UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

COMME EXIGENCE PARTIELLE DE LA MAÎTRISE EN INFORMATIQUE

OFFERTE EN VERTU D'UN PROTOCOLE D'ENTENTE AVEC

L'UNIVERSITÉ DU QUÉBEC À MONTRÉAL

PAR

SARA MORIN

ALGORITHME DE FOURMIS AVEC APPRENTISSAGE ET COMPORTEMENT SPÉCIALISÉS POUR L'ORDONNANCEMENT DE VOITURES

AOÛT 2005

Bibliothèque
Paul-Émile-Boulet

UQAC
Université du Québec
à Chicoutimi

Mise en garde/Advice

Afin de rendre accessible au plus grand nombre le résultat des travaux de recherche menés par ses étudiants gradués et dans l'esprit des règles qui régissent le dépôt et la diffusion des mémoires et thèses produits dans cette Institution, **l'Université du Québec à Chicoutimi (UQAC)** est fière de rendre accessible une version complète et gratuite de cette œuvre.

Motivated by a desire to make the results of its graduate students' research accessible to all, and in accordance with the rules governing the acceptation and diffusion of dissertations and theses in this Institution, the **Université du Québec à Chicoutimi (UQAC)** is proud to make a complete version of this work available at no cost to the reader.

L'auteur conserve néanmoins la propriété du droit d'auteur qui protège ce mémoire ou cette thèse. Ni le mémoire ou la thèse ni des extraits substantiels de ceux-ci ne peuvent être imprimés ou autrement reproduits sans son autorisation.

The author retains ownership of the copyright of this dissertation or thesis. Neither the dissertation or thesis, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

Résumé

Les problèmes d'optimisation combinatoire faisant partie de la classe de problèmes *NP-difficiles* sont abordés et traités de nombreuses manières dans la littérature. Lorsqu'il s'agit de résoudre ces problèmes, les méta-heuristiques sont des algorithmes bien adaptés aux conditions de résolution rencontrées dans les domaines pratiques. Elles offrent un compromis intéressant entre la qualité des solutions et le temps nécessaire à leur obtention. C'est en partie pourquoi le domaine des méta-heuristiques a pris beaucoup d'ampleur depuis les dernières années. De nombreuses méthodes y ont été introduites, dont l'Optimisation par Colonie de Fourmis (OCF) (*Ant Colony Optimization*) qui est un algorithme dont le comportement est basé sur celui des fourmis réelles. Depuis son introduction au début des années '90, l'OCF s'est montrée efficace pour la résolution de nombreux problèmes.

Parmi les problèmes d'optimisation combinatoire, on retrouve le Problème d'Ordonnancement de Voitures (POV) (*Car Sequencing Problem*). Ce problème théorique présente une problématique qui est également rencontrée dans les usines de production automobile. Le POV consiste à déterminer, à l'aide d'un carnet de commandes définissant les options requises pour chaque voiture à assembler, l'ordre dans lequel produire ces véhicules afin de minimiser les dépassements de capacité des postes de la chaîne de montage. Les véhicules sont produits en séquences dont l'ordre ne peut être modifié une fois le processus enclenché. Certaines options, comme le toit ouvrant, la traction intégrale ou les freins anti-blocage, nécessitent un temps de traitement plus important. Ces différences dans les temps de traitement impliquent des contraintes de capacité pour les postes de la chaîne qui y sont associés. Si, à un moment dans la séquence, la demande pour une option dépasse la capacité du poste qui la traite, il y a retard sur la chaîne et des coûts supplémentaires sont impliqués. En raison de sa structure et de ses contraintes globales de séquencement, le POV représente un défi intéressant en optimisation.

Ce travail de recherche propose un nouvel algorithme d'OCF spécifiquement adapté pour la résolution du POV. Un algorithme tiré de la littérature et conçu pour le POV sert à la fois de base de comparaison et d'algorithme de départ pour la réalisation du projet. La démarche consiste à modifier cet algorithme de départ,

étape par étape et de manière incrémentale, afin de le bonifier de certains éléments spécialisés pour résoudre le POV. Ces bonifications sont au nombre de trois : la première concerne les mécanismes d'apprentissage de l'algorithme, alors que les deux autres apportent une plus grande variation dans le comportement des fourmis.

Premièrement, la nature des contraintes du POV a inspiré une modification importante de la trace de phéromone. Cette trace est une mémoire collective à l'intérieur de laquelle est consignée l'apprentissage accumulé par la colonie. Un nouvel élément y est apporté afin de mieux l'adapter à la forme des contraintes du POV. Deuxièmement, des mécanismes permettant de diversifier la recherche de solutions sont proposés. Lorsqu'elle fait un choix, une fourmi cherche un compromis entre l'apprentissage et sa vision locale du problème. Les paramètres définissant l'importance relative de ces éléments sont ceux touchés par cette seconde modification. Finalement, un nouveau mode de construction des solutions est proposé. Une fourmi construit normalement sa solution de manière séquentielle, de la première position à la dernière. La bonification proposée permet à une fourmi de revenir sur des parties de solutions déjà fixées pour y insérer de nouveaux éléments.

Nous verrons que les trois éléments modifiés ou ajoutés à l'algorithme de départ contribuent à améliorer la performance générale. La nouvelle trace de phéromone permet d'identifier et d'encourager les motifs à répéter dans les solutions. Les variations des valeurs de paramètres aident à la diversification de la recherche dans l'espace de solutions et à mieux utiliser l'effort de calcul alloué. Quant au nouveau mode de construction, il se montre profitable dans certaines conditions et constitue une idée novatrice et généralisable pour traiter d'autres problèmes.

Les éléments proposés sont novateurs. Ce travail de recherche, loin de se prétendre exhaustif, apporte de nouveaux concepts avec un bon potentiel d'exploration. Les éléments présentés se sont montrés efficaces et pourraient aisément faire l'objet de travaux futurs pour en trouver des variantes encore plus intéressantes. Finalement, bien que le problème traité ici soit de nature théorique, les idées apportées sont transférables à la problématique industrielle pratique rencontrée par les fabricants automobiles.

Remerciements

Je désire d'abord témoigner ma gratitude envers ma directrice de travaux, Mme Caroline Gagné, qui a su me guider au cours de ce beau voyage. Son implication dans la réalisation de ce travail a été grandement appréciée, tout comme sa disponibilité et son savoir-faire. Merci pour ce beau mélange de professionnalisme et d'humanité.

Je dédie un merci spécial à Mme Gagné et à M. Marc Gravel pour m'avoir initiée au domaine de la recherche universitaire, pour m'avoir donné le goût de pousser plus loin l'expérience.

J'adresse un merci tout particulier et chaleureux à grand-maman Margot et à mes parents, Camil et Estelle, pour leur support et leur intérêt tout au long de mon cheminement. Vous êtes des personnes aussi généreuses que formidables et vous me rendez très fière.

Je remercie d'une manière aussi chaleureuse mon conjoint Simon, avec qui j'ai partagé mes études, mes voyages, mon bureau et maintenant ma vie. Merci de me permettre de garder au moins un pied sur terre, merci pour ce que tu es.

Je voudrais dire aux étudiants et professionnels du groupe de recherche en informatique que j'ai beaucoup apprécié travailler en leur compagnie. À ceux et celles qui y sont passés ou qui y séjournent présentement, merci pour le partage de vos connaissances, votre écoute et votre franche camaraderie.

Merci au personnel enseignant et administratif du département d'informatique et de mathématique (DIM) pour leur ouverture et leur disponibilité. J'adresse également mes remerciements aux membres de mon comité d'évaluation, M. Marc Gravel et Mme Odile Marcotte, pour leurs judicieux commentaires ayant permis d'améliorer la qualité de ce travail.

J'aimerais finalement remercier le Conseil de Recherche en Sciences Naturelles et en Génie du Canada (CRSNG), le Fond Québécois de la Recherche sur la Nature et les Technologies (FQRNT), ainsi que le programme de bourses d'excellence de l'UQAC pour leur support financier. Leurs généreuses contributions m'auront permis de me consacrer entièrement à mes travaux.

Table des matières

Liste des tableaux

Liste des figures

CHAPITRE 1 :

INTRODUCTION

Les problèmes d'optimisation combinatoire NP-difficiles sont présents partout dans notre environnement et ce, sous d'innombrables formes. Pensons, par exemple, à l'établissement des horaires de travail dans un hôpital, à la création des itinéraires pour les services de transport adapté ou à l'ordonnancement de la production dans les usines. Ces problèmes présentent un grand intérêt à la fois pour la communauté scientifique, qui travaille sur des versions épurées, et pour les praticiens en milieu industriel, qui font face à des problématiques rendues complexes par les contraintes de l'application réelle. Étant donné l'interrelation et la complémentarité entre ces domaines, la résolution des problèmes d'optimisation combinatoire est devenue une discipline d'actualité où se côtoient la recherche opérationnelle, l'informatique et les mathématiques.

Il existe une quantité impressionnante d'algorithmes pour résoudre chacun de ces problèmes, mais aucun qui garantisse l'optimalité des solutions et qui puisse s'exécuter dans un temps polynomial par rapport à la taille du problème. Dans un contexte réel où le problème à l'étude doit pouvoir être résolu dans un temps raisonnable et souvent de manière répétitive, les algorithmes de la classe des méta-heuristiques ont pris beaucoup d'ampleur depuis les dernières années. Cette popularité peut s'expliquer par le fait qu'elles offrent un compromis très intéressant entre le temps nécessaire à l'obtention d'une solution et la qualité de cette dernière, en plus d'être très flexibles et ainsi adaptables à une grande quantité de problèmes. Les méta-heuristiques sont des algorithmes approchés qui s'inspirent d'un élément naturel observé. La théorie de l'évolution de Darwin, le processus de

recuit du métal et la mémoire humaine ne sont que quelques exemples de concepts qui ont inspiré les chercheurs depuis quelques dizaines d'années.

Une méta-heuristique en particulier a été choisie pour faire l'objet de ce mémoire, soit l'optimisation par colonie de fourmis (*ant colony optimization*). Le comportement de cet algorithme est calqué sur celui des fourmis réelles qui communiquent entre elles à l'aide d'une substance chimique appelée la phéromone. L'Optimisation par Colonie de Fourmis (OCF) a été introduite au début des années '90 et s'est depuis montrée efficace pour une large classe de problèmes d'optimisation combinatoire, dont le problème du voyageur de commerce (*traveling salesman problem*) et le problème d'affectation quadratique (*quadratic assignment problem*).

Le problème d'ordonnancement de voitures (*car sequencing problem*) est un problème d'optimisation combinatoire bien connu dans la littérature et qui a été approché par de nombreuses méthodes. Bien que devenu une référence dans le domaine des problèmes de satisfaction de contraintes (*constraint satisfaction problems*), il est apparu relativement récemment dans celui des méthodes heuristiques. Sa structure particulière en fait un problème d'autant plus intéressant et qui représente un défi motivant pour l'adaptation d'une méthode de résolution.

Peu d'efforts ont été faits jusqu'à maintenant pour la résolution du Problème d'Ordonnancement de Voitures (POV) par les algorithmes de la classe des méta-heuristiques. C'est en partie ce qui motive l'orientation du présent travail de recherche vers l'adaptation d'un algorithme d'OCF dans le but de résoudre de

manière efficace le POV. Ce travail offre également la possibilité d'un transfert technologique potentiel vers une version industrielle du problème.

Les principaux concepts et courants des domaines des méta-heuristiques, de l'OCF et du POV sont présentés au Chapitre 2. L'optimisation combinatoire et les méta-heuristiques sont abordés de manière générale en première partie. Une fois situé parmi les méthodes de résolution existantes, l'OCF est expliqué en détails avec ses variantes et les principales applications à des problèmes d'optimisation combinatoire. Le POV est ensuite formalisé et décrit pour introduire le problème à l'étude. Un résumé des principaux algorithmes exacts et heuristiques utilisés pour sa résolution est présenté. Une brève description d'une version du problème en contexte industriel est ensuite donnée. Les objectifs spécifiques du présent travail sont finalement établis suite à cette revue de la littérature. De ces objectifs particuliers découlent trois grandes bonifications qui seront apportées à un algorithme d'OCF déjà existant.

Ces bonifications, présentées au Chapitre 3, contribuent à la spécialisation de l'algorithme pour la résolution du POV. Elles touchent la structure de la trace de phéromone, la valeur de certains paramètres en cours d'exécution et l'intégration d'un nouveau concept de construction dynamique des solutions. Pour chaque bonification, des explications détaillées sont données et une analyse des résultats obtenus suite aux tests numériques est faite.

D'abord, la nature du problème a inspiré la modification de la structure même de la trace de phéromone utilisée pour comptabiliser l'apprentissage de la colonie.

D'une matrice en deux dimensions associant une classe de voitures à chacune des autres, nous passons à une matrice à trois dimensions, où s'ajoute la distance entre les voitures. De cette façon, la trace de phéromone représente l'intérêt de placer deux voitures de classes données à une distance précise l'une de l'autre.

La seconde bonification apportée consiste en une variation en cours d'exécution des valeurs de certains paramètres de décision de la fourmi. Les paramètres touchés sont ceux qui dosent le compromis entre l'utilisation de l'apprentissage et la visibilité locale. En leur imposant une variation à peu près constante au cours de l'algorithme, on obtient des constructions de solutions sous différentes combinaisons de valeurs. De cette façon, on favorise tantôt l'apprentissage, tantôt les bénéfices locaux, ce qui apporte une plus grande variation dans le comportement des fourmis. Cette modification est d'ordre plus général et pourrait être appliquée à n'importe quel algorithme de fourmi, peu importe le problème traité.

La troisième et dernière bonification touche le mode de construction des solutions. Normalement, les fourmis construisent leurs solutions de manière séquentielle, en commençant par la première position de la séquence jusqu'à la dernière. Selon la modification apportée, les fourmis sont autorisées à certaines occasions à passer en mode de construction *dynamique* (par opposition à *séquentiel*) afin de tenter d'insérer une voiture parmi les positions déjà placées. Dans certains cas, le but est d'éliminer des conflits existants, et dans d'autres, de

perturber la fin de la séquence en cours. Comme la précédente, cette modification contribue à changer le comportement des fourmis.

En terminant, les conclusions générales de ce travail de recherche sont abordées au Chapitre 4. Nous y verrons également les avenues possibles pour de futurs projets de recherche et les généralisations possibles des concepts apportés.

CHAPITRE 2 :

OPTIMISATION PAR COLONIE DE FOURMIS ET PROBLÈME D'ORDONNANCEMENT DE VOITURES

2.1 Introduction

Afin d'amorcer le travail de réalisation présenté dans ce mémoire, il est essentiel de bien analyser les travaux réalisés par la communauté scientifique sur le sujet. Ces derniers sont présentés ici à travers les auteurs et courants les plus importants du domaine.

Ce chapitre a pour but de présenter une revue de la littérature relative aux deux sujets principaux de ce mémoire, soit l'optimisation par colonie de fourmis, une méta-heuristique dont le comportement est inspiré de celui des fourmilières réelles, et le problème d'ordonnancement de voitures.

La Section 2.2 porte sur les méta-heuristiques. Un rappel est d'abord fait sur les principaux concepts en optimisation combinatoire et les méthodes utilisées pour la résolution des problèmes. Parmi ces méthodes se trouvent les méta-heuristiques, définies par la suite.

L'optimisation par colonie de fourmis est abordée à la Section 2.3, d'abord en présentant les principes généraux de fonctionnement, puis en étudiant une version préliminaire, l'Ant System. L'algorithme détaillé de l'OCF actuel est ensuite expliqué. Le POV fait l'objet de la Section 2.4, où sont abordés la formulation et les principales méthodes utilisées dans la littérature pour le résoudre.

Suite à l'observation des tendances du domaine, il sera possible d'établir les objectifs du présent travail de recherche, qui sont présentés et décrits à la Section 2.5.

2.2 Les méta-heuristiques

Lorsqu'il est question d'optimisation combinatoire ou de résolution de problèmes NP-difficiles, les méta-heuristiques représentent un domaine en pleine effervescence. Alors qu'aucun algorithme exact s'exécutant en temps polynomial n'est connu pour ces problèmes, les méta-heuristiques offrent un compromis intéressant. Elles ne garantissent pas l'optimalité des solutions, mais elles s'exécutent dans un temps considérablement plus court que les méthodes dites « optimales ». Ces dernières sont très souvent inapplicables, parce que trop coûteuses en termes de temps de calcul, ou trop rigides. L'informatique contribue grandement au développement des méta-heuristiques en leur donnant une force et une capacité de calcul de plus en plus grande.

2.2.1 Mise en contexte

L'optimisation combinatoire est un sujet très exploité dans les domaines de la recherche opérationnelle, de l'informatique et des mathématiques discrètes. Les problèmes d'optimisation combinatoire peuvent être faciles à décrire, mais très difficiles à modéliser et à résoudre. Bon nombre d'entre eux font partie des problèmes dits NP-difficiles. Hao *et al.* [HAO'99] définissent un problème d'optimisation combinatoire comme un ensemble d'instances dont chacune est associée à un ensemble discret de solutions Z, un sous-ensemble X de Z représentant les solutions admissibles respectant un ensemble de contraintes et une fonction de coût f (ou fonction objectif) qui assigne à chaque solution $z \in X$, le

nombre $f(z)$, réel ou entier. Résoudre un tel problème, plus précisément une telle instance du problème, consiste à trouver une solution $z^* \in X$ optimisant la valeur de la fonction de coût f. Une telle solution est appelée *solution optimale* ou *optimum global*.

L'espace de solutions X pour une instance précise peut être imagé sous forme d'un graphique, tel qu'illustré à la Figure 2.1. Les différentes solutions sont représentées en abscisse (sur une échelle discrète) et les valeurs de la fonction $f(z)$ associées à chacune se trouvent en ordonnée. Une solution z_i est légèrement différente de z_{i-1} et de z_{i+1} et la différence entre deux solutions est établie selon une *structure de voisinage* donnée. Deux solutions rapprochées en abscisse sont voisines, i.e. qu'une perturbation simple de la première permet d'atteindre la seconde, et vice-versa. La Figure 2.1 illustre également le principe d'optimum local et global pour un problème de minimisation. L'*optimum global*, celui que l'on cherche à atteindre, est défini comme la meilleure solution possible pour une instance de problème [BLA'98]. Un *optimum local* est une solution à un problème qui est de meilleure qualité que les solutions qui en sont voisines, mais de moindre qualité que l'optimum global [BLA'98]. Une instance de problème comporte nécessairement un ou des optima globaux et habituellement plusieurs optima locaux.

