis, et la comparaison avec la Programmation Génétique montre l'intérêt de cette proposition. Beaucoup de possibilités restent ouvertes vis à vis d'AP, pour lui donner une robustesse suffisante pour se comparer avec les méthodes les plus récentes de programmation automatique, comme PIPE. Au chapitre des différences, on peut noter que le principe de gestion de la mémoire est bien plus simple à mettre en oeuvre dans AP que dans PIPE. Les résultats obtenus nous incitent à poursuivre dans cette voie, où de nombreuses extensions sont possibles :

- la génération d'arbre en partant des feuilles ;
- la recherche de sous-arbres dans l'ensemble des solutions formées pour en générer de nouvelles constituées de bons éléments ;
- une hybridation AP/GP ou AP générerait des populations de solutions pour des îlots où la GP ferait évoluer les arbres, avec bien sûr un mécanisme de communication pour la mise à jour de la phéromone ;
- l'utilisation d'une valeur heuristique ? La question ici est très vaste : comment évaluer si un sous-arbre ou un nœud est bon sans avoir une vision globale de la solution ? La solution ne serait-elle pas comme pour certaines applications de la PG, où une adaptation au type de problème à résoudre a été intégrée dans les opérateurs ou via une forme de typage ?

Il reste également beaucoup de voies à explorer au niveau de l'utilisation de la mémoire. Par rapport aux algorithmes génétiques, les fourmis utilisent déjà un type de mémoire plus subtil. On pourrait s'inspirer de certains comportements humains, comme par exemple la réaction d'orientation qui favoriserait l'exploitation d'un élément peu utilisé mais que l'on vient de trouver dans une bonne solution ...

Bibliographie

[1] E.H.L. Aarts and P.J.M. Van Laarhoven. Statistical cooling : a general approach to combinatorial optimization problems. *Philips J. Res.*, 40(193), 1985.

[2] R.C. Arkin. *Behavior-based robotics.* MIT Press, Cambridge, Massachusetts, 1998.

[3] V. Bachelet. *Métaheuristique parallèles hybrides : applications au problème d'affectation quadratique.* PhD thesis, Université de Lille 1, 1999.

[4] T. Bäck, editor. *Proceedings of the 7th International Conference on Genetic Algorithms*, East Lansing, Michigan, USA, jul 1997. Morgan Kaufmann.

[5] A. Baddeley. *La mémoire humaine - théorie et pratique.* Presses Universitaires de Grenoble (pug), 1992.

[6] S. Baluja. Population-Based Incremental Learning : A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, 1994.

[7] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University, 1995.

[8] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In Prieditis and Russell [173], pages 38–46.

[9] W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors. *Genetic and Evolutionary Computation Conference (GECCO-99)*, Orlando, Florida USA, July 13-17 1999. Morgan Kaufmann, San Francisco, CA.

[10] Wolfgang Banzhaf, Peter Nordin, Robert Keller, and Frank Francone. *Genetic Programming An Introduction.* Morgan Kaufmann, 1999.

[11] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA, Journal on computing 6*, pages 126–140, 1994.

[12] A. Bauer, B. Bullnheimer, R. Hartl, and C. Strauss. An Ant Colony Approach for the Single Machine Total Tardiness Problem. POM Working Paper 5/99, Department of Management Science, University of Vienna, 1999.

[13] A. Bauer, B. Bullnheimer, R.F. Hartl, and C. Strauss. An ant colony optimization approach for the single machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, Washington, USA, 1999.

[14] M.S. Bazaraa and H.D. Sherali. Benders'partitionning scheme applied to a new fomulation of the quadratic assignment problem. *Naval Research Logistics Quarterly*, 27 :29–41, 1980.

[15] D. Bertsekas and R. Gallager. *Data networks.* Prentice-Hall, 1992.

[16] G. Bilchev and I. Parmee. The ant colony metaphor for searching continuons design spaces. In T. Fogarty, editor, *Lecture Notes in Computer Science*, volume 993, pages 24–39. Springer Verlag, 1995.

[17] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence : From Natural to Artificial Systems.* Oxford University Press, New York, 1999.

[18] E. Bonabeau, F. Henaux, S. Guerin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in telecommunications networks with "smart" ant-like agents. Working Paper 98-01-003, Santa Fe Institute, 1998.

[19] E. Bonabeau and G. Theraulaz. *Intelligence Collective.* Hermes, 1994.

[20] M.F. Bramlette and E.E. Bouchard. *Handbook of genetic algorithms*, chapter 10 - Genetic Algorithms in Parametric Design of Aircraft. VNR Computer Library, 1991.

[21] B. Bullnheimer. *Ant Colony Optimization in Vehicle Routing.* Doctoral thesis, University of Vienna, January 1999.

[22] B. Bullnheimer, R.F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In *Second International Conference on Metaheuristcs*, Sophia-Antipolis, France, July 21-24 1997.

[23] B. Bullnheimer, R.F. Hartl, and C. Strauss. A new rank based version of the ant system - a computational study. Working Paper 3/97, Institute of Management Science, University of Vienna, Austria, 1997.

[24] B. Bullnheimer, R.F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics : Advances and Trends in Local Search Paradigms for Optimization.* Kluwer, Boston, 1999.

[25] B. Bullnheimer, R.F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research : Nonlinear Economic Dynamics and Control*, 1999.

[26] B. Bullnheimer, R.F. Hartl, and C. Strauss. A new rank based version of the ant system : A computational study. *Central European Journal for Operations Research and Economics*, 7(1) :25–38, 1999.