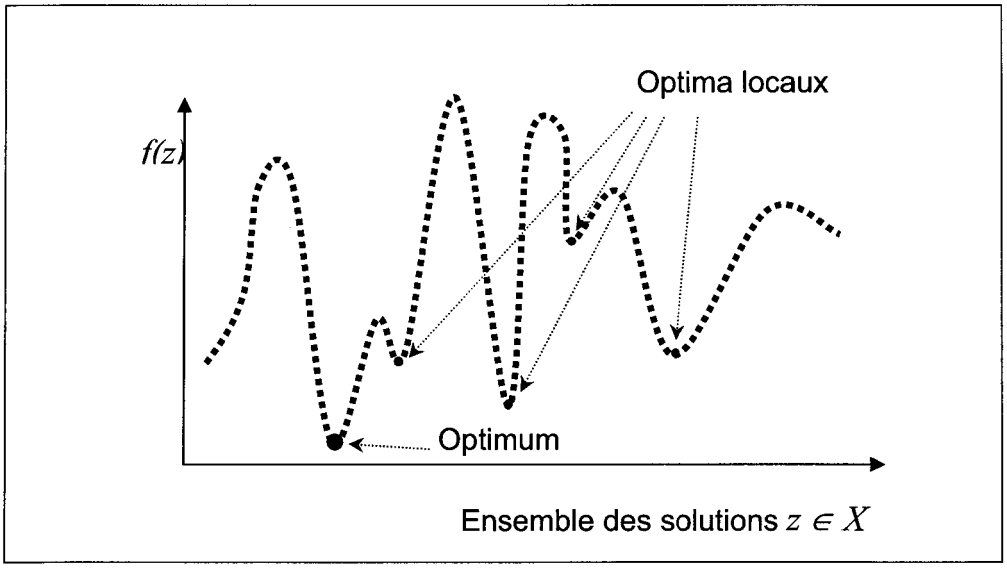

Figure 2.1 : Représentation de l'espace de solutions pour une instance d'un problème de minimisation

Les méthodes utilisées pour la résolution de problèmes d'optimisation combinatoire peuvent se classer en deux grandes catégories : les méthodes exactes et les méthodes approchées. Les premières garantissent la complétude de la recherche, alors que les secondes gagnent en efficacité, tout en perdant en complétude. La plupart du temps, la NP-complétude des problèmes rend les méthodes exactes très coûteuses en temps de calcul et parfois même inapplicables. Lorsque l'optimalité n'est pas une nécessité, les méthodes heuristiques sont une alternative intéressante.

Les heuristiques appartiennent à la catégorie des méthodes approchées. Le mot heuristique, dérivé du Grec « *heuriskein* », signifie *découvrir*. Une heuristique est « *une méthode, conçue pour un problème d'optimisation donné, qui produit une*

solution non nécessairement optimale lorsqu'on lui fournit une instance de ce problème. » [HAO'99].

Comme les heuristiques ne garantissent pas l'atteinte d'un optimum global, leur utilisation est justifiée dans certains cas [ZAN'81], soit : (1) Les données sont inexactes ou limitées ; (2) Un modèle simplifié du problème est utilisé ; (3) Aucune méthode exacte fiable n'existe ; (4) Une méthode exacte existe, mais n'est pas intéressante au niveau informatique parce que trop vorace en temps de calcul ou en espace mémoire ; (5) Elles sont utilisées pour améliorer la performance d'un optimiseur ; (6) Il est requis de résoudre le même problème fréquemment ; (7) Une solution heuristique est acceptable ; (8) On cherche une méthode simple et pouvant être comprise par les utilisateurs ; (9) Elles servent de dispositif d'apprentissage ; et/ou (10) Lorsqu'il existe d'autres limites concernant les ressources.

Il existe plusieurs classifications pour les heuristiques [BAL'81] [ZAN'89]. Parmi les catégories proposées, on retrouve, entre autres, les heuristiques de construction, d'amélioration, de programmation mathématique et les méta-heuristiques. Ces dernières sont des méthodes qui utilisent des heuristiques de première génération à l'intérieur de mécanismes de recherche plus complexes. Les méta-heuristiques sont définies de manière générale à la section suivante.

2.2.2 Définition

Les algorithmes méta-heuristiques sont inspirés de domaines très diversifiés. On y retrouve, entre autres, l'optimisation par colonie de fourmis (*Ant Colony Optimization - ACO*) [DOR'91b] [DOR'96c], les algorithmes génétiques (g*enetic algorithm*) [HOL'75], la recherche avec tabous (*tabu search)* [GLO'89] [GLO'90] et le recuit simulé (*simulated annealing*) [KIR'83].

Le terme méta-heuristique a été introduit par Glover en 1986, pour différencier la recherche avec tabous des autres heuristiques [HAO'99]. Les méta-heuristiques sont considérées comme la première famille d'algorithmes conçus pour une résolution informatique. Du Grec, le préfixe *meta-* signifie « au-delà » et indique un changement de niveau, un passage à un niveau « supérieur » pour étudier ou manipuler des informations de niveau inférieur. Une méta-heuristique est donc une stratégie principale qui guide d'autres heuristiques [LAG'02]. Plus précisément, une méta-heuristique est formellement définie comme un processus de génération itératif qui guide une heuristique sous-jacente en combinant intelligemment différents concepts pour *exploiter* et *explorer* l'espace de solutions, et où des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions près de l'optimum [OSM'96].

Il existe deux concepts importants pour les méta-heuristiques qui effectuent une recherche dans l'espace de solutions : l'*exploitation* et l'*exploration*. Les stratégies d'exploitation sont basées sur la modification des règles de choix pour encourager les caractéristiques de solutions historiquement reconnues comme

bonnes. Les mécanismes d'exploitation d'une méta-heuristique représentent ainsi sa capacité à extraire la meilleure solution dans une région donnée de l'espace de solutions. De son côté, l'exploration encourage le processus de recherche à atteindre des régions non encore visitées et à générer des solutions différentes de manière significative de celles rencontrées auparavant [LAG'02]. La qualité d'une méta-heuristique est proportionnelle à sa capacité à intensifier et à diversifier sa recherche.

Les méta-heuristiques sont aussi appelées « Algorithmes inspirés de la nature », parce qu'elles ont toujours pour base des principes observés dans des environnements naturels variés. Par exemple, l'optimisation par colonie de fourmis est inspirée des colonies de fourmis réelles qui parviennent à trouver le chemin le plus court entre la fourmilière et une réserve de nourriture. Les algorithmes génétiques reproduisent les mécanismes de la sélection naturelle de Darwin qui tendent vers une meilleure adaptation au milieu. La recherche avec tabous reprend le concept de mémoire observé chez l'Homme. Les déplacements dans l'espace de solutions sont mémorisés et cette information est utilisée pour empêcher une recherche locale de cycler. Le recuit simulé imite le processus physique de recuit du métal en amenant le système à un niveau de stabilité de ses molécules.

Notons qu'une méta-heuristique n'est pas propre à un problème précis, mais adaptable et applicable à une large classe de problèmes. Son principal avantage par rapport aux heuristiques de première génération est de posséder des mécanismes qui permettent d'éviter le *piège de l'optimum local*. Un algorithme de

recherche dans l'espace de solutions peut rester piégé dans un optimum local. Pour éviter d'y rester pris, les méta-heuristiques utilisent différents mécanismes qui permettent une certaine détérioration de la solution courante ou l'utilisation d'un ensemble de solutions.

L'optimisation par colonie de fourmis a été retenue comme méthode de résolution pour le présent travail. La section suivante y est consacrée.

2.3 L'Optimisation par Colonie de Fourmis

2.3.1 Principe général

L'OCF est une méta-heuristique dont le comportement est basé sur celui des fourmis réelles. L'algorithme est inspiré d'une expérience menée en 1989 par S. Goss *et al.* [GOS'89]. Des fourmis d'Argentine avaient été mises en contact avec une source de nourriture reliée à la fourmilière par un pont ayant deux branches de longueurs différentes, tel qu'illustré à la Figure 2.2. Les fourmis, insectes minuscules, presque aveugles et limités en intelligence, parvenaient après quelque temps à utiliser toujours le chemin le plus court entre la fourmilière et la nourriture. Il a été observé que la probabilité que toutes les fourmis viennent à emprunter le chemin le plus court augmentait avec la différence de longueur entre les deux branches du pont.

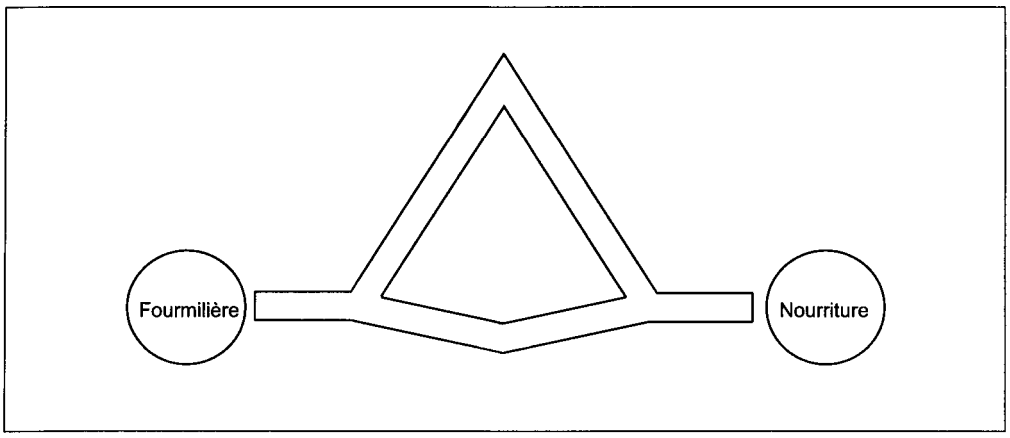

Figure 2.2 : Disposition de l'expérience de Goss *et al.* [DOR'99a]

Goss *et al.* [GOS'89] ont établi que les fourmis partagent des informations sur leurs expériences d'exploration à l'aide d'une substance chimique appelée

phéromone. Elles déposent par terre une certaine quantité de cette substance lorsqu'elles se déplacent et en même temps, scrutent le sol pour « lire » les informations laissées par leurs prédécesseurs. Lorsqu'elles ont à faire un choix quant à la direction à prendre, les fourmis sélectionnent un chemin de façon probabiliste en fonction de la quantité de phéromone présente sur les différentes voies possibles. Le comportement observé par Goss s'explique par le fait que les fourmis qui empruntent le chemin le plus court parviennent à la source de nourriture et reviennent à la fourmilière plus vite que celles qui ont pris le plus long chemin. En laissant toujours une trace de phéromone, elles rendent le chemin emprunté plus attirant pour les fourmis suivantes. De plus, comme la phéromone s'évapore avec le temps, le chemin le plus long est de moins en moins emprunté et sa trace disparaît presque complètement. Le temps nécessaire à la stabilisation du système est imputable au fait qu'aucune phéromone n'est présente sur aucune route au départ, les fourmis ont alors autant de chances de choisir le plus court que le plus long chemin.

Basé sur ces observations, l'OCF est un algorithme à l'intérieur duquel des agents indépendants, les fourmis artificielles, construisent des solutions de manière probabiliste en se basant sur une mémoire collective concernant les expériences antérieures : la trace de phéromone.

2.3.2 Le problème du voyageur de commerce

Le premier problème avoir été résolu à l'aide d'un algorithme de fourmi est le problème du voyageur de commerce (*Traveling Salesman Problem - TSP)* [ROB'49]. Cette section comprend une définition formelle du TSP qui sera utilisée dans les sections suivantes pour expliquer le fonctionnement des algorithmes de fourmis.

Le TSP est le problème d'un vendeur sur la route devant s'arrêter dans un ensemble fini de villes, en empruntant le chemin le plus court possible. Il doit visiter exactement une fois chaque ville et revenir au point de départ à la fin de la tournée. Une instance de TSP est donnée par un graphe pondéré (H, E), où H est l'ensemble des villes à visiter et E celui des liens entre ces dernières pour lesquels les longueurs sont connues. Les distances entre toutes les paires de villes (r, s) sont mémorisées dans une matrice de distances D. La taille d'une instance de TSP est $h = |H|$, le nombre de villes à parcourir.

La résolution d'une instance consiste à trouver un cycle hamiltonien de longueur minimale du graphe, où un cycle hamiltonien est un chemin fermé visitant exactement une fois chacun des nœuds.

Une instance de TSP peut être *symétrique* ou *asymétrique*. Dans le premier cas, $D_{rs} = D_{sr}$ et la matrice de distances est symétrique par rapport à la diagonale (qui n'est pas utilisée). Dans le cas d'une instance asymétrique, on a $D_{rs} \neq D_{sr}$. La

distance à parcourir pour passer de la ville r à la ville s peut donc ne pas être la même que celle pour passer de s à r.

2.3.3 Version préliminaire de l'OCF : L'Ant System

L'Ant System (AS) a été introduit par Dorigo, Colorni et Maniezzo en 1991 [DOR'91b]. Il est le premier algorithme de fourmis à apparaître et le prototype de plusieurs autres méthodes [MAN'99b]. L'AS a été créé au départ pour la résolution du problème du voyageur de commerce (symétrique et asymétrique) décrit à la section précédente.

Le fonctionnement général de l'AS consiste à construire des chemins pour un ensemble de fourmis, de manière probabiliste en se basant sur la mémoire collective (la trace de phéromone) et sur une vision locale du problème, et ce, pour un nombre de cycles donné. Un cycle, aussi appelé *itération*, est complété chaque fois que toutes les fourmis ont terminé la construction d'une solution.

Le principe de la trace de phéromone est très important pour l'AS. Les auteurs le décrivent comme un processus auto-catalytique [DOR'91a]. Un tel processus en est un qui, par définition, se renforce lui-même d'une manière qui conduit à une convergence rapide et, si aucune limitation n'est prévue, aboutit à l'explosion. Les fourmis de l'AS utilisent une matrice de phéromone pour communiquer. Cette matrice doit représenter la profitabilité associée aux différents éléments de solution. La représentativité des informations de la trace s'accroît

avec le temps puisque la matrice est mise à jour par les fourmis au cours de l'algorithme.

Les fourmis sont des agents simples et possèdent les caractéristiques suivantes : (1) Elles choisissent la prochaine ville où aller avec une probabilité qui est fonction de la distance entre les villes et de la quantité de phéromone présente sur l'arête les reliant ; (2) Pour les amener à construire des chemins légaux (i.e. qui passent exactement une fois par chaque ville), la transition vers les villes déjà visitées est interdite jusqu'à ce qu'un chemin soit complété ; et (3) Lorsqu'elles ont complété un chemin, les fourmis déposent une certaine quantité de phéromone sur chaque arête (r, s) empruntée pour leur solution.

Le Tableau 2.1 résume les termes de la notation utilisée pour l'algorithme de l'AS. La partie supérieure contient les variables utilisées et la partie inférieure les paramètres dont les valeurs sont à fixer. Ces termes seront expliqués plus en détails dans les paragraphes qui suivent.

Variable	Description
τ_{rs}	Quantité de phéromone présente entre r et s
$\Delta\tau_{rs}$	Quantité de phéromone à ajouter à τ_{rs}
η_{rs}	Visibilité locale entre r et s
P_{rs}^{k}	Probabilité pour la fourmi k de choisir l'élément s, connaissant r
L_k	Évaluation de la solution de la fourmi k
$tabou_k$	Les éléments déjà placés pour la fourmi k
y	Index de la liste tabou
t	Temps actuel (compteur de cycles)
Paramètre	**Description**
m	Nombre de fourmis
NC_{MAX}	Nombre maximal de cycles
τ_0	Quantité de phéromone initiale
α	Exposant de la trace de phéromone
β	Exposant de la visibilité locale
ρ	Persistance de la trace

Tableau 2.1 : Notation de l'AS

La Figure 2.3 présente le pseudo-code de l'Ant System. L'étape 1 initialise le compteur de cycles et la trace de phéromone. Un cycle commence à l'étape 2, où l'indice y des listes tabou est remis à 1 et chaque fourmi place sa ville de départ dans sa liste tabou. À l'étape 3, les fourmis construisent des chemins en parallèle qui sont évalués à l'étape 4, où la meilleure solution trouvée jusqu'à maintenant est mise à jour. À l'étape 5, il y a mise à jour globale de la trace. À l'étape 6, le compteur de cycles est incrémenté et on décide de poursuivre ou non l'exécution,

selon que le nombre de cycles NC_{MAX} est atteint ou qu'une autre condition d'arrêt préalablement définie est atteinte.

1. $t = 0$
   Initialiser chaque arête (r, s) à la valeur de τ_0

2. $y = 1$
   POUR chaque fourmi k = 1 à m *{Une fourmi sur chaque ville}*
   Placer la fourmi k sur la ville k et affecter $tabou_k(y) = k$

3. **POUR** y = 2 à h
   POUR chaque fourmi k = 1 à m
   Choisir une ville s selon P_{rs}^k, calculé selon l'Équation 2.4
   $tabou_k(y) = s$

4. **POUR** chaque fourmi k = 1 à m
   Calculer la longueur L_k du chemin décrit par k
   Mettre à jour la meilleure solution globale

5. **POUR** chaque arête (r, s)
   Mettre à jour τ_{rs} selon l'Équation 2.1

6. $t = t + 1$
   SI ($t < NC_{MAX}$) **ALORS**
   Vider toutes les listes tabou
   Aller à l'étape 2
   SINON
   ARRÊTER

Figure 2.3 : Pseudo-code de l'Ant System [DOR'96c]

Comme les fourmis déposent de la phéromone entre chaque paire de villes, la trace de phéromone est notée dans une matrice τ de nombres réels ayant la même structure que la matrice d'adjacence du graphe de l'instance. La valeur de

$\tau_{rs}(t)$ indique la quantité de phéromone présente sur l'arc *(r, s)* au temps *t*, la profitabilité d'emprunter le lien entre les villes r et s par le passé.

Au départ, tous les éléments de la matrice de trace sont initialisés à une petite valeur constante positive τ_0. L'accumulation de trace a pour but d'encourager les fourmis à retourner sur les arêtes que les expériences antérieures ont montrées profitables. Elle agit dans ce cas comme élément d'intensification de la recherche.

La mise à jour de la trace se fait de manière globale, à la fin de chaque cycle et selon l'Équation 2.1, où ρ est un paramètre.

$$\tau_{rs}(t+1) = \rho \cdot \tau_{rs}(t) + \Delta\tau_{rs} \tag{2.1}$$

On calcule d'abord le $\Delta\tau_{rs}$ pour chaque arête *(r, s)*, selon l'Équation 2.2, où $\Delta\tau_{rs}^{k}$ est la quantité de phéromone laissée par une fourmi k pour l'arc *(r, s)*. Cette quantité se calcule selon l'Équation 2.3, où A est une constante et L_k est la longueur du chemin de la fourmi k. Ainsi, la quantité de phéromone laissée par une fourmi sur une arête est fonction de la qualité de la solution finale qu'elle a produite. Toutes les fourmis participent à la mise à jour de la trace et elles n'ajoutent aucune phéromone sur les arêtes qu'elles n'ont pas empruntées pour construire leur solution.

$$\Delta\tau_{rs} = \sum_{k=1}^{m} \Delta\tau_{rs}^{k} \tag{2.2}$$

$$\Delta\tau_{rs}^{k} = \begin{cases} \dfrac{\Lambda}{L_k} & si\ la\ fourmi\ k\ emprunte\ l'arc\ (r,s) \\ 0 & sinon \end{cases} \tag{2.3}$$

Le coefficient ρ est appelé *persistance de la trace* et sa valeur est comprise entre 0 et 1 inclusivement. À chaque cycle, une partie de la phéromone sur toutes les arêtes s'évapore et seules celles empruntées en reçoivent en fonction de la qualité des solutions dont elles font partie. Le mécanisme d'évaporation a pour but d'éviter les accumulations de trace à l'infini et reproduit le phénomène d'évaporation de la phéromone naturelle. Ainsi, les arêtes dont la trace n'est pas renouvelée deviennent moins attirantes pour les fourmis.

Une liste tabou $tabou_k$ est associée à chaque fourmi k afin d'assurer les chemins hamiltoniens. La fourmi y ajoute chaque ville qu'elle visite et ne peut choisir de s'y déplacer à nouveau. Lorsqu'un chemin est complété, $tabou_k$ contient, dans l'ordre, les villes parcourues par la fourmi k et $tabou_k(y)$ est la $y^{ième}$ ville visitée par cette fourmi. Les listes tabou sont vidées après chaque cycle, une fois la mise à jour de la trace effectuée.

Lorsqu'une fourmi k doit faire le choix de la prochaine ville à visiter, elle calcule la probabilité de transition vers chaque ville potentielle $s \notin tabou_k$, notée

P_{rs}^k, où r est la dernière ville visitée par la fourmi k. Cette probabilité est fonction de τ_{rs} et d'une visibilité locale, η_{rs}, qui doit refléter l'attrait d'un élément de manière locale. Pour la fourmi adaptée au TSP, la valeur de η_{rs} est donnée par $1/D_{rs}$, ce qui donne à la ville s un attrait inversement proportionnel à la distance entre r et s. On calcule P_{rs}^k selon l'Équation 2.4, appelée *règle de transition*.

$$P_{rs}^k = \begin{cases} \dfrac{[\tau_{rs}]^\alpha \cdot [\eta_{rs}]^\beta}{\sum_{w \notin tabou_k} [\tau_{rw}]^\alpha \cdot [\eta_{rw}]^\beta} & si\ s \notin tabou_k \\ 0 & sinon \end{cases} \qquad (2.4)$$

Cette règle de transition est appelée règle *proportionnelle-aléatoire* (*random-proportional rule*). Notons que $\sum P_{rs} = 1$ et que les exposants α et β servent à balancer l'importance relative de la trace de phéromone versus la visibilité locale. Si $\alpha = 0$ ou $\beta = 0$, on obtient respectivement une heuristique de construction gourmande et un algorithme proche du *Q-Learning* [WAT'92], où seul l'apprentissage est pris en compte.

Il est intéressant d'observer les valeurs que prennent habituellement les paramètres de l'AS. Dorigo *et al.* [DOR'91b] [DOR'96c] ont effectué des tests pour connaître les meilleures valeurs de paramètres pour l'AS. On a trouvé que le

nombre de fourmis m doit être dans $O(h)$, où h est la taille du problème. La persistance de la trace ρ a donné les meilleurs résultats avec une valeur de 0,5. En effet, l'algorithme a besoin de pouvoir oublier une partie de l'expérience acquise pour mieux exploiter une nouvelle information globale qui lui arrive. Finalement, les valeurs données à α et β ont un impact important sur la qualité des solutions obtenues. Pour de grandes valeurs de α combinées à des valeurs de β qui ne sont pas suffisamment grandes, l'algorithme entre rapidement dans un état de *chemin-unique*, où les fourmis empruntent toujours les mêmes arêtes sans trouver de bonne solution. Si on ne donne pas une valeur suffisamment grande à α ou trop grande à β, alors l'algorithme ne peut produire que des solutions de qualité moyenne. Les auteurs concluent que α doit se tenir autour de 1, et β entre 2 et 5 pour le problème du TSP testé.

La complexité temporelle de l'AS est dans $O\left(NC_{MAX} \cdot h^2 \cdot m\right)$, mais comme on a observé une relation linéaire entre le nombre de villes h et le meilleur nombre de fourmis m, elle peut être considérée dans $O\left(NC_{MAX} \cdot h^3\right)$ [DOR'96a].

En conclusion, l'idée générale sous-jacente à l'AS est celle d'un processus auto-catalytique guidé par une force gourmande [COL'92b]. Le processus auto-catalytique utilisé seul tend à converger vers un chemin sous-optimal avec une vitesse exponentielle. Lorsqu'ils travaillent ensemble, il apparaît que la force

gourmande donne au processus auto-catalytique les suggestions qui lui permettent de converger rapidement vers de très bonnes solutions.

2.3.4 L'algorithme d'Optimisation par Colonie de Fourmis

L'OCF proprement dite a été introduite par Dorigo et Gambardella [DOR'97b], toujours pour une application au TSP. Quatre éléments majeurs sont modifiés ou ajoutés à l'AS pour créer l'OCF.

Premièrement, la règle de transition présentée à l'Équation 2.4 est modifiée pour offrir un meilleur balancement entre l'exploration de nouvelles zones de l'espace de solutions et l'exploitation des connaissances accumulées sur le problème. Il en résulte l'Équation 2.5, avec s étant la ville choisie par la fourmi k et r étant la dernière ville visitée, où Q_0 est un paramètre $(0 \leq Q_0 \leq 1)$, Q est un nombre généré aléatoirement, distribué uniformément entre $[0,1]$ et ψ est une ville choisie aléatoirement en fonction des probabilités calculées selon l'Équation 2.4. On procède ainsi à une intensification de la recherche avec une probabilité de Q_0, puisque l'on choisit alors l'argument maximal des $\left\{[\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}\right\} \forall w \notin tabou_k$, qui correspondent à l'attrait des différentes villes encore à visiter pour k. Avec une probabilité de $(1-Q_0)$, on optera pour la diversification en choisissant s selon la règle de transition de l'AS, ce qui ajoute à

la recherche un caractère probabiliste. Cette règle est appelée *pseudo-proportionnelle-aléatoire* (*pseudo-random-proportionnal rule*).

$$s = \begin{cases} argmax_{w \notin tabou_k} \left\{ [\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta} \right\} & si\ Q \leq Q_0 \\ \psi & sinon \end{cases} \quad (2.5)$$

Deuxièmement, afin de profiter d'une information locale supplémentaire, Dorigo *et al.* [DOR'99b] ont introduit une liste de candidats. L'ensemble des choix possibles à partir d'une ville est restreint aux lc villes les plus proches non encore sélectionnées. Cette limitation des candidats a également l'avantage de réduire le temps nécessaire au calcul des probabilités.

Troisièmement, la mise à jour de la trace de phéromone est modifiée dans le but de mieux diriger la recherche. Contrairement à l'AS, où toutes les fourmis participent à la mise à jour, une seule fourmi met à jour la trace de façon globale à la fin de chaque cycle et en fonction de la qualité de sa solution. Les auteurs ont expérimenté une mise à jour par la meilleure fourmi globale gb (*global-best*), celle ayant produit la meilleure solution trouvée depuis le début de l'algorithme, et par la meilleure du cycle (*iteration-best*). Les deux stratégies donnent des résultats très semblables avec une légère préférence pour la meilleure globale. La mise à jour *globale* est effectuée à la fin de chaque cycle, selon l'Équation 2.6, où $0 < \rho_{global} < 1$ est la persistance globale de la trace et L_{gb} est la longueur du chemin de la meilleure fourmi globale.

$$\tau_{rs}(t+1) = \rho_{global} \cdot \tau_{rs}(t) + (1-\rho_{global}) \cdot \Delta\tau_{rs}$$

$$où\ \Delta\tau_{rs} = \begin{cases} \dfrac{1}{L_{gb}} & si\ gb\ emprunte\ (r,s) \\ 0 & sinon \end{cases} \qquad (2.6)$$

Quatrièmement, on introduit une seconde mise à jour de la trace, la mise à jour *locale*, pour contribuer à la diversification de la recherche. Chaque fourmi, lorsqu'elle ajoute une ville à sa solution, diminue la quantité de phéromone sur l'arête *(r, s)* qu'elle vient d'emprunter, selon l'Équation 2.7, où $0 < \rho_{local} < 1$ est la persistance locale de la trace. Trois valeurs possibles ont été considérées pour $\Delta\tau$ [DOR'97b]. L'emploi de $\Delta\tau = 0$ donne les pires résultats. Les auteurs ont également essayé $\Delta\tau = 0$ et $\Delta\tau = \gamma \cdot \max_{w \notin tabou_k} \tau_{rw}$, où $0 \leq \gamma < 1$ est un paramètre. Cette dernière équation est inspirée du *Q-Learning*. Les OCF avec $\Delta\tau = \tau_0$ et avec $\Delta\tau = \gamma \cdot \max_{w \notin tabou_k} \tau_{rw}$ offrent des performances semblables. Comme le calcul de $\Delta\tau = 0$ requiert moins de temps, cette valeur a été retenue.

Erreur ! Des objets ne peuvent pas être créés à partir des codes de champs de mise en forme. (2.7)

Par la mise à jour locale, la quantité de phéromone sur les arêtes déjà empruntées diminue légèrement et ainsi, ces arêtes deviennent moins attrayantes

au cours du cycle. Ceci permet une meilleure diversité des solutions en encourageant les fourmis à choisir des arêtes moins empruntées. La mise à jour locale de la trace a été prouvée efficace avec un ensemble d'instances de TSP [DOR'97b].

Pour résumer les modifications apportées à l'AS, la Figure 2.4 montre l'algorithme détaillé de l'OCF. On y observe quatre nouveaux éléments. En (a), le choix de la prochaine ville est fait selon la règle pseudo-proportionnelle-aléatoire. Dans la même étape, en (b), on observe l'utilisation d'une liste de candidats. La mise à jour locale se trouve en (c) et en (d) la mise à jour globale est effectuée par la meilleure solution globale. Le lecteur peut se référer au Tableau 2.1 pour la notation de l'OCF. La seule différence est la duplication de la persistance de la trace ρ en ρ_{local} et ρ_{global} pour les mises à jour locale et globale de la trace.

1. $t = 0$
   Initialiser chaque arête (r, s) à la valeur de τ_0

2. $y = 1$
   POUR chaque fourmi $k = 1$ à m *{Une fourmi sur chaque ville}*
   Placer la fourmi k sur la ville k et affecter $tabou_k(y) = k$

3. **POUR** $y = 2$ à h
   POUR $k = 1$ à m
   Choisir une ville s parmi les lc villes candidates (a)(b)
   selon l'Équation 2.5
   Mettre à jour τ_{rs} selon l'Équation 2.7 (c)
   $tabou_k(y) = s$

4. **POUR** chaque fourmi $k = 1$ à m
   Calculer la longueur L_k du chemin décrit par k
   Mettre à jour la meilleure solution globale

5. **POUR** chaque arête $(r, s) \in tabou_{gb}$ *{Mise à jour globale}*
   Mettre à jour τ_{rs} selon l'Équation 2.6 (d)

6. $t = t + 1$
   SI $(t < NC_{MAX})$ **ALORS**
   Vider toutes les listes tabou
   Aller à l'étape 2
   SINON
   ARRÊTER

Figure 2.4 : Pseudo-code de l'OCF [DOR'97b]

Les valeurs habituellement données aux paramètres de l'OCF ne sont pas les mêmes que pour l'AS. Dorigo et Gambardella [DOR'97b] estiment qu'à l'exception de τ_0, les valeurs données aux paramètres sont largement indépendantes de la taille du problème. Le nombre de fourmis m doit désormais être fixé

expérimentalement [DOR'99a]. Dorigo et Gambardella [DOR'97b] ont proposé une formule pour calculer le nombre idéal de fourmis. Les auteurs concluent que m devrait être près de 10. Alors que ρ devait être près de 0,5 pour l'AS, il en va autrement pour l'OCF. Plus les valeurs de ρ_{local} et ρ_{global} sont près de 1 sans l'égaler, meilleurs sont les résultats, et souvent $\rho_{local} = \rho_{global}$. Les valeurs de α et β déterminées pour l'AS sont conservées pour l'OCF. La valeur de Q_0 a été fixée à 0,9, ce qui favorise l'exploitation. La quantité de trace initiale τ_0 est affectée, après expérimentations, à une valeur égale à $(h \cdot L_{oe})^{-1}$, où h est la taille du problème et L_{oe} est la longueur estimée du chemin optimal. Cette estimation est donnée par l'heuristique du plus proche voisin (*nearest neighbor heuristic*) dans le cas de Dorigo et Gambardella [DOR'97b].

Finalement, l'ajout d'une méthode de recherche locale est abordé par Dorigo et Gambardella [DOR'97b]. Les auteurs concluent que leur méthode est comparable aux autres méta-heuristiques, et veulent par cet ajout la rendre compétitive par rapport aux méthodes élaborées spécifiquement pour le TSP. Afin de conserver l'habileté de l'algorithme à résoudre les problèmes de TSP symétriques et asymétriques, la structure de voisinage appelée *restricted 3-opt* est appliquée. Les résultats obtenus permettent aux auteurs de conclure que l'OCF avec recherche locale donne d'excellents résultats pour les problèmes asymétriques. L'algorithme a une très bonne performance pour les instances

symétriques, mais insuffisante à surpasser le meilleur algorithme connu au moment des tests (le STSP-GA [FRI'96]). Les auteurs concluent qu'il serait intéressant d'utiliser une structure de voisinage encore plus efficace, telle l'algorithme de Lin-Kernighan.

2.3.5 Modifications des algorithmes originaux

Depuis leur introduction en 1991, les algorithmes de fourmis ont subi de nombreuses modifications dans le but d'en améliorer la performance ou de les adapter à des problèmes particuliers. Cette section aborde les principales modifications apportées au cours des années.

L'hybridation des fourmis avec la recherche locale se montre souvent efficace. Plusieurs auteurs ont greffé leur colonie d'une procédure d'amélioration locale exécutée avant la mise à jour de la trace. Mais l'hybridation la plus complète est probablement celle de Gambardella, Taillard et Dorigo, le HAS-QAP, pour « *Hybrid Ant System for the Quadratic Assignment Problem* » [GAM'99]. L'idée consiste à donner une solution de départ à chaque fourmi et laisser celle-ci se déplacer dans l'espace de solutions à l'aide d'une structure de voisinage basée sur la trace de phéromone accumulée. La mise à jour est uniquement globale et un mécanisme effaçant périodiquement toutes les traces de phéromone est utilisé pour éviter une convergence trop rapide.

Maniezzo [MAN'99a] propose une modification de la règle de transition visant à réduire le temps de calcul et à éliminer un paramètre. La règle donnée à

l'Équation 2.4 est modifiée pour obtenir l'Équation 2.8. Cette règle de transition est équivalente à la précédente : elle permet au décideur de donner une importance relative à la trace par rapport à la visibilité locale en plus d'éliminer un paramètre et d'utiliser des opérateurs moins exigeants en termes de temps.

$$P_{rs}^{k} = \begin{cases} \dfrac{\alpha \cdot \tau_{rs} + (1-\alpha) \cdot \eta_{rs}}{\sum_{w \notin tabou_k} \alpha \cdot \tau_{rw} + (1-\alpha) \cdot \eta_{rw}} & si\ s \notin tabou_k \\ 0 & sinon \end{cases} \tag{2.8}$$

Gagné, Gravel et Price [GAG'01] ont adapté l'OCF à un problème d'ordonnancement d'une machine unique (*Single Machine Scheduling*) à objectifs multiples. Dans un premier temps, les auteurs modifient la règle de transition en utilisant une matrice de distances pour chaque objectif et donc une visibilité locale pour chacun. De plus, l'algorithme est bonifié d'une fonction d'anticipation (*look-ahead function*). Cette fonction reflète la qualité potentielle de la solution en cours de construction et est insérée à la suite des visibilités locales dans la règle de transition. Son calcul prend en considération la partie construite, l'élément à ajouter et une borne inférieure de l'évaluation de la partie non complétée. Finalement, une procédure d'amélioration locale est ajoutée et appliquée aux meilleures solutions trouvées. Après avoir conclu en évaluant le potentiel de ces améliorations [GAG'02a], les auteurs utilisent l'algorithme pour la résolution de problèmes de nature industrielle sur lesquels ils avaient travaillé par le passé

[GRA'02]. Ils concluent que les matrices de visibilité multiples sont efficaces et que la recherche locale aide à l'amélioration des résultats mais engendre une augmentation importante du temps de calcul. Selon les auteurs, les simplifications nécessaires à l'application de la fonction d'anticipation semblent limiter son utilité pour la situation industrielle.

L'importance des temps de calcul nécessaires à une bonne exploration de l'espace de solutions par les méta-heuristiques a amené la communauté à se tourner vers les algorithmes parallèles [CUN'01]. Plusieurs auteurs ont travaillé avec succès sur la parallélisation d'algorithmes de fourmis. Notons entre autres Bullhheimer *et al.* [BUL'98] et Talbi *et al.* [TAL'01].

Nombre d'auteurs proposent des modifications relatives à la gestion de la trace de phéromone. Comme l'utilisation de cette trace est un élément essentiel de l'OCF, il est important d'en connaître les variantes présentes dans la littérature.

Stützle et Hoos [STU'97a] introduisent le MAX-MIN Ant System dans lequel la force de la trace de phéromone est restreinte à un intervalle donné. Des bornes inférieure et supérieure pour la trace sont établies en fonction de la taille de l'instance à résoudre. Le but de cette limitation est d'éviter la convergence et la stagnation trop rapide observée lorsque la mise à jour est faite uniquement par la meilleure fourmi du cycle. Cette méthode est considérée par ses auteurs comme meilleure que l'AS et au moins aussi performante que l'OCF. Plus tard, l'algorithme a été modifié avec succès pour utiliser la règle de transition pseudo-

proportionnelle-aléatoire de l'OCF et une phase de recherche locale a été introduite pour chaque fourmi.