[27] B. Bullnheimer, G. Kotsis, and C. Strauss. Parallelization strategies for the Ant System. In R. De Leone, A. Murli, P. Pardalos, and G. Toraldo, editors, *High Performance Algorithms and Software in Nonlinear Optimization*, volume 24 of *Applied Optimization*, pages 87–100. Kluwer, Dordrecht, 1997.

[28] R.E. Burkard, E. Çela, P.M. Pardalos, and L.S. Pitsoulis. *Handbook of Combinatorial Optimization*, chapter The Quadratic Assignment Problem, pages 241–338. P.M. Pardalos and D.-Z. Du, eds., Kluwer Academic Publishers, 1998.

[29] R.E. Burkard, S.E. Karisch, and F. Rendl. Qaplib - a quadratic assignment problem library. *Journal of Global Optimization-10*, pages 391–403, 1997.

[30] R.E. Burkard and J. Offermann. Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. *Zeitschrift für Operations Research*, 21 :B121–B132, 1977.

[31] V. Campos, M. Laguna, and R. Martin. *Scatter search for the linear ordering problem*, chapter 21, pages 332–339. McGraw-Hill, 1999. In New idea in optimization, D. Corne and M. Dorigo and F. Glover (Eds.).

[32] C. Caux, H. Pierreval, and M.C. Portman. Les algorithmes génétiques et leur application d'ordonnancement. *Automatique, Productique Informatique Industrielle*, 29(4-5) :409–443, 1995.

[33] M.Q. Chung. *Application des algorithmes génétiques à la résolution de problémes et à la commande de systèmes.* PhD thesis, Université de Paris XII, 1993.

[34] D. Cliff, P. Husbands, J.A. Meyer, and Stewart W., editors. *Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3.* MIT Press, Cambridge, Massachusetts, 1994.

[35] J.P. Cohoon and W.D. Paris. Genetic placement. *IEEE Trans. CAD*, CAD-6(6) :956–964, 1987.

[36] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In Varela and Bourgine [229], pages 134–142.

[37] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant System for job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1) :39–53, 1994.

[38] D.T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research 46*, pages 93–100, 1990.

[39] B. Corbara, A. Drogoul, D. Fresneau, and S. Lalande. *Towards a Practice of Autonomous Systems II*, chapter Simulating the Sociogenesis Process in Ant Colonies with MANTA. MIT Press, Cambridge, 1993.

[40] O. Cordón, I. Fernández de Viana, F. Herrera, and L. Moreno. A new ACO model integrating evolutionary computation concepts : The best-worst ant system. In Dorigo [59].

[41] D. Corne, M. Dorigo, and F. Glover, editors. *New idea in optimization.* McGraw-Hill, 1999.

[42] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimisation.* McGraw-Hill, London, UK, 1999.

[43] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48 :295–305, 1997.

[44] T. Crainic, M. Gendreau, P. Soriano, and M. Toulouse. A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations Research*, 42((1-4)) :359–383, 1993.

[45] V.D. Cung, T. Mautor, P. Michelon, and A. Tavares. A scatter search based approach for the quadratic assignment problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of ICEC'97*, pages 165–170. IEEE Press, 1997.

[46] C. Darwin. *On the origin of species (De l'origine des espèces).* John Murrey, London, 1859.

[47] L. Davis. *The genetic algorithm handbook*, chapter 17. Ed. New York : Van Nostrand Reinhold, 1991.

[48] L. Davis, editor. *Handbook of genetic algorithms.* VNR Computer Library, 1991.

[49] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[50] J.-L. Deneubourg, S. Goss, N.R. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The dynamics of collective sorting : robot-like ant and ant-like robots. In Meyer and Wilson [151], pages 356–365.

[51] J.L. Deneubourg, S. Aron, S. Goss, and J.M. Pasteels. The self organizing exploratory pattern of the argentine ant. *J. Insect Behavior*, 3 :159–168, 1990.

[52] Patrik D'haseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pages 256–261, Orlando, Florida, USA, 1994.

[53] G. Di Caro and M. Dorigo. AntNet : A mobile agents approach to adaptive routing. Technical Report 97-12, IRIDIA, Université Libre de Bruxelles, Belgium, 1997.

[54] G. Di Caro and M. Dorigo. Ant Colonies for Adaptive Routing in Packet-switched Communications Networks. In Eiben et al. [69].

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

[56] G. Di Caro and M. Dorigo. Mobile agents for adaptive routing. In *31st Hawaii International Conference on System*, Big Island of Hawaii, January 6-9, 1998.

[57] G. Di Caro and M. Dorigo. Two ant colony algorithms for best-effort routing in datagram networks. In *Proceedings of PDCS'98 - 10th International Conference on Parallel and Distributed Computing and Systems, Las Vegas, Nevada, October 28-31*, 1998.

[58] M. Dorigo. *Optimization, Learning and Natural Algorithms (in Italian).* PhD thesis, Politecnico di Milano, Italy, 1992.

[59] M. Dorigo, editor. *ANTS'2000 - From Ant Colonies to Artificial Ants : Second International Workshop on Ant Algorithms, Brussels, Belgium, September 8-9*, 2000.

[60] M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stygmergy. *Future Generation Computer Systems*, 16(8) :851–871, 2000.

[61] M. Dorigo, G. Di Caro, and T. Stützle, editors. *Special issue on Ant Colony Optimization to be published in the Future Generation Computer Systems journal.* North Holland, 2000.

[62] M. Dorigo and G. Di Caro. The Ant Colony Optimization Meta-Heuristic. In Corne et al. [42], pages 11–32.