Taillard et Gambardella [TAI'97b] proposent le *Fast Ant System* (FANT). Il s'agit d'un OCF avec recherche locale, possédant trois particularités : (1) Une seule fourmi est utilisée, ce qui peut être considéré simplement comme une valeur spécifique de paramètre ; (2) La trace de phéromone est traitée uniquement avec des nombres entiers et modifiée en fonction des améliorations de la meilleure solution globale ; et (3) Il n'y a pas d'évaporation de la phéromone à chaque cycle, mais plutôt une réinitialisation occasionnelle pour éviter la stagnation. Une comparaison entre le FANT et d'autres méta-heuristiques basées sur une mémoire adaptative montre que le FANT a le meilleur ratio qualité/temps. Il n'atteint pas les meilleurs résultats, mais il s'exécute considérablement plus vite que les autres algorithmes testés. Suite aux travaux sur le FANT, Stützle et Linke [STU'02] ont conçu un algorithme issu d'une hybridation entre l'HAS-QAP et le FANT, toujours pour le QAP. En tirant partie des forces de chaque algorithme, les auteurs obtiennent des résultats compétitifs par rapport aux meilleurs algorithmes courants.

Bullnheimer, Hartl et Straub [BUL'99a] proposent une variante appelée *Rank-based Ant System*, dont le but est d'éviter la surcharge de phéromone sur les arêtes des chemins sous-optimaux qui entraîne la convergence de l'algorithme. À la fin de chaque cycle, avant la mise à jour, les fourmis sont classées en fonction de la qualité de leur solution. La mise à jour de la trace se fait par les meilleures fourmis et en fonction non pas de leur qualité, mais de leur rang dans ce

classement. Le Rank-based Ant System se montre compétitif lorsque comparé à un algorithme génétique et à un recuit simulé sur des instances de TSP. De plus, sur les instances de plus grande taille, il les surpasse en termes de moyenne et surtout de pire cas.

Merkle et Middendorf [MER'00] présentent une nouvelle manière d'utiliser la trace de phéromone pour l'adapter au problème de retard total pondéré sur machine unique (*single machine total weighted tardiness problem*). Le problème consiste à ordonner des commandes en minimisant le retard total généré, où le retard total est une somme pondérée des unités de temps de retard de chaque commande par rapport à sa date de livraison prévue. Les auteurs utilisent une trace qui associe une position de la séquence à une commande et les mises à jour sont faites de manière standard. Cependant, lorsque la trace doit être considérée pour la règle de transition, on utilise la somme des traces associant la commande candidate à chacune des positions inférieures à la courante. On cherche ainsi à éviter qu'une commande, une fois passée sa date de livraison prévue, soit repoussée au bout de la séquence parce qu'aucune bonne solution n'a été trouvée dans laquelle celle-ci était placée après cette même date. Cette nouvelle règle de transition se montre très efficace lorsque comparée à une version standard de l'OCF. Elle offre une meilleure performance sur les 125 instances tests utilisées et même sur une version non-pondérée du problème.

Guntsch et Middendorf [GUN'02a] apportent une toute nouvelle notion à l'OCF : l'utilisation d'une population, dans un algorithme appelé le *FIFO-Queue*

ACO (aussi appelé *P-ACO* pour *Population based ACO*). Il s'agit d'un OCF où les meilleures solutions sont conservées dans une population finie et où il n'y a pas d'évaporation de la trace. À chaque cycle, la meilleure solution du cycle est insérée dans la population et la matrice de phéromone est mise à jour, comme dans l'OCF standard. Lorsque la population atteint une taille donnée, la plus ancienne solution est retirée de la population et les éléments de la matrice de trace y correspondant sont diminués de l'exacte quantité qu'avait ajouté cette solution lors de son introduction dans la population. D'autres heuristiques de choix de la solution à retirer ont aussi été considérées [GUN'02b]. Les quantités de phéromone ajoutée/retirée sont calculées en fonction de la qualité de la solution introduite par rapport à la celle de la meilleure solution globale, d'une borne supérieure, de la valeur initiale de la trace de phéromone et de la taille de la population. La principale force du FIFO-Queue ACO est sa faculté à s'adapter aux problèmes dynamiques. Des tests conduits sur des instances dynamiques de TSP et de QAP ont montré une efficacité surprenante et une facilité de réparation des solutions et de la matrice de trace. Le FIFO-Queue ACO est repris plus tard par Scheuermann *et al.* [SCH'04] pour une application sur un problème de FPGA (*Field-Programmable Gate Arrays*) pour lequel il permet une nette accélération du calcul, avec des résultats très intéressants.

En conclusion, beaucoup d'auteurs ont présenté des algorithmes dérivés de l'OCF original. La majeure partie des modifications apportées concerne évidemment la trace de phéromone, surtout la façon de la mettre à jour et les

valeurs que peuvent prendre ses éléments. Peu ou pas de chercheurs se sont arrêtés à modifier la structure même de la trace de phéromone, tous utilisent une matrice à deux dimensions et en adaptent le contenu au problème traité.

2.3.6 Applications d'algorithmes de fourmis à des problèmes d'optimisation

Cette section aborde quelques problèmes d'optimisation fréquemment traités à l'aide des algorithmes de fourmis. Les travaux cités ici représentent une liste non exhaustive des efforts de la communauté scientifique concernant chacun des problèmes. Les publications les plus pertinentes ont été retenues.

Le problème du voyageur de commerce (symétrique ou asymétrique), présenté à la Section 2.3.2, a été le premier problème traité par les algorithmes de colonies de fourmis et probablement celui qui l'est le plus souvent. Il a été utilisé tout au long de l'évolution des algorithmes de fourmis, de l'AS à l'OCF. Les conclusions sur la méthode publiées par Dorigo, Maniezzo et Colorni [DOR'91b] [DOR'91a] étant très encourageantes, ces derniers ont poussé l'expérience. Dans le cadre de nouveaux tests [COL'92b], ils établissent l'importance de l'exposant β et concluent que l'efficacité générale de l'algorithme diminue lorsque le nombre de fourmis est trop grand. Une analyse plus approfondie de l'algorithme et de sa complexité est présentée plus tard [COL'92a]. En 1995, Gambardella et Dorigo [GAM'95] ont jeté les bases de ce qui deviendra l'OCF en introduisant une forme de mise à jour locale de la trace et la règle de transition pseudo-proportionnelle-aléatoire. Cet algorithme intermédiaire est appelé *Ant-Q* et est issu d'un

croisement entre l'AS et le *Q-Learning*, un algorithme d'*apprentissage par renforcement* (*reinforcement learning*). Peu avant l'introduction de l'OCF telle que nous la connaissons, Gambardella et Dorigo [GAM'96] ont présenté une version améliorée de l'Ant-Q, l'*Ant Colony System* (ACS), qui présente une nouvelle mise à jour locale de la trace. Les auteurs concluent à l'efficacité de cette mise à jour pour le TSP et à l'efficacité de la liste de candidats pour une taille $lc = [10,20]$. Les expérimentations sur l'Ant-Q [DOR'96b] sont poursuivies en utilisant pour les tests des instances de TSP bien connues de la littérature. Celles-ci permettent de statuer notamment sur l'importance de la synergie résultant de la coopération entre les fourmis et sur la nécessité de l'exploration dans la règle de transition. Par la suite, les mêmes auteurs ont présenté le premier OCF officiel, qui a donné des résultats intéressants pour le TSP [DOR'97b]. D'autres auteurs ont abordé le TSP à l'aide de l'AS ou de l'OCF. Stützle et Hoos [STU'97a] [STU'97b] [STU'00], et plus tard Stützle et Dorigo [STU'99], ont appliqué le MAX-MIN Ant System au TSP et ont également obtenu des résultats encourageants. Dorigo et Gambardella [DOR'97a] observent que les performances de l'OCF pour le TSP symétrique ne peuvent atteindre celles des méthodes spécifiques élaborées pour le problème. Ils concluent en la pertinence d'adapter l'algorithme à une classe de problèmes plus large.

Le *problème d'affectation quadratique* (*Quadratic Assignment Problem - QAP*) est également souvent résolu à l'aide d'algorithmes de fourmis. Un QAP consiste en l'affectation d'un nombre donné d'activités à autant d'endroits, où les

mots *activité* et *endroit* doivent être pris dans leur sens général. Le but de la résolution est de produire une permutation qui minimise le coût total d'affectation, qui est calculé selon les distances entre les endroits et le coût de transport entre les activités. Maniezzo et Colorni [MAN'99c] présentent un algorithme d'OCF performant dans lequel une méthode d'amélioration locale est exécutée sur chaque solution, la trace est accumulée pour chaque affectation activité-endroit et la visibilité locale associant une activité à un endroit est représentée par une borne inférieure du coût d'une solution contenant cette affectation. Taillard et Gambardella [TAI'97a] ont appliqué l'OCF au QAP d'une manière différente, à l'aide de l'HAS-QAP (survolé à la section précédente). La trace de phéromone est la même et une procédure d'amélioration locale est également ajoutée. Cependant, aucune visibilité locale n'est utilisée, le nombre de fourmis est fixé à 1 et la mise à jour de la trace est très différente de celle de l'OCF présenté à la Section 2.3.4. L'approche ne comprend qu'une mise à jour globale, mais celle-ci est double et toujours faite par incrément entier. Les arcs empruntés par la meilleure fourmi globale reçoivent une quantité de phéromone fixe et ceux empruntés par la fourmi du cycle courant reçoivent une autre quantité de phéromone. Cette dernière quantité est utilisée pour réinitialiser la matrice de trace au départ et à chaque fois que l'algorithme stagne. Elle est alors incrémentée d'une unité, avant la mise à jour et la réinitialisation de la trace. Cette approche se révèle compétitive pour les problèmes structurés, abaissant même les meilleures solutions connues pour deux des instances de plus grande taille de QAPLIB

[BUR'97], mais offre une performance moyenne sur les instances uniformes et aléatoires. Maniezzo [MAN'99a] propose une version de l'OCF pour le QAP se rapprochant du *Branch and Bound*. Il utilise également une borne inférieure comme visibilité locale et une procédure d'amélioration locale, mais modifie aussi la mise à jour de la trace. Il n'y a aucune mise à jour locale et la mise à jour globale est faite en fonction de la borne inférieure de la solution partielle et de la moyenne des coûts obtenus pour les dernières solutions complétées. La méthode proposée se montre très performante lorsque comparée à la recherche avec tabous et au GRASP, autant en termes de qualité de solutions, qu'en termes de temps de calcul.

Le routage des messages dans les réseaux de télécommunication représente également un domaine d'application important de l'OCF. Il s'agit d'un problème multi-objectifs stochastique et distribué qui consiste à établir des tables de routage qui minimisent le temps nécessaire à l'envoi de messages entre des paires de noeuds et balancent le trafic sur le réseau (*Load Balancing*). La table de routage de chaque nœud redirige chaque paquet entrant vers un nœud voisin, en fonction de son nœud de destination. Pour la résolution par OCF, des fourmis sont lancées sur le réseau à intervalles plus ou moins constants, à partir d'un point de départ et avec une destination aléatoire. Il existe une matrice de phéromone pour chaque nœud qui est en fait une matrice de probabilités de taille $((Nombre\_de\_noeuds-1)\times(Nombre\_de\_noeuds-1))$, où pour chaque nœud de destination, il existe une probabilité de router le message vers chaque

nœud connecté (les nœuds inaccessibles directement par le nœud propriétaire de la matrice ont une probabilité nulle). Lorsqu'une case de la matrice est modifiée, au moins une case correspondant à un autre nœud connecté doit l'être également pour conserver des probabilités valables. La matrice d'un nœud est mise à jour par les fourmis qui y passent. Schoonderwoerd *et al.* [SCH'97a] [SCH'97b] ont utilisé ce principe de dépôt de phéromone dans un algorithme nommé l'*Ant-Based Control* (ABC). Dans l'ABC, la mise à jour de la phéromone sur le lien emprunté se fait en fonction de l'âge de la fourmi lorsqu'elle arrive au nœud, qui est égal au temps qu'elle a mis pour se rendre de son point de départ au nœud. Les autres cases de la matrice sont diminuées. Cette méthode comporte de nombreux avantages selon les auteurs lorsqu'ils la comparent aux agents mobiles. Les résultats sont très encourageants pour les problèmes théoriques testés et les auteurs proposent donc de tester l'algorithme sur des instances réelles pour statuer sur la qualité de la méthode. Di Caro et Dorigo [DIC'97] [DIC'98a] [DIC'98b] ont proposé l'*AntNet*, où deux groupes de fourmis sont à l'œuvre : les *forward* et les *backward*. Lorsqu'une fourmi *forward* arrive à sa destination, elle transmet sa mémoire à une fourmi *backward* qui refait en sens inverse son chemin et met à jour la trace de phéromone. Comme les valeurs optimales ne peuvent être connues puisqu'elles dépendent du réseau et de son état actuel, l'âge d'une fourmi pris seul ne peut être un bon indicateur de qualité de solution. La valeur à ajouter lors de la mise à jour est donc calculée en fonction de l'âge de la fourmi et de son âge moyen pour les voyages précédents. Comparé par les auteurs à d'autres

algorithmes de résolution, l'AntNet se montre toujours aussi efficace et parfois plus que les autres, avec une fluctuation moins grande. Elle ne surpasse cependant pas la méthode classique appelée *Deamon*, mais les auteurs considèrent sa performance comme excellente. Denby et Le Hégarat-Mascle [DEN'03] ont adapté l'AntNet à un problème de routage des communications satellites. L'application sur des réseaux à trafic normal et élevé produit des résultats proches de ceux de l'algorithme exact de *Dijkstra*, ce qui permet aux auteurs de dire que la méthode atteint des solutions près de l'optimal. De plus, l'AntNet développé se révèle plus efficace et moins demandant en ressources de calcul que le *shortest path first*.

En terminant, des algorithmes de fourmis ont été appliqués à beaucoup d'autres problèmes pour lesquels de bons résultats ont été obtenus. Parmi ces problèmes se trouvent le problème d'ordonnancement séquentiel (*sequential ordering problem*) [GAM'97], le problème de tournée de véhicules (*vehicule routing problem*) [BUL'99b] [BUL'99c], le problème d'ordonnancement sur une machine unique (*single machine scheduling*) (uni et multi-objectifs) [GAG'02b] [GAG'02a] [GRA'02] et le problème d'optimisation de l'aménagement du clavier (*Optimization of keyboard arrangement problem*) [EGG'03], pour n'en nommer que quelques-uns.

2.4 Le Problème d'Ordonnancement de Voitures

2.4.1 Formulation

Le Problème d'Ordonnancement de Voitures (POV), mieux connu sous le nom de *car-sequencing problem*, est apparu dans le secteur de la production automobile lorsque les fabricants ont abandonné la production de masse pour adopter la personnalisation des véhicules. Avec les demandes variées des clients, les lignes de montage multimodèles ont fait leur apparition, accompagnées d'un lot de nouvelles contraintes [LOP'00]. Bien qu'il représente une problématique industrielle concrète, le POV décrit dans la présente section est considéré comme le modèle théorique du problème. Le problème traité dans la littérature est en effet une version épurée de la problématique réelle. On peut supposer qu'il est ainsi plus simple à modéliser et que les impacts des méthodes utilisées sont plus facilement identifiables.

Le POV consiste à donner l'ordre dans lequel des automobiles doivent être produites, en considérant les options de chacune et les contraintes de capacité de la chaîne de montage. En 1998, Gent et Walsh [GEN'98] ont proposé une preuve de la NP-complétude du POV par une réduction du chemin hamiltonien. Cette démonstration a cependant été jugée incorrecte par Kis [KIS'04] qui a prouvé par une autre démarche en 2004 que le POV était non seulement un problème NP-difficile, mais qu'il l'était au sens fort.

Dans l'usine, les voitures avancent à une vitesse constante sur la courroie d'un convoyeur. Celui-ci transporte les véhicules, un à la suite de l'autre, pour les

faire passer par chaque poste de travail, même si chaque voiture n'y est pas nécessairement traitée. À chaque poste est associée une option. Le lecteur de CD, le toit ouvrant et les freins anti-blocage sont des exemples d'options. On connaît le carnet de commande qui indique pour chaque véhicule à produire les options requises (demandées par le client).

Une instance de problème a d'abord une taille n qui indique le nombre total de voitures à placer. Une instance se caractérise également par son nombre total d'options o et pour chaque option j, il existe une contrainte de capacité q_j / p_j. Comme chacune de ces options requiert un temps de traitement différent et que les véhicules à produire en demandent différentes combinaisons, on doit tenir compte des contraintes de *capacité* des divers postes de travail de la ligne. Par exemple, le poste des toits ouvrant peut présenter une contrainte de la forme 2/3 qui signifie que sur n'importe quelle sous-séquence de 3 véhicules consécutifs sur la ligne, au plus, 2 peuvent demander le toit ouvrant. S'il arrive que plus de 2 voitures demandent l'option dans un même bloc de 3 véhicules, il y a alors *violation de la contrainte de capacité* de l'option. Ce sont ces violations de contraintes que l'on cherche à minimiser, puisqu'elles entraînent des coûts supplémentaires ou parfois même l'invalidité d'une séquence à produire.

On peut regrouper les véhicules demandant les mêmes options pour créer des classes de voitures et ainsi simplifier la résolution. Pour chacune des v classes ainsi formées, on connaît le nombre de voitures à produire. Ces quantités

représentent les contraintes de *production* qui exigent qu'exactement c_i véhicules de la classe i soient produits.

La composition de chacune des classes est notée dans une matrice A de taille $(v \times o)$, affectée de la manière suivante :

$$a_{ij} = \begin{cases} 1 & si\ la\ classe\ i\ requiert\ l'option\ j \\ 0 & sinon \end{cases} \tag{2.9}$$

Le nombre total de voitures b_j nécessitant une option j, toutes classes confondues, est calculé selon l'Équation 2.10.

$$b_j = \sum_{i=1}^{v} a_{ij} c_i \tag{2.10}$$

Le Tableau 2.2 présente un exemple d'instance de POV tiré de CSPLib [GEN'99]. Pour ce problème, on observe les valeurs suivantes : $n = 200$, $o = 5$, $v = 17$. Les contraintes q_j / p_j associées à chacune des 5 options sont $\{1/2, 2/3, 1/3, 2/5, 1/5\}$. La colonne « Nombre de voitures » donne c_i pour chaque classe, dont la composition est donnée par les 5 colonnes suivantes, qui correspondent au contenu de la matrice A. Finalement, le nombre de voitures b_j requérant chaque option j est donné dans la dernière ligne du tableau.