[63] M. Dorigo and L.M. Gambardella. A study of some properties of Ant-Q. In Voigt et al. [230], pages 656–665.

[64] M. Dorigo and L.M. Gambardella. Ant colonies for the Traveling Salesman Problem. *BioSystems*, 43 :73–81, 1997.

[65] M. Dorigo and L.M. Gambardella. Ant Colony Sytem : A cooperative learning approach to the Travelling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1) :53–66, 1997.

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

[67] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1) :29–41, 1996.

[68] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic : Algorithms, applications and advances. In F. Glover and G. Kochenberger, editors, *Metaheuristics Handbook*, International Series in Operations Research and Management Science. Kluwer, 2001. Technical Report IRIDIA/2000-32, IRIDIA, Université Libre de Bruxelles, Belgium, 2000.

[69] A.E. Eiben, T. Bäck, H.-P. Schwefel, and M. Schoenauer, editors. *Fifth International Conference on Parallel Problem Solving from Nature (PPSN V)*. Springer-Verlag, New York, 1998.

[70] A.E. Eiben, R. Hintering, and Z. Michalewicz. Parameter control in evolutionay algoritms. In *IEEE Trans. on Ev. Comp. vol 3, n 4*, pages 124–141, 1999.

[71] A.N. Elshafei. Hospital layout as a quadratic assignment problem. *Operations Research Quarterly*, 28 :167–179, 1977.

[72] U. Faigle and W. Kern. Some convergence results for probabilistic tabu search. *ORSA Journal on Computing*, pages 32–39, 1992.

[73] E. Falkenauer. A new representation and opterator for genetic algorithm applied to groupingt problems. *Evolutionary computation, MIT*, 2(2), 1994.

[74] J. Ferber. *Les Systèmes multi-agents, vers une intelligence collective.* InterEdition, 1995.

[75] C. Fleurent and J. Ferland. Genetic hybrids for the quadratic assignement problem. *DIMACS Series in Math*, Theorical Comuter Sci(16) :190–206, 1994.

[76] C. Fleurent, F. Glover, P. Michelon, and Z. Valli. A scatter search approach for unconstrainted continuous optimization. In IEEE Press, editor, *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 643–648, 1996.

[77] Fogel, Owens, and Walsh. *Artificial Intelligence Through Simulated Evolution.* Wiley and sons, 1966.

[78] D. B. Fogel. *Evolutionary Computation : Toward a New Philosophy of Machine Intelligence.* New York : IEEE Press, 1995.

[79] Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, Edmund Ronald, and Marc Schoenauer, editors. *Proceedings of Evolution Artificielle 99 EA'99*, volume 1829 of *LNCS*, Dunkerque, France, nov 1999. Springer.

[80] Cyril Fonlupt and Denis Robilliard. Genetic programming with dynamic fitness for a remote sensing application. In *[191]*, pages 191–200, 2000.

[81] L. Françoise, N. Monmarché, and G. Venturini. Vers un robot modélisant la perception visuelle des fourmis. In *Actes des Colloques Insectes Sociaux*, volume 13, pages 169–172, Tours, France, 1-3 Septembre 1999.

[82] D. Fresneau. *Biologie et comportement social d'une fourmi ponérine néotropicale (Pachycondyla apicalis).* Thèse d'etat, Université de Paris XIII, Laboratoire d'Ethologie Expérimentale et Comparée, France, 1994.

[83] C.-L. Gallien. *Biologie.*, volume 2. Génétique. puf, 1994.

[84] L. M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3) :237–255, 2000.

[85] L. M. Gambardella, E. Taillard, and G. Agazzi. MACS-VRPTW : A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. In Corne et al. [42], pages 63–76.

[86] L. M. Gambardella, E. Taillard, and M. Dorigo. Ant Colonies for the QAP. *Journal of the Operational Research Society*, 50 :167–176, 1999.

[87] L.M. Gambardella and M. Dorigo. Ant-Q : A reinforcement learning approach to the Travelling Salesman Problem. In Prieditis and Russell [173], pages 252–260.

[88] L.M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *Proceedings of the third IEEE International Conference on Evolutionary Computation*, pages 622–627, Nagoya, Japan, 1996. IEEE Press.

[89] L.M. Gambardella and M. Dorigo. HAS-SOP : An Hybrid Ant System for the Sequential Ordering Problem. Technical Report 97-11, IDSIA, Lugano, Switzerland, 1997.

[90] L.M. Gambardella, É.D. Taillard, and M. Dorigo. Ant colonies for the QAP. Technical Report 97-4, IDSIA, Lugano, Switzerland, 1997.

[91] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of $\mathcal{NP}$-Completeness.* Freeman, San Francisco, California, 1979.

[92] A.M. Geoffrion and G.W. Grave. Scheduling parallel production lines with changeover costs : Practical applications of a quadratic assignment/lp approach. *Operatations Research*, 1976.

[93] P.C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics*, 10 :305–311, 1962.

[94] F. Glover. Heuristic for integer programming using surrogate constraints. *Decision Science*, 8(1) :156–166, 1977.

[95] F. Glover. Future paths for integer programming and link to ai. *Computers and Operations Research*, pages 533–549, 1986.

[96] F. Glover. Tabu search - part 1. *ORSA Journal on Computing*, 1((3)) :190–206, 1989.

[97] F. Glover. Tabu search - part 2. *ORSA Journal on Computing*, 2((1)) :4–32, 1990.

[98] F. Glover. Tabu search fundamentals and uses. Technical report, University of Colorado, Boulder, 1995.

[99] F. Glover. A template for scatter search ans path reliking. *Artificial Evolution, lecture notes in Computer Science 1363*, 49 :231–255, 1998. J-K. Hao and E. Lutton and E. Ronald and M. Schoenauer and D. Snyers (Eds).

[100] F. Glover. *Scatter search and path relinking*, chapter 19, pages 298–316. McGraw-Hill, 1999. In New idea in optimization, D. Corne and M. Dorigo and F. Glover (Eds.).

[101] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, 1989.

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

[103] E.D. Goodman, A.Y. Tetelbaum, and V.M. Kureichik. A genetic algorithm approach to compaction, bin packing, and nesting problems. Technical Report 940702, Case center for computer-aided engeneering and manufacturing, Michigan state university, 1994.

[104] S. Goss, S. Aron, J.L. Deneubourg, and J.M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenchaften*, 76 :579–581, 1989.

[105] P.P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *bellicositermes natalensis* et *cubitermes sp.* la théorie de la stigmergie : essai d'interprétation du comportement des termites contructeurs. *Insectes Sociaux*, 6 :41–80, 1959.

[106] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-16, 1986.

[107] J.J. Grefenstette. Genetic algorithm and their applications. In NJ : Lawrence Erlbaum Associates Hillsdale, editor, *Proceeding of the 2nd Int. Conf. On Genetic Algorithm*, 1987.

[108] L. Gritz and J.K. Hahn. Genetic programming evolution of controllers for a 3-d charater animation. In *Proc. of Genetic Programming'97 Conference*, 1997.

[109] Z. Hafidi, E-G. Talbi, and J-M. Geib. Mars : Un ordonnanceur adaptatif d'applications parallèles dans un environnement multi-utilisateurs. In *RenPar'8-8ème Rencontres Francophones du Parallélisme*, pages 37–40, Bordeaux, France, Mai 1996.

[110] W.D. Hamilton. The genetical evolution of social behaviour i, ii. *Journal of Theorical Biology*, 7 :152, 1964.

[111] Simon Handley. Predicting whether or not a 60-base DNA sequence contains a centrally-located splice site using genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming : From Theory to Real-World Applications*, pages 98–103, Tahoe City, California, USA, 9 July 1995.

[112] D.R. Heffley. Assigning runners to a relay team. *Optimal Startegies in Sports*, pages 169–171, 1977. S.P. Ladany and R.E. Machol, eds, North-Holland, Amsterdam.

[113] A. Hertz and D. De Werra. Using tabu search techniques for graph coloring. *Computing*, 39 :345–351, 1987.

[114] M. Heusse, D. Snyers, S. Guérin, and P. Kuntz. Adaptive agent-driven routing and load balancing in communication networks. Technical Report RR-98001-IASC, ENST de Bretagne, 1998.

[115] J. Holland. *Adaptation in natural and artificial systems.* MIT Press, 1975.

[116] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MPU, 1975.

[117] B. Hölldobler and E.O. Wilson. *The Ants.* Springer Verlag, Berlin, Germany, 1990.

[118] B. Hölldobler and E.O. Wilson. *Voyage chez les Fourmis.* Seuil, 1996.

[119] Fromman Hozlboog Verlag, editor. *Evolutionstrategie : Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*, Stuttgart, 1973.

[120] D.S. Johnson and L.A. McGeoch. The traveling salesman problem : a case study. In Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 215–310. John Wiley and Sons, Chichester, 1997.

[121] J. Kelly, B. Rangaswamy, and J. Xu. A scatter search-based learning algorithm for neural network training. *Journal of heuristics 2*, pages 129–146, 1996.

[122] A. Khanna and J Zinky. The revised ARPANET routing metric. In *SIGCOMM'89*, volume 19(4), pages 45–56, 1989.

[123] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220 :671–680, 1983.

[124] T. Kohonen. *Self-Organisation and Associative Memory.* Springer-Verlag, 1988. Second edition.

[125] T.C. Koopmans and M.J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25 :53–76, 1957.

[126] John Koza. *Genetic Programming : On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[127] John Koza. *Genetic Programming II : Automac Discovery of Reusable Programs.* The MIT Press, 1994.

[128] John Koza, Forrest Bennet III, David Andre, and Martin Keane. *Genetic programming III : Darwinian Invention and Problem Solving.* Morgan Kaufmann, 1999.

[129] John R. Koza, David Andre, Forrest H Bennett III, and Martin A. Keane. Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996 : Proceedings of the First Annual Conference*, pages 132–149, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[130] J. Krarup and P.M. Pruzan. Computer-aided layout design. *Mathematical Programming Study*, 9 :75–94, 1978.

[131] M. Laguna, H. Lourenco, and R Marti. Multi-objective scatter search for efficient assignments of proctors to examinations. Working paper, University of Colorado, 1999.

[132] D. Lambrinos, M. Maris, H. Kobayashi, T. Labhart, R. Pfeifer, and R. Wehner. An autonomous agent navigating with a polarized light compass. *Adaptive Behavior*, 6(1) :131–161, 1997.

[133] William B. Langdon. Size fair and homologous tree genetic programming crossovers. In *[9]*, pages 1092–1097, 1999.

[134] E.L. Lawler. The quadratic assignment problem. *Management Science*, 9 :586–599, 1963.

[135] G. Leguizamón and Z. Michalewicz. A new version of Ant System for subset problems. In *Proceedings of CEC99 - Congress on Evolutionary Computation, Washington DC*. IEEE Press, July 6-9 1999.