PROB60_02						
Taille (n)	200					
Nombre d'options (o)	5					
Nombre de classes (v)	17					
Options (j)		1	2	3	4	5
Contrainte de capacité (q_j/p_j)		1/2	2/3	1/3	2/5	1/5
Classes (i)	Nombre de voitures (c_i)	Composition (a_{ij})				
1	2	1	0	1	1	0
2	2	1	0	0	0	1
3	4	1	1	0	0	0
4	1	1	1	1	0	0
5	30	0	0	0	0	1
6	1	1	1	0	1	0
7	1	1	0	1	0	1
8	39	0	0	1	0	0
9	54	0	0	0	1	0
10	1	0	1	1	0	1
11	2	0	0	1	1	0
12	3	1	0	1	0	0
13	1	1	0	0	1	1
14	53	0	1	0	0	0
15	1	0	1	1	0	0
16	1	0	1	0	0	1
17	4	1	0	0	0	0
Nombre de voitures requérant l'option (b_j)		19	62	50	60	36

Tableau 2.2 : Une instance de POV (prob60_02)

Un résumé de la notation utilisée pour le POV est donné au Tableau 2.3 .

Élément	Description
n	Taille du problème, i.e. nombre de voitures à placer
q_j / p_j	Contrainte de capacité pour l'option j
b_j	Nombre total de voitures requérant l'option j
o	Nombre d'options
v	Nombre de classes de voitures
c_i	Nombre de voitures de la classe i
a_{ij}	Composition des classes (vrai, si la classe i comprend l'option j)

Tableau 2.3 : Notation générale utilisée pour le POV

Afin d'évaluer une solution, la séquence de classes correspondante est traduite en o séquences binaires, une pour chaque option. Les éléments à chacune des positions de ces séquences indiquent si la voiture correspondante possède l'option en question. Le lecteur peut se référer à l'exemple de la Figure 2.5 qui illustre une séquence partielle de solution pour l'instance du Tableau 2.2. On observe en (a) qu'il en résulte o vecteurs binaires qui indiquent pour chaque position si la classe de voitures placée nécessite chacune des options. Chacun des vecteurs binaires, correspondant à une option j, est pris séparément. Pour compter les violations de contrainte, on isole tous les blocs complets de p_j voitures de la séquence binaire et on calcule la somme de leurs éléments. Si cette somme est plus grande que le q_j permis, on dit alors qu'il y a violation de la contrainte ou *conflit*. Dans l'exemple de la Figure 2.5 en (b), on a choisi l'option 3 qui a une contrainte de capacité de 1/3 pour fin de démonstration. On voit en pointillés les

différents blocs évalués, et les deux blocs générant des conflits sont encadrés de lignes pleines. En répétant cette méthode pour chaque option, on obtient un nombre de conflits pour la sous-séquence donnée, tel que montré en (c). En l'appliquant sur toute la longueur de la séquence, on obtient un nombre conflits qui indique la qualité globale de la solution.

Sous-séquence d'une solution au problème du Tableau 2.2 à décomposer :

... 17 4 2 13 8 10 2 ...

(a) Pour chaque option j, avec q_j/p_j entre parenthèses :

1	(1/2)	...	1	1	1	1	0	0	1	...
2	(2/3)	...	0	1	0	0	0	1	0	...
3	(1/3)	...	0	1	0	0	1	1	0	...
4	(2/5)	...	0	0	0	1	0	0	0	...
5	(1/5)	...	0	0	1	1	0	1	1	...

(b) On a isolé l'option 3 dont la contrainte de capacité est de 1/3.

... 0 1 0 0 1 1 0 ...

Pour l'option 3, cette sous-séquence donne 2 conflits.

(c) En résumé, on a 8 conflits dans la sous-séquence qui sont répartis ainsi :

1	(1/2)	...	1	1	1	1	0	0	1	...
2	(2/3)	...	0	1	0	0	0	1	0	...
3	(1/3)	...	0	1	0	0	1	1	0	...
4	(2/5)	...	0	0	0	1	0	0	0	...
5	(1/5)	...	0	0	1	1	0	1	1	...

Figure 2.5 : Évaluation d'une sous-séquence de solution

On évalue la difficulté d'une instance par le nombre n de voitures à ordonnancer, mais aussi en calculant la moyenne des taux d'utilisation tu de ses options. Le taux d'utilisation est un indicatif de l'espace nécessaire pour placer

toutes les voitures requérant l'option j sans créer de conflit : plus le taux est grand, moins on dispose de marge pour placer les voitures requérant l'option en question. Le taux d'utilisation tu_j se calcule selon l'Équation 2.11. Si $tu_j > 1$, l'option j est dite *sur-contrainte* (*over-constrained*). Dans ce cas, l'instance contenant l'option est nécessairement impossible à résoudre sans conflit. Cependant, le fait que $tu_j \leq 1$ pour toutes les options j ne garantit pas qu'une solution sans conflit existe, puisqu'il peut résulter des incompatibilités de la combinaison des options.

$$tu_j = \frac{b_j \cdot q_j}{n \cdot p_j}, où\ j\ est\ une\ option \tag{2.11}$$

Une borne inférieure BI du nombre de conflits pour les instances montrant une seule option sur-contrainte a été calculée par Warwick et Tsang [WAR'95]. Cette borne est donnée à l'Équation 2.12, où j est l'option sur-contrainte et $Excédent(j)$ représente le nombre excédentaire de voitures ayant l'option j.

$$BI = \frac{Excédent(j)}{p_j - q_j} \cdot p_j - q_j \tag{2.12}$$

La valeur de $Excédent(j)$, dont le calcul est donné par l'Équation 2.13, est obtenue à partir du nombre de voitures présentes dans l'instance et du nombre maximal permis pour ne pas engendrer de conflits.

$$Excédent(j) = b_j - Maximum(j)$$

$$où\ Maximum(j) = n \cdot \frac{q_j}{p_j} \tag{2.13}$$

Les instances de problèmes qui seront utilisées pour ce travail se divisent en 3 ensembles. Les deux premiers sont tirés de CSPLib [GEN'99] et identifiés comme ensemble test 1 (ET1) et ensemble test 2 (ET2). L'ET1 contient 70 instances regroupées dans 7 sous-groupes de 10 instances chacun, selon le taux d'utilisation moyen pour l'ensemble des options, de 0,60, 0,65, 0,70, 0,65, 0,70, 0,75, 0,80, 0,85 et 0,90. Les instances avec un taux d'utilisation moyen de 0.XX sont identifiées XX-01, ..., XX-10 et on se réfère au groupe par XX-*. Toutes ces instances sont possibles à résoudre sans conflit et possèdent 200 voitures, 5 options et entre 17 à 30 classes de voitures. L'ET2 a été produit par deux générateurs de problèmes, l'un développé à l'Université de Leeds et l'autre à l'Université de Essex [SMI'97]. Il est composé de 9 instances plus difficiles comprenant 100 voitures, 5 options, entre 18 à 24 classes et des taux d'utilisation moyens élevés (égalant 1 pour certaines options). L'ensemble test 3 (ET3) a été généré aléatoirement par Gravel *et al.* [GRA'05]. Il contient 30 instances de taille 200, 300 ou 400 et présente les mêmes nombres d'options et de classes que l'ET1. Des taux d'utilisation des options dépassant 90% et la combinaison des différentes options dans les classes rendent ces instances plus difficiles.

Le Tableau 2.4 regroupe les instances des 3 ensembles test. La colonne « Nombre de conflits » donne le nombre de conflits de la meilleure solution connue pour chaque instance. Les références informent sur les publications des auteurs ayant atteint les premiers ces solutions. Les instances dont on ne connaît pas la solution optimale sont identifiées par $\star$.

ET1		ET2			ET3		
Groupe de séquences	Nombre de conflits [GEN'99]	Instance	Nombre de conflits		Instance		Nombre de conflits [GRA'05]
60-*	0	10-93	3	[GOT'03]	200_01		0
65-*	0	16-81	0	[GOT'03]	200_02	★	2
70-*	0	19-71	2	[GEN'98] [GOT'03]	200_03	★	4
75-*	0	21-90	★ 2	[GOT'03]	200_04	★	7
80-*	0	26-82	0	[GOT'03]	200_05	★	6
85-*	0	36-92	2	[GOT'03]	200_06	★	6
90-*	0	41-66	0	[GOT'03]	200_07		0
		4-72	0	[REG'97]	200_08	★	8
		6-76	6	[GOT'03]	200_09	★	10
					200_10	★	19
					300_01		0
					300_02	★	12
					300_03	★	13
					300_04	★	7
					300_05	★	29
					300_06	★	2
					300_07		0
					300_08	★	8
					300_09	★	7
					300_10	★	21
					400_01	★	1
					400_02	★	16
					400_03	★	9
					400_04	★	19
					400_05		0
					400_06		0
					400_07	★	4
					400_08	★	4
					400_09	★	5
					400_10		0

★ Meilleure solution connue à ce jour (non nécessairement optimale)

Tableau 2.4 : Meilleures solutions connues pour les ensembles test

Notons en terminant que le POV peut également être abordé en considérant des objectifs différents [LOP'00]. Parmi ceux-ci se trouvent la minimisation des violations de contraintes de groupement lorsqu'il est avantageux de placer les voitures de même type à proximité, de lissage de charge lorsqu'on cherche à uniformiser la demande en main d'œuvre tout au long de la chaîne, et de juste-à-temps où l'on veut garder constant le taux d'utilisation de différentes pièces.

2.4.2 Résolution par les méthodes exactes

Parmi les méthodes de résolution exactes pour résoudre le POV, on retrouve nombre d'algorithmes adaptés aux problèmes de satisfaction de contraintes (*Constraint Satisfaction Problem* - CSP). Ces derniers sont utilisés pour déterminer si une instance d'un problème possède ou non une solution sans conflit. Il ne s'agit donc pas de chercher la meilleure solution possible comme dans le domaine de l'optimisation.

Tsang [TSA'93a] définit un CSP comme étant un problème composé d'un ensemble fini de variables, chacune associée à un domaine fini, et d'un ensemble de contraintes qui restreignent les valeurs pouvant être prises simultanément par les variables. La tâche à accomplir consiste à assigner une valeur à chaque variable, tout en satisfaisant les contraintes.

On dit qu'une instance de CSP est *satisfaisable* lorsqu'il existe une affectation des variables qui ne viole aucune contrainte et *non satisfaisable* dans le cas contraire. Dans ce second cas, il s'agit d'un Maximal-CSP [FRE'92] (aussi appelé

Partial CSP [WAR'95]) où l'on cherche une solution qui satisfait autant de contraintes possible, i.e. qui minimise le nombre de conflits.

Les algorithmes de CSP traditionnels garantissent l'optimalité des résultats par une exploration complète ou partielle de l'arbre de solutions, où chaque arc correspond à l'affectation d'une valeur à une variable. Les nœuds de l'arbre sont examinés et sondés selon une méthode donnée et la gestion des « culs de sac » peut être faite de différentes manières (*backtracking*, *back jumping*, *forward checking*, etc.). À mesure que l'algorithme parcourt l'arbre de solutions, il construit une solution en affectant une valeur à chaque variable, qu'il peut remettre en question par la suite [TSA'93b] [TSA'93c].

L'outil *Ilog Solver* est une librairie commerciale de programmation par contraintes supportant la définition de nouvelles contraintes. Il est fréquemment utilisé pour l'expérimentation dans le domaine, compte tenu de sa flexibilité et de sa rapidité de développement.

Le POV a longtemps été traité comme un problème de satisfaction de contraintes dans la littérature et certaines instances ont donc pu être résolues par les méthodes de CSP traditionnelles. Il appartient à un cas particulier de CSP appelé PermutCSP. La tâche consiste alors à trouver une permutation de n valeurs connues pouvant être assignées à n variables, sous certaines contraintes [SOL'00]. Deux formulations ont été expérimentées dans la littérature. Les positions de la séquence du POV peuvent être les variables et les classes de voitures les valeurs possibles (le domaine des variables). Inversement, les

variables peuvent être chacune des voitures (sans regroupement en classes) et les valeurs, les positions dans la séquence. La première formulation est celle généralement employée dans la communauté scientifique en raison du nombre de solutions différentes possibles qui est de v^n en comparaison de n^n pour la seconde [SMI'97].

Les contraintes à respecter sont de 2 types : celles de capacité des options ainsi que celles de production. Ces contraintes sont n-aires (par opposition à binaire), ce qui signifie qu'elles sont reliées à plus de 2 variables. La particularité du POV réside dans ses contraintes globales de séquencement (*sequencing global constraints*) de la forme « au plus » (*at most*) qui rendent le problème beaucoup plus difficile à résoudre [DAV'94].

Régin et Puget [REG'97] ont fait un travail important pour la recherche dans le domaine de la résolution du POV par les méthodes de CSP traditionnelles. Ils proposent un algorithme de filtrage qui s'avère très efficace pour le problème. Il s'agit d'un algorithme de type *backtracking* où le filtrage des contraintes est effectué à chaque nœud. Les auteurs ont amené le concept de marge (*slack*), qui est une mesure de la difficulté de placer toutes les voitures nécessitant une option j. Cette marge est calculée selon l'Équation 2.14.

$$Marge_j = n - q_j \left(\frac{b_j}{p_j} \right) \tag{2.14}$$

Une marge négative indique l'impossibilité de placer toutes les voitures possédant l'option j sans créer de conflits. Une valeur positive ou nulle donne le nombre de positions qui ne sont pas nécessairement utilisées pour placer les voitures et qui sont donc disponibles pour éviter les conflits sur les autres options. Au lieu de choisir une variable (une position) et de lui affecter une valeur (une classe), l'idée ici est de choisir une option et de tenter de placer toutes les voitures la requérant, toujours sans créer de conflits. On y arrive en ne laissant dans les domaines des variables que les classes qui demandent l'option sélectionnée. Les options sont choisies dans l'ordre croissant de leur marge. Une particularité importante de l'arbre de recherche est donc que les arcs ne sont pas des affectations de valeurs à des variables, mais des affectations d'options à des variables. Cette méthode est particulièrement adaptée aux problèmes comportant des contraintes globales de séquencement [REG'97].

Smith [SMI'97] s'est attardée sur l'ordre d'instanciation des variables pour un problème de POV. Après avoir testé plusieurs heuristiques d'ordonnancement de variables, l'auteur arrive à la conclusion que celles-ci doivent être affectées dans l'ordre numérique, du début à la fin de la séquence, sans laisser de vides à combler plus tard. On se tourne ensuite vers l'ordonnancement des valeurs affectées pour améliorer la méthode. Trois heuristiques de choix de valeurs ont été expérimentées: choisir en premier (1) les classes requérant les options avec les tu_j les plus grands ; (2) les classes comportant le plus d'options ; et (3) les classes pour lesquelles $g(i)$ est la plus grande, où $g(i)$ est une fonction représentant la

difficulté de placer une classe i, en tenant compte du taux d'utilisation de ses options et de son nombre d'options. Les trois heuristiques semblent équivalentes en termes de temps de calcul et d'efficacité. Ces résultats ne sont cependant pas satisfaisants selon l'auteur.

Inspiré de la modélisation de Smith [SMI'97], Boivin [BOI'05] a présenté un algorithme de filtrage avant (forward checking) utilisant les contraintes de capacité de la chaîne d'assemblage ainsi que les contraintes implicites du problèmes [DIN'88] pour favoriser la recherche de solutions. Cette méthode s'est montrée efficace sur l'ET1. De plus, des heuristiques d'ordonnancement de valeurs orientant la recherche ont été testées. Une version concurrente et bonifiée d'heuristiques a permis d'obtenir de bons résultats sur l'ET2. L'importance des heuristiques d'ordonnancement de valeurs a été démontrée par les résultats de l'algorithme et cette conclusion a été confirmée à l'aide d'Ilog Solver.

Pour conclure sur les méthodes de CSP, il a été observé que celles-ci ne représentent pas toujours une solution adéquate pour la résolution du POV [CHE'92] [SMI'97]. Le POV est complexe en termes de contraintes et il en résulte un CSP de très grande taille. Le temps requis pour l'exécution des algorithmes est trop long pour qu'ils soient appliqués en milieu industriel, où d'autant plus, l'optimalité n'est pas toujours une nécessité.

La programmation linéaire (*Linear Programming*) est une autre méthode exacte pour la résolution des problèmes d'optimisation. Gravel *et al.* [GRA'05] ont comparé la performance d'un modèle de programmation linéaire en nombres

entiers (*Integer Linear Programming)* à celle d'un OCF (qui sera analysé plus en détails à la section suivante). L'outil de résolution utilisé est XPRESS. Pour l'ET1, le modèle a rapidement trouvé une solution à chacun des problèmes. Dans l'ET2, 4 problèmes ont également été résolus dans un court laps de temps. Les 5 autres ont demandé un temps plus long et il a seulement été possible de confirmer les meilleures solutions connues. Ces résultats sont rassemblés dans le Tableau 2.5 pour l'ET1 et dans le Tableau 2.6 pour l'ET2.

Instances \ Méthodes	Taux de réussite (%)						
	GENET [DAV'99]	Swap-GENET [DAV'99]	E-GENET [LEE'95]	Ant-P-Solver [SOL'00]	Ilog Solver [SOL'00]	ILP [GRA'05]	OCF [GRA'05]
60-*	100	100	74	100	40	100	100
65-*	100	100	80	---	---	100	100
70-*	100	100	81	100	20	100	100
75-*	100	100	84	---	---	100	100
80-*	100	100	53	100	60	100	100
85-*	100	100	---	98	50	100	100
90-*	100	100	---	57	80	100	100

Tableau 2.5 : Récapitulatif des taux de réussite (%) obtenus pour l'ET1

Instance	Meilleure connue	Gourmand (DHU) [GOT'03]	OCF [GOT'03]	Méthodes Recherche locale (DED) [GOT'03]	OCF [GRA'05]	OCF + recherche locale [GRA'05]
10-93	3	12,0	4,7	5,5	4,24	3,8
16-81	0	11,0	0,9	1,6	0,12	0,0
19-71	2	7,0	2,7	2,1	2,08	2,0
21-90	★ 2	7,0	2,0	2,2	2,63	2,0
26-82	0	3,0	0,0	0,3	0,00	0,0
36-92	2	7,0	2,0	2,9	2,29	2,0
41-66	0	2,0	0,0	0,0	0,00	0,0
4-72	0	2,0	0,0	0,8	0,00	0,0
6-76	6	6,0	6,0	6,0	6,00	6,0

★ Meilleure solution connue à ce jour (non nécessairement optimale)

Tableau 2.6 : Récapitulatif des nombres de conflits moyens obtenus pour ET2

2.4.3 Résolution par les méthodes heuristiques

Au milieu des années '80, Parrello et Kabat [PAR'86] ont abordé le problème dans l'optique de l'intelligence artificielle classique. Ils ont présenté et décrit en détails une modélisation du problème à base de prédicats et de règles de production. L'outil ITP (*Interactive Theorem Proving*) a fourni le moteur d'inférence nécessaire à la résolution des problèmes. Le but était de déterminer la faisabilité d'un système expert développé sous ITP. Selon les auteurs, un POV de taille intéressante et concrète ne peut être résolu efficacement par les outils disponibles à ce moment, en raison de la complexité de ses prédicats. Ces derniers ne touchent pas seulement un élément, mais un ensemble d'éléments, ce qui amène l'utilisation d'une base de connaissances de second ordre. Les auteurs restent cependant ouverts sur la méthode, expliquant que les outils évoluent.

Suite à ces travaux, Dincbas *et al.* [DIN'88] ont proposé un modèle de programmation logique par contrainte (*constraint logic programming*). Ils ont utilisé le langage CLIPS qui associe la forme relationnelle, les variables logiques et le caractère non-déterministe de la programmation logique aux mécanismes puissants de résolution de la programmation par contraintes. Les auteurs présentent le modèle de résolution et les résultats obtenus pour des instances générées aléatoirement. Les instances les plus difficiles étaient de taille n = 200 et avaient un taux moyen d'utilisation de 90%. Dans tous les cas, le modèle est parvenu à trouver une solution sans conflit. Cependant, les temps requis étaient considérables, ce qui a amené les auteurs à envisager l'ajout d'heuristiques.

Davenport et Tsang [DAV'99] ont adapté GENET pour le POV. Il s'agit d'un algorithme de *Hill-Climbing* développé en tant que réseau de neurones pour la résolution de CSP à contraintes binaires et basé sur des principes de réparation des solutions. GENET comprend une variation de l'heuristique de *Min-Conflict* [MIN'93]. Cette dernière, à la différence des méthodes de CSP traditionnelles, travaille sur des affectations complètes des variables, lui donnant la possibilité d'exploiter l'information qu'une solution partielle ne pourrait donner. La procédure consiste à choisir, dans une solution complète, des positions faisant partie de conflits et d'en changer la valeur en minimisant le nombre de conflits. La différence majeure entre le *Min-Conflict* et GENET est que ce dernier est en mesure de sortir d'un optimum local par des mouvements latéraux dans l'espace de solutions en

utilisant le poids affecté à chaque contrainte. De GENET, a été inspiré le Swap-GENET [DAV'99] qui se distingue par la manière de modifier la valeur d'une variable (position) en conflit. Une variable en conflit est choisie, un échange avec chaque élément différent de celle-ci est évalué et l'échange qui crée le moins de conflits est choisi. Cette méthode est particulièrement appropriée pour les problèmes d'ordonnancement : on respecte ainsi automatiquement les contraintes de séquence. GENET et Swap-GENET ont été comparés sur les problèmes de l'ET1 [DAV'99]. Généralement, Swap-GENET est supérieur à GENET. Les résultats sont transcrits dans le Tableau 2.5, où les cellules contenant « --- » indiquent des données inexistantes. Pour les problèmes à taux d'utilisation faible, GENET requiert moins de temps de calcul, alors que pour les problèmes plus contraints, Swap-GENET est nettement plus efficace.

Vu son efficacité, GENET a encouragé le développement d'algorithmes dérivés. Lee *et al.* [LEE'95] se sont inspirés de GENET et ont modifié la modélisation du réseau original pour étendre les résolutions aux problèmes à contraintes non binaires (ce qui à ce moment n'était pas reconnu) en introduisant E-GENET (*Extended GENET*). Il en résulte un réseau plus homogène en termes de contraintes, plus petit, donc moins gourmand en mémoire, et plus flexible. On a comparé E-GENET avec le GENET original et il s'est avéré que pour les instances du POV, E-GENET avait un taux de succès légèrement inférieur à GENET, mais qu'il utilisait près de la moitié moins de réparations pour arriver à ces résultats. Le Tableau 2.5 présente les résultats obtenus avec le E-GENET, avec une limite de

1000 réparations. De manière générale, Lee *et al.* concluent que E-GENET est aussi performant que GENET sur les problèmes à contraintes binaires, mais qu'il surpasse son concurrent pour les problèmes *n*-aires.

Warwick et Tsang [WAR'95] ont utilisé les Algorithmes Génétiques (*Genetic Algorithms*) pour la résolution du POV. Ils proposent le GAcSP, un algorithme générique comportant un mécanisme élitiste, un opérateur de croisement adaptatif et un processus de réparation et d'amélioration locale des enfants, spécifiquement modélisé pour la résolution de Max-CSP. Sur des instances générées aléatoirement, on a comparé les résultats du GAcSP, d'une recherche avec tabous et d'un *Heuristic-Repair*. Il en résulte que le GAcSP n'est pas affecté par la taille du problème et qu'il surpasse les deux autres algorithmes mis à l'épreuve. Hyun *et al.* [HYU'98] se sont penchés sur le problème multi-objectifs de POV. En plus de minimiser le nombre de conflits concernant les options, on tente de conserver un taux d'utilisation constant des pièces (utile dans les systèmes en juste-à-temps) et on minimise les coûts reliés aux temps de mise en course (*setup*).

Solnon [SOL'00] a produit l'Ant-P-Solver, un algorithme d'OCF pour la résolution du POV. Se basant sur les travaux en ordonnancement des variables et des valeurs de Régin et Puget [REG'97] et de Smith [SMI'97], la règle de transition a été adaptée. La visibilité locale utilisée est le nombre de nouveaux conflits engendrés par la classe. On y a également introduit un taux d'utilisation, représenté par la somme des taux d'utilisation des options requises par la classe candidate. On cherche ainsi à favoriser le choix des classes les plus contraintes en

début de séquence. Les résultats obtenus avec l'Ant-P-Solver sur les instances de l'ET1 sont comparés avec ceux d'Ilog Solver. Ces résultats apparaissent au Tableau 2.5. L'Ant-P-Solver était limité à 5000 cycles, Ilog Solver utilisait les contraintes globales IlcSequence et était limité à 3600 secondes de temps de processeur. Les résultats montrent l'efficacité de l'Ant-P-Solver pour les instances 60-* à 80-*. On s'explique ces résultats par l'incapacité pour Ilog Solver de faire un filtrage efficace pour les instances ayant un taux d'utilisation global moins élevé [SOL'00]. Par la suite, on a bonifié l'algorithme de départ d'une recherche locale (*Min-Conflict, Random-walk*) et d'une phase de prétraitement [SOL'02]. L'ajout de la recherche locale a permis de conclure que les deux méthodes se complètent bien et tirent avantage des caractéristiques l'une de l'autre. La phase de prétraitement s'est montrée efficace pour améliorer l'exploration de l'espace de solutions. Cependant, aucun résultat concernant le POV n'est disponible.

Gottlieb *et al.* [GOT'03] ont comparé les performances de plusieurs algorithmes de résolution. Les auteurs ont expérimenté plusieurs versions d'algorithme gourmand, d'OCF et de recherche locale. L'algorithme gourmand a été obtenu en utilisant $\alpha = 0$ avec l'algorithme d'OCF. L'objectif était de tester l'efficacité de différentes heuristiques de choix pour les classes de voitures. L'heuristique retenue pour l'OCF et l'algorithme gourmand est le *Dynamic Highest Utlization rate* (DHU), qui utilise une version dynamique du tu_j des options. La recherche locale est guidée par l'heuristique de *Dynamic Even Distribution* (DED), qui cherche à répartir le plus également possible la demande des différentes

options au cours de la séquence. Les meilleures versions de chaque méthode parviennent à résoudre toutes les instances de l'ET1 avec un taux de réussite de 100%. Les nombres de conflits moyens obtenus pour l'ET2 sont donnés au Tableau 2.6.

Gravel *et al.* [GRA'05] ont conçu un algorithme d'OCF pour lequel la règle de transition est composée de la trace de phéromone, du nombre de nouveaux conflits engendrés par l'ajout de la classe et d'une marge dynamique pour cette même classe. Cet algorithme est expliqué à la Section 3.2, le lecteur peut donc s'y référer pour plus de détails. Les taux de réussite pour les instances de l'ET1, tous à 100%, sont donnés au Tableau 2.5 et les nombres de conflits moyens obtenus pour l'ET2, au Tableau 2.6. Par la suite, une phase de recherche locale a été ajoutée à l'algorithme afin d'améliorer ses performances sur les problèmes de l'ET2. Les résultats obtenus sont meilleurs que sans la recherche locale, mais le temps de calcul a légèrement augmenté. Les auteurs ont observé que l'ajout de la recherche locale à l'OCF diminue le nombre de cycles nécessaires pour obtenir la solution finale, à l'exception de l'instance 10_93. L'algorithme final avec recherche locale a été ensuite exécuté pour résoudre les instances de l'ET3. Le Tableau 2.7 présente les résultats obtenus, où le symbole ⋆ identifie les instances pour lesquelles ont ne connaît pas la solution optimale.

Instance	Meilleure connue	Nombre de conflits
200_01	0	1,00
200_02	★ 2	2,41
200_03	★ 4	6,04
200_04	★ 7	7,57
200_05	★ 6	6,40
200_06	★ 6	6,00
200_07	0	0,00
200_08	★ 8	8,00
200_09	★ 10	10,00
200_10	★ 19	19,09
300_01	0	2,15
300_02	★ 12	12,02
300_03	★ 13	13,06
300_04	★ 7	8,16
300_05	★ 29	32,28
300_06	★ 2	4,38
300_07	0	0,59
300_08	★ 8	8,00
300_09	★ 7	7,46
300_10	★ 21	22,60
400_01	★ 1	2,52
400_02	★ 16	17,37
400_03	★ 9	9,91
400_04	★ 19	19,01
400_05	0	0,01
400_06	0	0,33
400_07	★ 4	5,44
400_08	★ 4	5,30
400_09	★ 5	7,63
400_10	0	0,95

★ Meilleure solution connue à ce jour (non nécessairement optimale)

Tableau 2.7 : Nombres de conflits moyens obtenus par Gravel *et al.* [GRA'05] pour l'ET3 à l'aide d'un OCF avec recherche locale

En conclusion, de plus en plus d'efforts sont faits dans la communauté scientifique pour la résolution du POV. On observe que l'OCF avec recherche

locale proposé par Gravel *et al.* est, pour l'instant, la méthode la plus efficace pour la résolution des instances test proposées, pour l'ET1 et pour la majorité des instances de l'ET2. L'ET3 présente un intérêt compte tenu de la grande difficulté de ses instances. Bien qu'il y ait encore très peu de résultats disponibles pour l'ET3, on peut croire qu'il pourra être utilisé pour évaluer plus efficacement la performance des algorithmes.

2.4.4 Généralisation du POV en contexte industriel

Comme pour bon nombre de problèmes théoriques, il existe pour le POV des versions appliquées, tirées directement du domaine industriel. La problématique du POV est rencontrée au quotidien par les producteurs automobiles qui font face, cependant, à des versions du problème plus complexes et/ou difficiles à résoudre.

Un nombre plus restreint de chercheurs se sont consacrés au problème de nature industrielle, qu'on appellera le POV+. Comme ce dernier est tiré d'applications en usine, une diversité de modélisations et d'approches a émergé, mais peu d'entre elles ont pu être publiées. Deux grandes approches ont été utilisées dans la littérature pour les problèmes de POV+ : l'ordonnancement et le cadencement [LOP'00], dont la première est la plus exploitée et consiste, tout comme pour le POV, à donner l'ordre dans lequel les véhicules doivent être produits.

La Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF) a récemment publié un document formalisant la problématique

rencontrée concrètement par le producteur automobile français Renault [NGU'03]. Cette publication s'est faite dans le cadre du « Challenge ROADEF'2005 » et en collaboration étroite avec Renault. Cette version du POV+ est expliquée brièvement dans les paragraphes suivants.

Dans les usines observées, les voitures doivent passer par 3 ateliers distincts au cours de leur production et l'ordonnancement des voitures doit être le même pour chacun d'eux. Ces ateliers sont : la tôlerie, la peinture et le montage. Dans sa version théorique, le POV s'arrête uniquement aux contraintes de l'atelier de montage. Pour sa part, le POV+ tient compte des ateliers de peinture et de montage, considérant que l'atelier de tôlerie génère des contraintes négligeables.

Le matériel de peinture ne peut produire qu'une seule couleur à la fois et doit être obligatoirement nettoyé lorsque deux voitures consécutives n'ont pas la même couleur ou lorsqu'un nombre maximum donné de voitures ont été peintes de manière consécutive. La purge du matériel de peinture engendre des coûts reliés au solvant utilisé pour nettoyer les équipements. Il est donc avantageux de regrouper les voitures de même couleur ensemble pour ainsi minimiser les changements de teinte, sans toutefois dépasser le maximum permis.

Dans l'atelier de montage, on distingue deux types de contraintes de capacité: les *prioritaires* et les *non-prioritaires*. Cette distinction vient du fait que l'on possède des moyens de contrer les effets des violations sur certaines contraintes, ce qui les rend moins critiques, et donc non-prioritaires. Le respect de

ces contraintes peut être considéré comme une préférence, contrairement à celui des contraintes prioritaires qui est une nécessité.

Un véhicule appartient à la fois à une classe prioritaire et à une classe non-prioritaire. Ces classes sont formées, comme pour le POV théorique, en regroupant les voitures qui demandent les mêmes options pour chaque type.

La comptabilisation des conflits se fait de la même manière pour les deux types de contraintes de capacité, mais est différente de celle du POV théorique. Sommairement, cette évaluation favorise une plus grande distance entre les voitures en conflit.

Le POV+ est donc un problème multi-objectifs, où 3 objectifs sont à minimiser et pour lesquels aucun compromis n'est possible : un ordre de priorité est établi et aucune détérioration d'un objectif n'est acceptée pour une amélioration sur un objectif d'ordre inférieur. Ces objectifs correspondent au nombre de changements de teinte, au nombre de conflits sur les contraintes prioritaires et au nombre de conflits sur les contraintes non-prioritaires. Le carnet de commande comprend les compositions en options prioritaires et non prioritaires et la couleur de chaque véhicule. Il arrive que chacun des objectifs pris séparément soit facile à résoudre. La difficulté du problème réside, entre autres, dans la combinaison de ces trois objectifs qui sont souvent contradictoires.

Soulignons le travail de Gagné, Gravel et Price [GAG'05] qui ont présenté un algorithme d'OCF pour la résolution d'instances du problème fournies par le *Groupe Renault*, dont les tailles atteignent 1260 voitures. Les auteurs présentent

d'abord le développement d'un modèle de programmation linéaire en nombres entiers (*Integer Linear Programming*) qui, bien qu'il ait permis de déterminer la satisfaisabilité de certaines instances, s'avère être limité par les outils de résolutions actuels pour les instances de très grande taille. Les auteurs proposent donc une approche heuristique multi-objectifs sans compensation. Ils reprennent en fait un algorithme publié plus tôt pour le POV théorique [GRA'05], qui s'était montré très efficace et qu'ils adaptent à la résolution du POV+. L'algorithme bonifié, entre autres, d'une procédure de recherche locale, atteint de meilleures performances que le recuit simulé utilisé par la compagnie Renault sur toutes les instances. Ce recuit simulé, conçu en 1992 par Chew *et al.* [CHE'92], s'était montré prometteur à l'époque.

Soulignons en terminant que bon nombre de variantes du POV+ existent. Nommons par exemple les versions multi-objectifs (avec compensation entre les objectifs) [HYU'98], avec dates de remise [LOV'00] et avec affectation à des postes de travail [BUK'02].

2.5 Objectifs de la recherche

À la lumière des informations recueillies dans ce chapitre et de l'analyse de la littérature qui y est présentée, on observe que le problème d'ordonnancement de voitures est devenu une référence dans les domaines de l'optimisation combinatoire et de la résolution de problèmes de satisfaction de contraintes. Nous avons vu qu'il est possible de transférer la formulation à une version plus proche de la réalité industrielle. On remarque également que les méthodes heuristiques, dont l'optimisation par colonie de fourmis, présentent un potentiel pour l'optimisation du POV. Ce potentiel existe à la fois pour la version théorique du problème et pour une version appliquée en industrie, où la rapidité d'exécution est un avantage considérable.

Pour ces raisons, le travail de recherche qui fera l'objet de ce mémoire a pour objectif global de proposer un algorithme d'OCF efficace en termes de qualité de solution pour la résolution du POV comprenant des éléments adaptés spécifiquement pour ce dernier. Plus précisément, le travail devra rencontrer les objectifs spécifiques suivants :

1. Proposer de nouvelles structures de trace de phéromone pour un algorithme d'OCF et en évaluer les impacts sur la qualité des solutions, pour la résolution du POV.

2. Identifier des mécanismes permettant de diversifier la recherche de solutions pour un OCF et en évaluer les impacts sur la qualité des solutions, pour la résolution du POV.
3. Proposer un nouveau mode de construction des solutions dans un algorithme d'OCF afin de permettre la modification des éléments déjà fixés et en évaluer les impacts sur la qualité des solutions, pour la résolution du POV.

L'atteinte de ces objectifs se fera par la bonification d'un algorithme d'OCF déjà adapté pour la résolution du POV théorique. L'algorithme de Gravel *et al.* [GRA'05] (voir Section 2.3) servira de modèle de base et sera modifié de manière incrémentale. Des modifications y seront apportées afin de l'adapter d'avantage à la structure particulière du POV. Les versions d'algorithme proposées seront évaluées selon leur nombre de conflits moyen pour chaque instance du problème et le temps requis pour leur exécution. Les nombres de conflits moyens seront comparés aux meilleurs obtenus dans la littérature à ce jour. Étant donné l'impossibilité de comparer les temps de résolution dans la littérature entre eux et avec ceux qui seront obtenus, ces données seront utilisées à des fins informatives et ne pourront servir qu'à comparer les algorithmes de manière approximative.

Au terme de ce travail, il sera possible de conclure sur l'efficacité des nouvelles approches proposées pour le problème du POV.

2.6 Conclusion

Ce chapitre avait pour objectif l'analyse de la littérature concernant l'OCF et le POV. Les concepts clés, la formalisation et les principaux efforts de la communauté scientifique y ont été abordés.

On constate que les méta-heuristiques représentent un domaine en pleine expansion et cet intérêt leur est probablement accordé en raison de leur facilité relative d'adaptation et de leur excellent compromis qualité-temps. Nous avons vu que l'OCF a été adapté à de nombreux problèmes. Le POV a aussi été résolu par des algorithmes de fourmis et cette association s'est montrée très efficace selon les tests numériques conduits sur des ensembles d'instances considérées comme difficiles.

On note que le POV a été abordé dans différents domaines, dont les méthodes exactes de résolution de problème de satisfaction de contraintes, la programmation en nombres entiers et les méta-heuristiques. Ces dernières présentent le double avantage d'être malléables et de s'exécuter dans un temps raisonnable. Ces qualités sont à ne pas négliger étant donné le transfert potentiel à des problèmes de l'industrie. Advenant cette éventualité, la rapidité d'exécution et la possibilité d'optimiser les solutions si aucune ne peut être trouvée sans conflit seront des atouts majeurs pour les méta-heuristiques et pour l'OCF.