[136] Brian Lent. Evolution of trade strategies using genetic algorithms and genetic programming. In John R. Koza, editor, *Genetic Algorithms at Stanford 1994*, pages 87–98. Stanford Bookstore, Stanford, California, 94305-3079 USA, December 1994.

[137] Y.-C. Liang and A.E. Smith. An ant system approach to redundancy allocation. In NJ IEEE Press, Piscataway, editor, *In Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1478–1484, 1999.

[138] F.-T. Lin, C.-Y Kao, and C.-C. Hsu. Applying the genetic approach to simulated annealing in solving some $\mathcal{NP}$-hard problems. *IEEE Trans. on Syst. Man. and Cybernetic*, 23(6) :1752–1767, Nov-Dec 1993.

[139] E. Lutton and P. Martinez. A genetic algorithm for the detection of 2d geometric primitives in images. In *12-IPCR*, Jerusalem, Israel, 1994.

[140] G.S. Malkin and M.E. Steenstrup. *Routing in Communications Networks*, chapter 3, Distance-Vector Routing, pages 83–98. Prentice-Hall, 1995.

[141] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. Technical Report CSR 98-1, C. L. in Scienze dell'Informazione, Università di Bologna, Sede di Cesena, Italy, 1998.

[142] V. Maniezzo and A. Carbonaro. An ANTS heuristic for the Frequency Assignment Problem. Technical Report CSR 98-4, C. L. in Scienze dell'Informazione, Università di Bologna, Sede di Cesena, Italy, 1998.

[143] V. Maniezzo and A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8) :927–935, 2000.

[144] V. Maniezzo and A. Colorni. The Ant System applied to the Quadratic Assignment Problem. In *IEEE Transactions on Knowledge and Data Engineering*. to appear, 1999.

[145] V. Maniezzo, A. Colorni, and M. Dorigo. The Ant System applied to the Quadratic Assignment Problem. Technical Report 94-28, IRIDIA, Université Libre de Bruxelles, Belgium, 1994.

[146] R. Männer and B. Manderick, editors. *Second International Conference on Parallel Problem Solving from Nature (PPSN II)*. Elsevier Science, 1992.

[147] J.M. McQuillian, I. Richer, and E.C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transaction on Communications*, 28 :711–719, 1980.

[148] G.J. Mendel. *Experiments in Plant-hybridization.* Princeton University Press, 1865.

[149] D. Merkle, M. Middendorf, and H. Schmeck. Ant Colony Optimization for Resource-Constrainted Project Scheduling. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 893–900, Las Vegas, Nevada, July 10 - 12 2000. Morgan Kaufmann, San Francisco, CA.

[150] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculation by fast computing machines. *J. of chem. Physics*, 21 :1087–1092, 1953.

[151] J.A Meyer and S. Wilson, editors. *First International Conference on Simulation of Adaptive Behavior.* MIT Press, Cambridge, Massachusetts, 1990.

[152] Z. Michalewicz. *Genetic algorithms + Data structure = Evolution programs.* collection *Artificial Intelligence.* Springer Verlag, Berlin Heidelberg, 1992.

[153] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, third - 1999 edition, 1996.

[154] R. Michel and M. Middendorf. An Island Model Based Ant System with Lookahead for the Shortest Supersequence Problem. In Eiben et al. [69], pages 692–701.

[155] R. Michel and M. Middendorf. An ACO algorithm for the Shortest Common Supersequence Problem. In Corne et al. [42], pages 51–61.

[156] Melanie Mitchell. *An Introduction to Genetic Algorithms.* The MIT Press, 1996.

[157] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[158] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[159] N. Monmarché. *Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation.* Thèse de doctorat, Laboratoire d'Informatique, Université de Tours, décembre 2000.

[160] N. Monmarché, G. Nocent, M. Slimane, and G. Venturini. Imagine : a tool for generating HTML style sheets with an interactive genetic algorithm on genes frequencies. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC'99)*, volume 3, pages 640–645, Tokyo, Japan, October 12-15 1999.

[161] N. Monmarché, G. Nocent, M. Slimane, G. Venturini, and P. Santini. Génération de feuilles de style pour site web par un algorithme génétique interactif. In *Hypertextes et Hypermédias, Réalisations, Outils & Méthodes (H2PTM'99)*, pages 271–282, Paris, France, 23-24 septembre 1999. Hermes.

[162] N. Monmarché, G. Nocent, G. Venturini, and P. Santini. On generating HTML style sheets with an interactive genetic algorithm based on gene frequencies. In *Artificial Evolution 99, Lecture Notes in Artificial Intelligence*, volume 1829, pages 99–110, Dunkerque, France, November 3-5 1999. Springer-Verlag.

[163] N. Monmarché, E. Ramat, G. Dromel, M. Slimane, and G. Venturini. On the similarities between AS, BSC and PBIL : toward the birth of a new meta-heuristic. Rapport interne 215, Laboratoire d'Informatique de l'Université de Tours, E3i Tours, Janvier 1999. 14 pages.

[164] N. Monmarché, M. Slimane, and G. Venturini. Antclass : discovery of clusters in numeric data by an hybridization of an ant colony with the kmeans algorithm. Rapport interne 213, Laboratoire d'Informatique de l'Université de Tours, E3i Tours, Janvier 1999. 21 pages.

[165] N. Monmarché, G. Venturini, and M. Slimane. Antclass, découverte de classes dans des données numériques grâce à l'hybridation d'une colonie de fourmis et l'algorithme des centres mobiles. In *Conférence d'Apprentissage*, pages 169–176, Ecole Polytechnique, Palaiseau, France, 15-18 juin 1999.

[166] N. Monmarché, G. Venturini, and M. Slimane. On how *Pachycondyla apicalis* ants suggest a new search algorithm. *Future Generation Computer Systems*, 16(8) :937–946, 2000.

[167] J. Moy. Ospf version 2, 1994. Request For Comments (RFC) 1583, Network Working Group.

[168] Yuichi Nagata and Shigenobu Kobayashi. Edge assembly crossover : A high-power genetic algorithm for the traveling salesman problem. In *[4]*, pages 450–457, 1997.

[169] G. Navarro Varela and M.C. Sinclair. Ant Colony Optimisation for Virtual-Wavelength-Path Routing and Wavelength Allocation. In *Congress on Evolutionary Computation*, Washington DC, USA, July 1999.

[170] C.E. Nugen, T.E. Wollmann, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Journal of operation research*, 16 :150–173, 1969.

[171] M.W. Padberg and M.P. Rijal. *Location scheduling design and integer programming*. Kluwer Academic Publishers, Boston, 1996.

[172] R. Pfeifer, B. Blumberg, J.A. Meyer, and S.W. Wilson, editors. *Fith International Conference on Simulation of Adaptive Behavior*. MIT Press, 1998.

[173] A. Prieditis and S. Russell, editors. *Twelfth International Conference on Machine Learning, Lake Tahoe, California*. Morgan Kaufmann, San Mateo, California, 1995.

[174] R. Quinn and K. Espenschield. Control of a hexapod robot using a biologically inspired neural network. In R. Beer, R. Ritzmann, and McKenna T., editors, *Biological Neural Networks in Invertebrate Neuroethothology and Robotics*, pages 365–381. Academic Press, San Diego, CA, 1993.

[175] H. Ramalhinho Lourenco and D. Serra. Adaptive approach heuristics for the generalized assignment problem. Technical report, Department of economics and management, Universitat Ponpeu Fabra, Barcelona, Spain, 1998.

[176] C.L. Reeves. A genetic approach to bin-packing". In *Proc. of the INWGA*, pages 35–49, Vaasa, January 1995.

[177] C.R. Reeves. *Modern Heuristic techniques for combinatorial problems*, chapter 4 - Genetic Algorithms. McGraw-Hill, 1995.

[178] Denis Robilliard and Cyril Fonlupt. A shepherd and a sheepdog to guide evolutionary computation ? In *[79]*, pages 277–291, 1999.

[179] Y. Rochat and E.D. Taillard. Probabilistic diversification and intensification in local search for vehicule routing. *Journal of heuristics*, pages 147–167, 1995.

[180] O. Roux, C. C. Fonlupt, D. Robilliard, and E.-G. Talbi. Antabu. In *From Ant Colonies to Artificial Ants : First International Workshop on Ant Colony Optimization (ANTS'98)*, Brussels, Belgium, 1998.

[181] O. Roux and C. Fonlupt. Ant programming : Or how to use ants for automatic programming. In *From Ant Colonies to Artificial Ants : 2nd International Workshop on Ant Colony Optimization*, Brussels, Belgium, September 8-9 2000.

[182] Olivier Roux, Cyril Fonlupt, and Denis Robilliard. Co-operative improvement for a combinatorial optimization algorithm. In *[79]*, pages 231–241, 1999.

[183] S. Rudlof and M. Koeppen. Stochastic hill climbing with learning by vectors of normal distributions. In *Proceedings of the first Online Workshop on Soft Computing*, 1996. `http ://www.bioele.nuee.nagoya-u.ac.jp/wsc1/papers/p077.html`.

[184] A. Russell, Thiel D., and A. Mackay-Sim. Sensing odour trails for mobile robot navigation. In *IEEE International Conference on Robotics and Automation*, pages 2672–2677, San Diego, CA, may 1994.

[185] Khuri S. and M. Schütz. Evolutionary heuristique for the bin packing problem. In *Proc. of ICANNGA'95*, Alès, France, April 1995.

[186] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23 :555–565, 1976.

[187] R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2) :123–141, 1997.

[188] J.D. Schaffer and A. Morishima. An adaptative crossover distribution machenism for genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the 2^{nd} International Conference on Genetic Algorithms*, pages 36–40. Morgan Kaufmann, 1987.

[189] B. Schatz, S. Chameron, G. Beugnon, and T.S. Collett. The use of path integration to guide route learning in ants. *Nature*, 399 :769–772, 24/06/1999. Macmillan Publishers Ltd.

[190] M. Schoenauer. Shape representation fot evolutionary optimization and identification. In *EUROGEN'96*, 1996.

[191] M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.-J. Merelo, and H.-P. Schwefel (Eds), editors. *Sixth International Conference on Parallel Problem Solving from Nature*. Lecture Notes in Computer Science 1917, Springer Verlag, September 16-20 2000.

[192] M. Schoenauer, F. Jouve, and L. Kallel. *Evolutionary Computation in Engineering*, chapter chapter Identification of mechanical inclusions, pages 477–494. Springer-Verlag, 1997.

[193] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2) :169–207, 1997.

[194] R. Schoonderwoerd, O. Holland, and Bruten J. Ant-like agents for load balancing in telecommunications networks. In *Agents'97*, pages 209–216, Marina del Rey, Canada, 1997. ACM Press.

[195] H.-P. Schwefel. *Numerical Optimization of Computer Models.* John Wiley & Sons, New-York, 1981. 1995 - 2^{nd} edition.