En terminant, on observe que les méta-heuristiques représentent une avenue intéressante pour la résolution du POV théorique, mais aussi de problèmes de POV+ rencontrés au quotidien par les producteurs automobiles. Elles fournissent

des solutions de très bonne qualité, à un coût moindre que celui des méthodes exactes.

CHAPITRE 3 :

BONIFICATION D'UN ALGORITHME D'OCF POUR LA RÉSOLUTION DU POV

3.1 Introduction

L'analyse de la littérature réalisée a mené à l'élaboration d'un travail de recherche consistant à définir une stratégie de résolution d'OCF de façon à prendre en considération la structure et les particularités du POV théorique.

Les résultats prometteurs obtenus récemment par Gravel, Gagné et Price [GRA'05] en utilisant l'OCF pour la résolution du POV ont encouragé la poursuite de travaux s'inscrivant dans la même veine. Le potentiel d'adaptabilité de l'OCF, la possibilité de généralisation du POV en un problème industriel ainsi que le faible nombre de publications unissant ces deux sujets font également partie des arguments en faveur de cette orientation.

L'algorithme d'OCF présenté par Gravel *et al.* constitue l'algorithme de départ qui sera bonifié d'éléments spécialement adaptés à la structure du POV. Le code source utilisé par les auteurs est repris, ce qui assure de meilleures conditions pour la comparaison avec les algorithmes bonifiés. La méthodologie employée consiste en la modification par incrément de cet algorithme de départ. À chaque étape du développement, un élément est modifié ou ajouté pour obtenir une nouvelle version de l'algorithme et ce changement est conservé pour les versions subséquentes. Le projet se déroule en trois phases principales : la définition d'une nouvelle structure de trace de phéromone, la variation de certains paramètres en cours d'exécution et la proposition d'un nouveau concept de construction des solutions. Afin de valider l'apport des éléments modifiés ou ajoutés, l'algorithme est exécuté sur les ensembles test 1, 2 et 3 à chaque étape de sa transformation.

Les environnements logiciel et matériel sont constants tout au long des tests pour lesquels les résultats sont présentés dans ce chapitre. Les résultats présentés pour l'algorithme de départ, ainsi que pour les différentes versions proposés, ont été obtenus sur une machine à processeur Xeon 3,06 Ghz, sous Windows XP, avec 1024 Mo de mémoire vive. Tous les algorithmes sont réalisés dans le langage C/C++, avec le compilateur Visual C++ 6.0 pour le code original et avec Visual C++ .NET pour les modifications. Chaque version de l'algorithme a été exécutée 100 fois pour chaque instance.

L'algorithme de départ (OCF-0) est détaillé à la Section 3.2, où sont expliquées toutes les modifications ayant été nécessaires à l'adaptation pour résoudre le POV. Nous verrons comment sont construites les solutions, comment est utilisée la trace de phéromone et comment la règle de transition tient compte des particularités du problème et restreint le nombre d'éléments à considérer à l'aide d'une gestion intelligente des candidats. Les détails des résultats obtenus sont également reportés. Par la suite, trois bonifications de l'algorithme sont présentées, dont chacune inclut la précédente. La structure de la trace de phéromone est d'abord modifiée à la Section 3.3 pour obtenir l'OCF-3D. À cette nouvelle version, est ajoutée une variation en cours d'exécution de certains paramètres de l'OCF. L'OCF-EV est ainsi créé et il est présenté à la Section 3.4. Finalement, un concept nouveau de construction de solution est intégré à l'algorithme à la Section 3.5 pour obtenir l'OCF-M, où les fourmis utilisent une stratégie mixte de construction. Afin de bien résumer la progression des versions

d'OCF présentées, une récapitulation des performances observées accompagnée d'une analyse de l'impact des différents éléments ajoutés est présentée à la Section 3.6.

3.2 L'algorithme de départ (OCF-0)

Pour l'article « *Review and comparison of three methods for the solution of the car-sequencing problem* », Gravel, Gagné et Price ont travaillé à l'élaboration d'un OCF spécialement adapté à la résolution du POV. Cet algorithme, appelé OCF-0 dans le présent document, est présenté à la Figure 3.1. Les sections suivantes expliquent les détails de son implantation.

```
Initialiser l'ensemble de la matrice de trace de phéromone à τ0
t = 1
TANT QUE t < NCMAX ET Sgb comporte au moins un conflit
    Donner une classe de départ à chaque fourmi en initialisant Sk(1)
    POUR y = 2 à n
        POUR k = 1 à m
            Choisir Sk(y), la classe de la voiture en position y selon
                l'Équation 3.3
            Mise à jour locale de la trace selon l'Équation 3.1
    POUR k = 1 à m
        Évaluer Lk
    Mise à jour globale de la matrice de trace selon l'Équation 3.2
    Mettre à jour Sgb, la meilleure solution trouvée jusqu'à maintenant
    t = t + 1
```

Figure 3.1 : Pseudo-code de l'OCF-0 [GRA'05]

3.2.1 Construction des solutions

Chaque fourmi k doit construire une séquence S_k, où $S_k(y)$ contient la classe de la voiture à produire en position y. La variable y^* est utilisée pour désigner la position en cours d'affectation. Les voitures sont placées de manière séquentielle,

de la position 1 à n, et sont représentées par le numéro de la classe leur correspondant.

La séquence produite doit respecter les contraintes de demande du problème et donc inclure exactement c_i voitures de chaque classe i. Pour ce faire, chaque fourmi k utilise une liste $tabou_k$ dans laquelle elle insère les classes pour lesquelles elle a placé toutes les voitures demandées au cours de sa construction. On dit qu'une classe est *disponible* pour la fourmi k si elle n'est pas élément de $tabou_k$.

La première voiture de la séquence est déterminée de manière probabiliste et peut être différente pour chaque fourmi. Chaque classe de voiture a une chance d'être sélectionnée proportionnelle au nombre d'options qui la composent. L'évaluation des solutions construites est représentée par le nombre de conflits contenu dans la séquence. Ce nombre de conflits est compté selon la méthode présentée à la Section 2.4.1 et noté L_k pour une fourmi k.

3.2.2 La trace de phéromone

La trace de phéromone est stockée dans une matrice τ symétrique de taille $v \times v$, où v est le nombre de classes de l'instance. L'élément $\tau_{i,i'}$ reflète l'intérêt de placer une voiture de classe i immédiatement avant ou après une voiture de classe i'. La diagonale de la matrice est utilisée, puisque deux voitures de même classe peuvent être placées côte à côte. Les mises à jour de la trace se font

toujours de manière symétrique. Cependant, afin d'alléger le texte et les équations présentées, cette double mise à jour n'est pas explicitement montrée. Le lecteur doit donc considérer qu'à chaque fois qu'un élément $\tau_{i,\,i'}$ est modifié, $\tau_{i',\,i}$ reçoit la même valeur.

La mise à jour locale est faite par chaque fourmi k, selon l'Équation 3.1, où $S_k(y^*\text{-}1)$ et $S_k(y^*)$ sont respectivement les classes de l'avant-dernière et de la dernière voiture placée.

$$\tau_{S_k(y^*-1),S_k(y^*)} = \rho_{local} \cdot \tau_{S_k(y^*-1),S_k(y^*)} + (1-\rho_{local}) \cdot \Delta\tau$$
$$où\ \Delta\tau = \tau_0 \tag{3.1}$$

La mise à jour globale de la trace est faite à l'aide de S_+, la meilleure solution du cycle. Elle utilise L_{gb}, le nombre de conflits de la meilleure solution globale trouvée, et L_+, le nombre de conflits de la meilleure solution du cycle. Elle est faite selon l'Équation 3.2, où $Nb\_Apparitions_{S+}$ est le nombre de fois où une voiture de classe i est adjacente à une autre de classe i' dans S_+, la séquence de la meilleure fourmi du cycle. Une augmentation de L_{gb}/L_+, représentant le ratio du nombre de conflits de la meilleure solution globale à celui de la meilleure solution du cycle, est observée chaque fois que deux classes se retrouvent côte à côte dans S_+.

$$\tau_{i,i'} = \rho_{local} \cdot \tau_{i,i'}(t) + (1 - \rho_{global}) \cdot \Delta\tau_{i,i'}$$

$$où\ \Delta\tau_{i,i'} = \begin{cases} Nb\_Apparitions_{S_+} \cdot \dfrac{L_{gb}}{L_+} & si\ S_+\ emprunte\ (i,\ i') \\ 0 & sinon \end{cases} \qquad (3.2)$$

3.2.3 La règle de transition et la gestion des candidats

La règle de transition est parmi les éléments les plus importants à considérer pour l'adaptation d'un OCF. En plus de la trace de phéromone, cette règle tient compte d'une visibilité locale, une heuristique gourmande tirée de la nature même du problème à l'étude. La règle de transition utilisée pour l'OCF-0 est calquée sur la règle pseudo-proportionnelle aléatoire et reproduite à l'Équation 3.3. Un OCF standard comporte normalement un seul terme de visibilité locale, alors que l'OCF-0 en utilise deux distincts : le nombre de nouveaux conflits ajoutés et la difficulté d'une classe. Ces concepts ainsi que l'utilisation de la trace de phéromone sont expliqués dans les paragraphes qui suivent.

$$S_k(y^*) = \begin{cases} \arg\max_{w \in Candidat_k} \left\{ \left[\tau_{S_k(y^*-1),w}\right]^{\alpha} \cdot \left[\eta_w\right]^{\beta} \cdot \left[d_w\right]^{\delta} \right\} & si \quad Q \leq Q_0, \\ \psi & sinon \end{cases}$$

où ψ est choisi selon la probabilité suivante: (3.3)

$$P^k_{S_k(y^*-1),i} = \begin{cases} \dfrac{\left[\tau_{S_k(y^*-1),i}\right]^{\alpha} \cdot \left[\eta_i\right]^{\beta} \cdot \left[d_i\right]^{\delta}}{\sum_{w \in Candidat_k} \left[\tau_{S_k(y^*-1),w}\right]^{\alpha} \cdot \left[\eta_w\right]^{\beta} \cdot \left[d_w\right]^{\delta}} & si\ i \in Candidat_k \\ 0 & sinon \end{cases}$$

Nous avons vu que la trace de phéromone est stockée dans une matrice τ de taille $v \times v$. On évalue la quantité de phéromone entre la classe de la dernière voiture placée et celle de la voiture candidate en utilisant l'élément $\tau_{S_k(y^*-1),w}$, où w est la classe candidate. Ce terme est élevé à la puissance α, comme dans l'OCF standard.

La première visibilité locale η_t est calculée selon l'Équation 3.4, où $NouvConflits_i$ est le nombre de nouveaux conflits ajoutés par l'insertion d'une voiture de classe i à la fin de la séquence en cours. On cherche ainsi à minimiser le nombre de conflits ajoutés. L'exposant de ce terme reste β. On remarque une addition de 1 au dénominateur afin d'éviter une division par 0.

$$\eta_i = \frac{1}{1 + NouvConflits_i} \tag{3.4}$$

La seconde visibilité locale ajoutée à la règle de transition est représentative de la difficulté associée à une classe et est notée d_i pour une classe i candidate. Cette mesure est obtenue par la somme des taux d'utilisation dynamiques *tu'* des options requises par la classe candidate, tel que reporté à l'Équation 3.5.

$$d_i = \sum_{j=1}^{o} a_{ij} \cdot tu'_j \qquad où \; tu'_j = \frac{b'_j \cdot p_j}{n' \cdot q_j} \tag{3.5}$$

Le taux d'utilisation dynamique tu'_j d'une option a sensiblement la même signification que son taux d'utilisation tu_j (voir l'Équation 2.11), à la différence qu'il représente le taux d'utilisation dans la partie restante de la séquence. Il est donc modifié à chaque pas et propre à chaque fourmi. Il est calculé à l'aide de b'_j, le nombre de voitures requérant l'option j encore à placer, et de n', la taille de la sous-séquence disponible $\left(n' = \left(n - y^* + 1\right)\right)$.

Ce deuxième terme de visibilité a pour objectif de favoriser l'ordonnancement des classes de voitures les plus difficiles le plus tôt possible. Si une classe de voitures est difficile à placer considérant l'espace restant, cette difficulté ne peut qu'augmenter si elle n'est pas placée. On tente ainsi d'éviter de se retrouver en fin de solution avec les voitures les plus contraintes. À ce terme est également associé un exposant, noté δ.

La gestion de la liste des classes candidates est une particularité importante de l'OCF-0. Une fourmi choisit parmi les classes qui ne génèrent pas de nouveaux

conflits aussi souvent qu'elle le peut. Lorsqu'une fourmi k doit placer une voiture, elle évalue $NouvConflits_i \ \forall \ i \notin \ tabou_k$. S'il existe une ou plusieurs classes qui n'engendrent pas de nouveaux conflits, seules ces dernières seront insérées dans $Candidat_k$. De plus, s'il existe une classe « facile » ne requérant aucune option, celle-ci est retirée de $Candidat_k$, à moins qu'elle ne soit la seule classe candidate. De cette façon, les voitures de cette classe, qui sont faciles à placer car elles ne peuvent générer de nouveaux conflits, sont réservées pour les cas où toutes les autres classes génèrent de nouveaux conflits.

3.2.4 Résultats de l'OCF-0

Les résultats obtenus par l'OCF-0 sont très encourageants lorsque comparés à ceux d'autres algorithmes proposés dans la littérature. Ils serviront de base de comparaison pour les versions bonifiées de l'algorithme présentées pour ce travail.

Les paramètres de l'OCF ont été fixés par suite d'essais numériques aux valeurs suivantes : $\{\alpha,\beta,\delta\}=\{1,6,3\}$, $\tau_0=0,005$, $\rho_{local}=\rho_{global}=0,99$, $m=15$, $Q_0=0,9$, $NC_{MAX}=1000$. Chaque résultat est calculé sur un ensemble de 100 résolutions.

Les Tableaux 3.1, 3.2 et 3.3 résument les performances atteintes par l'OCF-0 pour l'ET1, l'ET2 et l'ET3. Les tableaux de résultats présentés dans ce chapitre ont tous la même structure et présentent les colonnes suivantes : « Instance » : nom de chaque instance ; « Meilleure solution connue » : nombre de conflits de la

meilleure solution connue ; « Nombre de conflits moyen » : nombre de conflits moyen obtenu pour l'ensemble des résolutions ; « Cycle de sortie moyen » : cycle moyen auquel la solution finale d'une résolution a été trouvée ; « Temps de calcul moyen (sec) » : temps moyen en secondes nécessaire à une résolution, excluant les étapes de lecture/écriture dans les fichiers et les phases d'initialisation. Les tableaux pour l'ET2 et l'ET3 présentent en plus : « Écart-type » : écart-type du nombre de conflits pour l'ensemble des résolutions ; « MIN » et « MAX » : nombres de conflits minimum et maximum obtenus parmi les 100 résolutions. Notons que pour l'ET1, les instances sont regroupées par groupe de 10 en fonction de leur niveau de difficulté et que les statistiques sont calculées sur l'ensemble des résolutions de chaque groupe.

Groupe d'instances	Meilleure solution connue	OCF-0		
		Nombre de conflits moyen	Cycle de sortie moyen	Temps de calcul moyen (sec)
60-*	0	0,00	1,13	0,02
65-*	0	0,00	1,19	0,02
70-*	0	0,00	1,28	0,02
75-*	0	0,00	1,43	0,02
80-*	0	0,00	1,61	0,03
85-*	0	0,00	1,93	0,04
90-*	0	0,00	1,90	0,03

Tableau 3.1 : Résultats obtenus par l'OCF-0 pour l'ET1 [GRA'05]

Instance	Meilleure solution connue	OCF-0					
		Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
10_93	3	4,24	0,47	3	5	298,93	10,10
16_81	0	0,12	0,33	0	1	329,10	4,23
19_71	2	2,08	0,27	2	3	309,48	9,26
21_90	2	2,63	0,49	2	3	324,39	8,73
26_82	0	0,00	0,00	0	0	83,72	0,74
36_92	2	2,29	0,46	2	3	346,02	8,50
4_72	0	0,00	0,00	0	0	58,72	0,50
41_66	0	0,00	0,00	0	0	5,84	0,04
6_76	6	6,00	0,00	6	6	1,01	7,94

Tableau 3.2 : Résultats obtenus par l'OCF-0 pour l'ET2 [GRA'05]

Instance	Meilleure solution connue	OCF-0					
		Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
200_01	0	3,80	0,62	2	5	293,44	19,46
200_02	2	4,14	0,51	3	5	422,62	19,42
200_03	4	8,90	0,59	8	10	350,44	19,70
200_04	7	9,86	0,51	8	11	274,59	20,49
200_05	6	8,81	0,42	8	10	201,67	18,28
200_06	6	6,87	0,39	6	8	232,09	19,03
200_07	0	2,99	0,36	2	4	344,48	18,95
200_08	8	8,00	0,00	8	8	51,06	17,31
200_09	10	11,85	0,48	11	13	284,31	19,93
200_10	19	21,44	0,57	20	23	313,96	18,65
300_01	0	5,33	0,71	4	7	386,80	30,23
300_02	12	13,15	0,39	13	15	307,19	30,68
300_03	13	14,54	0,54	13	16	370,63	32,26
300_04	7	10,33	0,68	9	12	390,43	30,25
300_05	29	40,55	1,06	36	43	371,53	29,99
300_06	2	7,59	0,74	6	9	255,48	30,21
300_07	0	2,89	0,63	1	4	173,35	30,07
300_08	8	9,17	0,38	9	10	229,22	29,00
300_09	7	9,05	0,56	8	10	274,34	28,76
300_10	21	34,63	1,04	32	37	237,32	28,57
400_01	1	3,01	0,56	2	4	201,01	41,77
400_02	16	23,28	0,82	21	25	280,48	38,95
400_03	9	11,65	0,50	10	12	212,39	38,40
400_04	19	21,96	0,75	20	24	454,46	43,57
400_05	0	3,48	0,96	1	6	346,67	36,47
400_06	0	4,20	0,97	1	6	263,12	39,85
400_07	4	7,65	0,83	5	9	260,60	36,93
400_08	4	11,54	1,62	6	14	179,93	35,51
400_09	5	17,98	1,10	15	20	209,19	41,38
400_10	0	4,24	0,99	2	7	77,19	40,94

Tableau 3.3 : Résultats obtenus par l'OCF-0 pour l'ET3 [GRA'05]

Notons que les cycles de sortie moyens pour l'ET1 et les écarts-types pour l'ET2 ont été obtenus directement des auteurs et ne figurent pas dans l'article publié. Il en va de même pour les résultats de l'ET3 qui avaient été obtenus à l'aide

d'un OCF avec recherche locale. L'obtention des résultats sans recherche locale permettra de meilleures comparaisons avec les algorithmes présentés. Les temps de calcul pour tous les ensembles sont donnés à titre indicatif puisque les environnements de compilation ne sont pas exactement les mêmes et que des accélérations ont été apportées au code original.