[196] H.P. Schwefel. *Numerical Optimization of Computer Models.* John Wiley & Sons, New-York, second edition, 1995.

[197] M. Sebag and A. Ducoulombier. Extending population-based incremental learning to continuous spaces. In Eiben et al. [69].

[198] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal of Computing*, 2(1) :33–45, 1990.

[199] C. Solnon. Ant-p-solveur : un solveur de contraintes à base de fourmis artificielles. In *Journées Francophones sur la Programmation Logique et par Contraintes*, pages 189–204. Hermes, juin 2000.

[200] C. Solnon. Solving permutation constraint satisfaction problems with artificial ants. In *Prooceedings of the 14th European Conference on Artificial Intelligence*, 2000.

[201] W.M. Spears. Adapting crossover in evolutionary algorithms. In J.R. McDonnell, R.G. Reynolds, and D.B. Fogel, editors, *Proceeding of the 4^{th} Annual Conference on Evolutionary Programming*, pages 367–384. MIT Press, 1995.

[202] L Steinberg. The backboard wiring problem : A placement algorithm. *SIAM Review*, 3 :37–50, 1961.

[203] T. Stützle. Max-min ant system for the quadratic assignment problems. Technical Report AIDA-97-04, Intellectics group, Department of Computer Science, Darmstadt University of Technology, Germany, 1997.

[204] T. Stützle. An ant approach to the Flow Shop Problem. In *EUFIT'98*, pages 1560–1564, Aachen, 1998.

[205] T. Stützle. *Local search algorithms for combinatorial problems : Analysis, improvements and new applications.* PhD thesis, Fachbereich Informatik, TU Darmstadt, Germany, 1998.

[206] T. Stützle and M. Dorigo. ACO algorithms for the Quadratic Assignment Problem. In Corne et al. [42], pages 33–50.

[207] T. Stützle and M. Dorigo. ACO algorithms for the Traveling Salesman Problem. In K. Miettinen, M. Mäkelä, P. Neittaanmäki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science : Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications.* John Wiley & Sons, 1999.

[208] T. Stützle and H. Hoos. The $\mathcal{MAX} - \mathcal{MIN}$ Ant System and local search for combinatorial optimization problems : Towards adaptive tools for global optimization. In *2nd Metaheuristics International Conference*, Sophia-Antipolis, France, July 21-24 1997.

[209] T. Stützle and H. Hoos. $\mathcal{MAX}-\mathcal{MIN}$ Ant System and local search for the Traveling Salesman Problem. In *Proceedings of the fourth International Conference on Evolutionary Computation*, pages 308–313. IEEE Press, 1997.

[210] T. Stützle and H. Hoos. Improvements on the Ant System : Introducing the $\mathcal{MAX} - \mathcal{MIN}$ Ant System. In *Third International Conference on Artificial Neural Networks and Genetic Algorithms*, University of East Anglia, Norwich, UK, 1997. Springer Verlag.

[211] T. Stützle and H. Hoos. Max-min ant system and local search for combinatoriel optimisation problems. In S. Voss, S. Mertello, I.H. Osman, and C. Roucairol, editors, *Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization*, pages 313–329. Kluwer Academics, Boston, 1999.

[212] T. Stützle and H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8) :889–914, 2000.

[213] D. Subramanian, P. Druschel, and J. Chen. Ants and reinforcement learning : A case study in routing in dynamic networks. In *IJCAI (2)*, pages 832–839, 1997.

[214] Gilbert Syswerda. A study of reproduction in generational and steady-state genetic algorithms. In Rawlins, editor, *Foundation of Genetic Algorithms*. Morgan Kaufman, 1991.

[215] E. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science 3*, pages 87–103, 1995.

[216] E. Taillard and L. M. Gambardella. An ant approach for Structured Quadratic Assignment Problems. In *2nd Metaheuristics International Conference*, Sophia-Antipolis, France, July 21-24 1997.

[217] E.D. Taillard. Robust taboo search fot the quadratique assignment problem. *Parallel Computing*, 17 :443–455, 1991.

[218] E.D. Taillard. *La programmation à mémoire adaptative et les algorithmes pseudo-gloutons : nouvelles perspectives pour les méta-heuristiques*. PhD thesis, Laboratoire PRiSM, Université de Versailles Saint-Quentin-en-Yvelines, 1998. Thèse d'habilitation.

[219] E.D. Taillard, L-M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive memory programming : A unified view of meta-heuristics. In *EURO XVI Conference Tutorial and Research Reviews booklet*, Brussels, 1998. Technical report IDSIA-19-98, IDSIA, Lugano, 1998.

[220] E.-G. Talbi. *Contributions à la résolution parallèle de problèmes d'optimisation combinatoire*. PhD thesis, Université de Lille 1, 2000. Habilitation à Diriger des recherches.

[221] E.-G. Talbi, P. Bessière, J.-M. Ahuctzin, and E. Mazer. Parallel robot motion planning in a dynamic environment. In *Second Joint int. conf. on simulation of Adaptive Beahvior SAB 92 : From animals to animats*, pages 7–11, Honolulu, Hawai, 1992. MIT Press.

[222] E-G. Talbi, Z. Hafidi, and J-M. Geib. Parallel adaptive tabu search for large optimization problems. In *MIC'97-2nd Metaheuristics International Conference*, Sophia Antipolis, France, Juillet 1997.

[223] E-G Talbi, O. Roux, C. Fonlupt, and D. Robillard. Parallel ant colonies for combinatorial optimization problems. In J. Rolim, editor, *Workshop on Biologically Inspired Solutions to Parallel Processing Systems*. Springer-Verlag, 1999.

[224] E-G Talbi, O. Roux, C. Fonlupt, and D. Robillard. Parallel ant colonies for the quadratic assignement problem. *Future Generation Computer Systems*, 17(4) :441–449, 2001.

[225] EG Talbi. *Solutions to Parallel and Distributed Computing Problems : Lessons from Biological Sciences*, chapter A Taxonomy of Hybrid Metaheuristics. John Wiley and Sons, 2000. B.A. Zomaya (Ed.).

[226] I. Ugi, J. Bauer, J. Friedrich, J. Gasteiger, C. Jochum, and W Schubert. Neue anwendungsgebiete für computer in des chemie. *Angewandte Chemie*, 91 :99–111, 1979.

[227] R. Van der Put. Routing in the faxfactory using mobile agents. Technical report, KPN research, The Netherlands, 1998.

[228] R. Van der Put and L. Rothkrantz. Routing in packet switched networks using agents. *Simulation Practice and Theory*, 1999.

[229] F. Varela and P. Bourgine, editors. *First European Conference on Artificial Life, Paris, France*. MIT Press, Cambridge, Massachusetts, 1991.

[230] H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors. *Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV)*. Springer-Verlag, Berlin, September 22-27 1996.

[231] I.A. Wagner and A.M. Bruckstein. Hamiltonian(t) - an ant-inspired heuristic for recognizing hamiltonian graphs. In *Conference on Evolutionary Computation*, 1999.

[232] I.A. Wagner, M. Lindenbaum, and A.M. Bruckstein. ANTS : Agent, Networks, Trees, and Subgraphs. *Future Generation Computer Systems*, 16(8) :915–926, 2000.

[233] B. Werber. *Les fourmis*. Albin Michel, 1991.

[234] T. White and B. Pagurek. Application oriented routing with biologically-inspired agents. In Banzhaf et al. [9], page 1453. Poster paper.

[235] T. White, B. Pagurek, and F. Oppacher. Connection management using adaptive mobile agents. In H.R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pages 802–809. CSREA Press, 1998.

[236] D. Whitley. Fundamental principles of deception in genetic search. In G.J.E. Rawlins, editor, *First workshop on Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann, 1991.

[237] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Building better test functions. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 239–246. Morgan Kaufmann, San Francisco, CA, 1995.

[238] E.O. Wilson. *Sociobiology, The New Synthesis*. Harward University Press, 1975.

[239] M. Wodrich. *Ant Colony Optimisation, an empirical investigation into an ant colony metaphor for continuous function optimisation*. Undergraduate thesis, Department of Electrical and Electronic Engineering, University of Cape Town, 1996.

[240] Douglas Zongker and Bill Punch. lilgp 1.01 user's manual. Technical report, Michigan State University, USA, 26 March 1996.

Titre :
La mémoire dans les algorithmes à colonie de fourmis :
applications à l'optimisation et à la programmation automatique

Résumé : Dans ce mémoire, nous présentons les méta-heuristiques inspirées du comportement des fourmis lors de la recherche de nourriture, les OCF. Nous confrontons ces méthodes face aux principales méta-heuristiques connues. Pour cela, nous proposons de nous placer sous le point de vue de l'utilisation de la mémoire et nous présentons taxinomie qui étend celle des AMP. Nous proposons deux nouvelles adaptations du modèle des fourmis. La première est l'algorithme ANTabu, il s'agit d'une méthode hybride pour la résolution du PAQ. Il associe l'utilisation des fourmis artificielles et d'une méthode de recherche locale robuste : la recherche tabou. Le parallélisme intrinsèque des systèmes de fourmis nous a amené à développer un modèle parallèle pour ANTabu. Cette méthode intègre également une puissante fonction de diversification et l'utilisation de bornes qui lui permettent d'éviter d'être piégé au niveau d'optima locaux. La seconde application développée est AP, cet algorithme est l'adaptation du modèle de coopération des fourmis à la programmation automatique. Son mécanisme de fonctionnement est simple, puisque à chaque itération on crée une nouvelle population en utilisant l'information emmagasinée par la phéromone. L'intérêt de cette gestion de l'information est qu'elle n'utilise pas de mécanismes complexes. Nous présentons cette méthode face à l'algorithme de base tel que Koza l'a défini.

Mots-clés : fourmis artificielles, algorithme à essaims, mémoire, optimisation, QAP, programmation automatique.

Title :
Memory in ants colony algorithms : applications to optimization and automatic programming

Abstract : This thesis presents meta-heuristic based on the behaviour of natural ants looking for food. These heuristics are known as Ants Colony Optimization or ACO. We propose to compare the ACO paradigm with other well-known heuristics with regards to the use of the memory. Then, we introduce two applications of the ACO algorithms. The first application, ANTabu is an ACO scheme for the QAP. ANTabu combines the ants' paradigm with a robust local search technique (Tabu search). A parallel model developed for ANTabu is introduced. The second application lies in the machine-learning field. This scheme called AP (Automatic Programming) applies the cooperative behaviour of ants to automatically build programs. This method is then compared to the classical automatic generation of programs : Genetic Programming.

Keywords : Artificial ants, swarm algorithms, memory, optimization, QAP, automatic programming.

Laboratoire d'Informatique du Littoral, UPRES-JE 2335, Axe MESC (Modélisation, Evolution et Simulation des systèmes complexes), Maison de la recherche Blaise Pascal, 50 rue Ferdinand Buisson, B.P. 719, 62228 Calais cedex (http ://www-lil.univ-littoral.fr)