3.3 Nouvelle structure de trace de phéromone

L'étude de la nature du problème d'ordonnancement de voitures a inspiré une modification importante de la structure même de la trace de phéromone et de son utilisation. Cette section présente le cheminement de cette première bonification. Dans un premier temps, une version préliminaire est présentée, où seule l'utilisation de la trace est modifiée. Une seconde version est ensuite obtenue, où la matrice de trace de phéromone est modifiée pour tenir compte des particularités du POV.

3.3.1 Phase préliminaire: Trace sur un horizon (OCF-H)

La trace de phéromone dans l'OCF-0 est structurée de manière à donner l'intérêt de placer deux classes de voitures l'une à côté de l'autre. Cependant, en raison des contraintes qui s'étendent sur des blocs de voitures, une voiture a une influence sur l'évaluation de la solution en fonction, non seulement des deux voitures adjacentes, mais aussi de ses voisines plus éloignées. Comme une voiture peut participer à un conflit dans chaque bloc d'au plus p_{max} positions duquel elle fait partie, une voiture aura une influence sur les voitures situées aux p_{max} positions qui la précèdent et la suivent.

Ce sont ces déductions qui ont mené à la première modification de la trace de phéromone : son utilisation sur un horizon de p_{max} positions. La structure de la matrice de trace de phéromone reste la même, seule son utilisation est modifiée.

On cherche maintenant le bénéfice de placer une classe de voitures en considérant son interaction avec les classes situées à proximité (à au plus p_{max} positions de distance).

L'algorithme résultant de l'OCF-0 avec utilisation de la trace de phéromone sur un horizon de p_{max} est appelé OCF-H. Les modifications apportées à la trace sont expliquées dans les paragraphes qui suivent. Elles touchent l'utilisation de la trace de phéromone dans la règle de transition, la mise à jour locale et la mise à jour globale.

L'utilisation de la trace de phéromone pour le calcul de la règle de transition est un point important de cette modification. Afin de refléter l'intérêt de placer une classe candidate à proximité des dernières voitures de la séquence, on utilise maintenant une somme des traces de phéromone accumulées entre cette classe candidate et les classes des p_{max} dernières voitures. Cette somme est illustrée à la Figure 3.2, où on a pris pour exemple l'instance 60_02 du Tableau 2.2 avec $p_{max} = 5$. Dans cet exemple, on cherche l'intérêt de placer une voiture de classe i à la fin de la séquence en cours (en position y*). Les positions prises en considération sont encadrées de pointillés. Les éléments associés aux voitures encadrées sont additionnés, tel que montré en (a), pour obtenir la somme générale en (b).

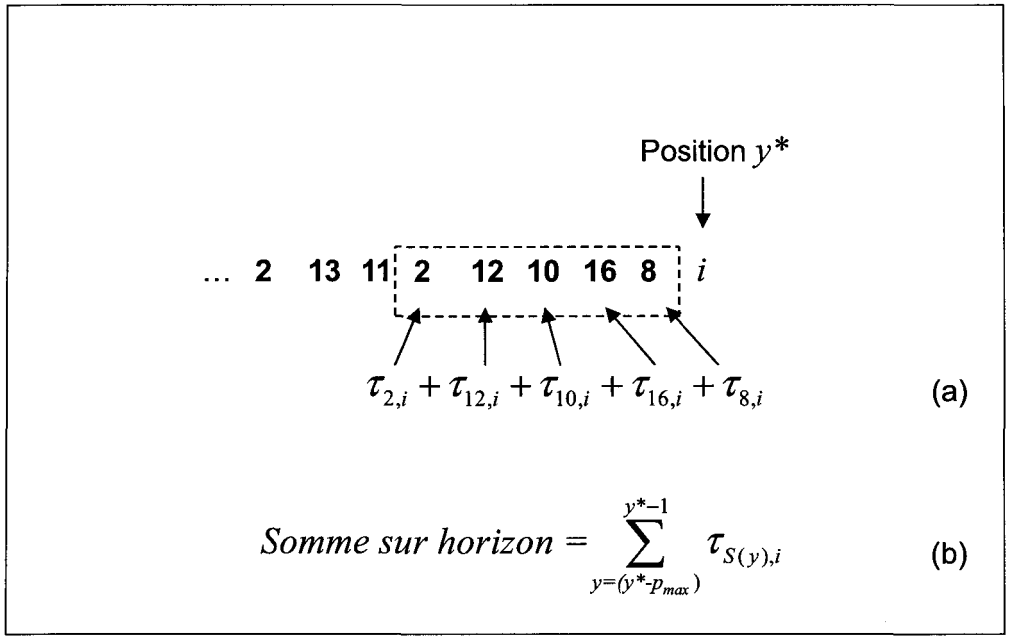

Figure 3.2 : Illustration de la somme de trace de phéromone utilisée dans la règle de transition de l'OCF-H

Une fois la classe de voitures choisie à l'aide de la règle de transition, la mise à jour locale est effectuée entre la classe de la voiture placée en position y^* et les classes des p_{max} voitures précédentes. La mise à jour locale implique donc les mêmes voitures que la sommation sur un horizon calculée pour la règle de transition, pour un total de p_{max} mises à jour.

À la fin d'un cycle, la meilleure solution S_+ est utilisée pour faire la mise à jour globale de la trace de phéromone. Cette mise à jour globale se fait entre une classe de voitures à une position y donnée et chacune des classes des p_{max} positions y' suivantes, tel que reproduit à la Figure 3.3. Notons que la mise à jour

de la diagonale est possible lorsque deux voitures de même classe se retrouvent dans un même horizon.

Évaporation de l'ensemble de la matrice de trace selon ρ_{global}
POUR chaque position y de la séquence S_+
  POUR chaque position y' entre $(y+1)$ et $(y+p_{\max})$

$$\tau_{S_+(y),s_+(y')} = \tau_{S_+(y),s_+(y')} + {L_{gb}}/{L_+}$$

Figure 3.3 : Mise à jour globale pour l'OCF-H

Pour assurer la validité de la comparaison entre les algorithmes, les valeurs des paramètres de l'OCF-H sont les mêmes que pour l'OCF-0. Le Tableau 3.4 montre les résultats obtenus pour l'ET1, où les nombres de conflits moyens de la version précédente (l'OCF-0) ont été reportés pour faciliter la comparaison. Ces moyennes indiquent une performance en tous points équivalente à l'OCF-0, à l'exception des temps de calcul qui sont un peu plus élevés pour l'OCF-H. Ce dernier parvient à trouver une solution optimale pour chacune des 100 résolutions de chaque instance, en 4 cycles ou moins dans 90,8 % des résolutions.

Groupe d'instances	Meilleure solution connue	OCF-0	OCF-H		
		Nombre de conflits moyen	Nombre de conflits moyen	Cycle de sortie moyen	Temps de calcul moyen (sec)
60-*	0	0,00	0,00	1,38	0,04
65-*	0	0,00	0,00	1,48	0,04
70-*	0	0,00	0,00	1,79	0,05
75-*	0	0,00	0,00	2,34	0,06
80-*	0	0,00	0,00	3,23	0,09
85-*	0	0,00	0,00	3,82	0,11
90-*	0	0,00	0,00	10,42	0,32

Tableau 3.4 : Résultats obtenus par l'OCF-H pour l'ET1

Les résultats donnés par l'OCF-H pour l'ET2 sont reportés au Tableau 3.5. À l'exception des instances 41_66 et 6_76, pour lesquelles l'OCF-H est équivalente à l'OCF-0, on observe des nombres de conflits moyens plus élevés pour l'OCF-H. Ces augmentations sont de l'ordre 13% à 683%. Ces pourcentages sont calculés par la différence entre le nombre de conflits moyen de l'OCF-H et celui de l'OCF-0, divisé par le nombre de conflits moyen de l'OCF-0. Toutefois, l'OCF-H est en mesure d'atteindre au moins une fois sur 100 résolutions une solution équivalente à la meilleure connue pour toutes les instances, à l'exception de l'instance 10_93. Le cycle de sortie moyen est d'au plus 356 cycles, soit 35,6% des cycles disponibles, ce qui peut indiquer que les cycles restants ne sont pas souvent utiles à l'algorithme. Finalement, bien qu'enregistrés dans des conditions différentes, les temps de calcul requis sont comparables à ceux de l'OCF-0, mis à part pour les instances pour lesquelles la meilleure solution connue ne comporte pas de conflits où l'OCF-H est généralement plus long à exécuter. Cela s'explique facilement par le fait qu'une fois qu'une solution sans conflit est trouvée, les algorithmes arrêtent

l'exécution en cours puisqu'il est impossible d'améliorer la solution. Comme l'OCF-0 obtient plus souvent des solutions optimales pour ces instances, il est logique que son temps de calcul moyen soit moindre. En résumé, l'ajout des opérations nécessaires pour passer de l'OCF-0 à l'OCF-H, n'engendre aucune augmentation notable du temps de calcul.

		OCF-0	OCF-H					
Instance	Meilleure solution connue	Nombre de conflits moyen	Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
10_93	3	4,20	5,33	0,55	4	6	306,20	10,52
16_81	0	0,10	0,94	0,31	0	2	280,04	10,08
19_71	2	2,10	2,93	0,26	2	3	209,82	9,43
21_90	2	2,60	2,98	0,14	2	3	183,08	9,43
26_82	0	0,00	0,86	0,35	0	1	84,15	8,46
36_92	2	2,30	3,35	0,52	2	4	357,92	9,26
4_72	0	0,00	0,09	0,29	0	1	279,40	3,22
41_66	0	0,00	0,00	0,00	0	0	12,62	0,10
6_76	6	6,00	6,00	0,00	6	6	1,01	8,43

Tableau 3.5 : Résultats obtenus par l'OCF-H pour l'ET2

Les résultats de l'OCF-H pour l'ET3 sont présentés au Tableau 3.6. Les nombres de conflits moyens sont supérieurs à ceux de l'OCF-0 dans tous les cas, avec des pourcentages d'augmentation entre 0,1% et 158%. On obtient cependant, pour 19 des 30 instances, des pourcentages de dégradation par rapport aux moyennes de l'OCF-0 inférieurs à 50%. L'OCF-H parvient à trouver une solution équivalente à la meilleure connue dans le seul cas de l'instance 200_08.

Instance	Meilleure solution connue	OCF-0	OCF-H					
		Nombre de conflits moyen	Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
200_01	0	3,80	8,38	0,86	6	10	318,24	29,00
200_02	2	4,14	4,85	0,36	4	5	205,64	27,51
200_03	4	8,90	12,36	0,73	11	14	407,52	28,59
200_04	7	9,86	12,38	0,63	10	13	274,95	30,76
200_05	6	8,81	10,10	0,46	9	11	315,93	26,16
200_06	6	6,87	8,15	0,54	7	9	342,64	27,16
200_07	0	2,99	7,18	0,80	5	8	273,16	27,54
200_08	8	8,00	8,01	0,10	8	9	230,78	25,04
200_09	10	11,85	12,73	0,45	12	13	260,54	28,33
200_10	19	21,44	25,48	0,73	23	27	364,98	25,87
300_01	0	5,33	8,06	0,76	6	9	249,88	62,87
300_02	12	13,15	17,76	0,92	15	19	294,33	62,34
300_03	13	14,54	20,50	0,93	17	22	295,21	63,72
300_04	7	10,33	12,49	0,73	11	14	355,72	55,88
300_05	29	40,55	48,83	1,04	45	51	387,18	58,94
300_06	2	7,59	11,05	0,73	9	12	364,71	59,51
300_07	0	2,89	7,45	0,90	6	9	211,30	55,05
300_08	8	9,17	14,48	1,37	11	17	72,87	55,22
300_09	7	9,05	11,19	0,81	9	13	296,16	53,42
300_10	21	34,63	38,29	1,11	35	41	361,65	54,52
400_01	1	3,01	4,44	0,56	3	5	294,49	88,62
400_02	16	23,28	27,30	1,14	25	29	231,53	87,44
400_03	9	11,65	12,56	0,50	12	13	285,17	86,76
400_04	19	21,96	28,82	1,22	25	31	431,30	99,00
400_05	0	3,48	4,93	0,95	2	7	393,91	93,63
400_06	0	4,20	6,44	0,92	3	8	462,10	87,02
400_07	4	7,65	8,48	0,83	6	10	406,51	77,69
400_08	4	11,54	12,42	1,16	10	15	153,29	81,93
400_09	5	17,98	23,68	1,38	19	26	316,61	90,85
400_10	0	4,24	7,83	0,77	6	9	375,88	91,36

Tableau 3.6 : Résultats obtenus par l'OCF-H pour l'ET3

En conclusion de cette première phase de modification de la trace de phéromone, l'OCF-H produit des résultats définitivement inférieurs à ceux de

l'OCF-0, mais encourageants lorsque comparés à ceux publiés dans la littérature et présentés au Tableau 2.6. Avec des cycles de sortie moyens relativement peu élevés, il est possible d'améliorer les performances de cet algorithme préliminaire.

L'idée d'utiliser la trace de phéromone sur un horizon présente cependant un intérêt. Cette première phase de modification de la trace de phéromone ouvre la porte à une nouvelle structure de trace proposée à la section suivante.

3.3.2 Modification de la structure de la trace : trace à trois dimensions (OCF-3D)

Rappelons que selon la première modification de la trace, la somme de trace utilisée pour la règle de transition représente l'intérêt de placer une voiture de classe candidate i en fonction des p_{max} dernières voitures de la séquence. Cependant, il peut être bénéfique de placer deux voitures de classes i et i' à une distance de quatre positions l'une de l'autre, alors qu'une distance de deux positions peut être très désavantageuse. L'information utilisée par l'OCF-H est donc imprécise puisqu'elle correspond à une somme d'éléments confondus. Cette hypothèse est confirmée lorsque l'on tente de modifier la valeur de l'exposant α, car la détérioration des performances est proportionnelle à l'augmentation de la valeur du paramètre. Pour la seconde modification de la trace de phéromone, on cherche donc à distinguer l'information contenue dans la matrice de trace.

Pour ce faire, une dimension est ajoutée à la matrice de trace de phéromone afin de distribuer la quantité de phéromone $\tau_{i,i'}$ sur chacune des p_{max} distances

possibles. La trace de phéromone est donc contenue dans une matrice tridimensionnelle τ' de taille $v \times v \times p_{max}$, où $\tau'_{i,i',Dist}$ représente l'intérêt de placer des voitures de classes i et i' à une distance de $Dist$ positions l'une de l'autre. L'utilisation de la trace et les mises à jour locale et globale se font selon le même principe, sur un horizon de p_{max}, à la différence que les calculs tiennent compte de la distance entre les voitures. Cette nouvelle version est appelée OCF-3D et expliquée dans les paragraphes suivants.

La somme de trace utilisée pour le calcul de la règle de transition est modifiée par rapport à l'OCF-H. La Figure 3.4 illustre ces modifications pour un exemple où l'on cherche à mesurer l'intérêt de placer en position y^* une voiture de classe i. Notons l'ajout de la troisième dimension pour la matrice τ', dont l'indice correspond à la distance entre la position courante y^* et les p_{max} dernières voitures. On remarque l'indice ajouté en (a) et en (b), où ce même indice de la matrice est obtenu par $y^* - y$.

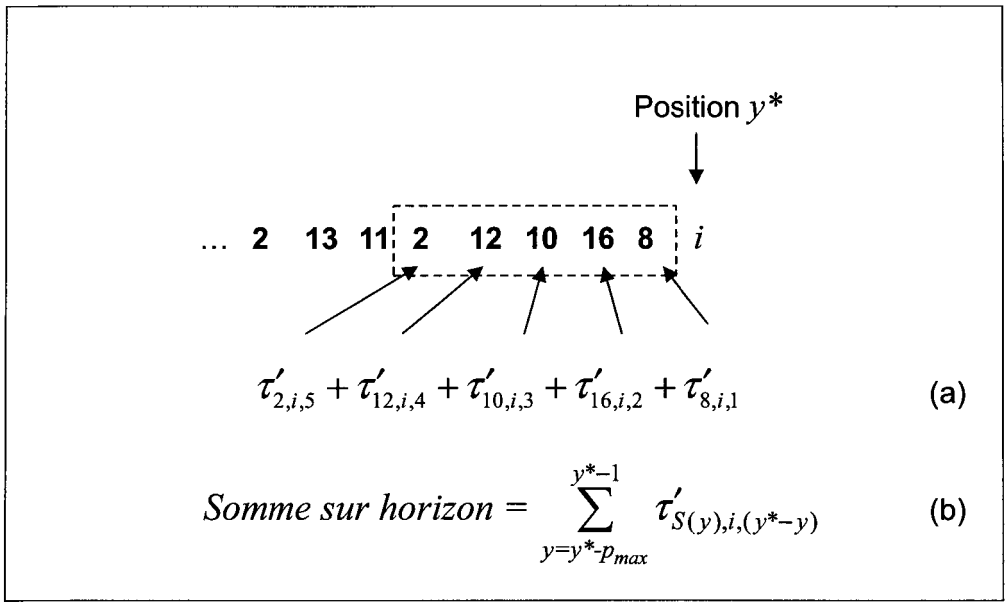

Figure 3.4 : Illustration de la somme de trace de phéromone utilisée dans la règle de transition de l'OCF-3D

La mise à jour locale est effectuée selon le même principe, en ajoutant l'information sur la distance. Elle se fait, comme pour l'OCF-H, sur les p_{max} éléments de τ' correspondants, tel que montré à la Figure 3.5.

POUR chaque position y entre (y^*-p_{max}) et (y^*-1)

$$\tau'_{S_k(y^*),S_k(y),y^*-y} = \rho_{local} \cdot \tau'_{S_k(y^*),S_k(y),y^*-y} + (1-\rho_{local}) \cdot \tau'_{S_k(y^*),S_k(y),y^*-y}$$

Figure 3.5 : Mise à jour locale pour l'OCF-3D

Suivant cette même logique, la mise à jour globale de la trace de phéromone est faite à la fin de chaque cycle, par la meilleure solution S_+, et sur les éléments de τ' aux indices $S_+(y)$, $S_+(y')$ et en fonction de la distance $y'-y$. La Figure 3.6

reflète ce changement, identifié simplement par l'ajout de la troisième dimension à l'élément de τ' mis à jour.

Évaporation de l'ensemble de la matrice de trace selon ρ_{global}
POUR chaque position y de la séquence S_+
  POUR chaque position y' entre $(y+1)$ et $(y+p_{\max})$

$$\tau'_{S_+(y),S_+(y'),y'-y} = \tau'_{S_+(y),S_+(y'),y'-y} + (1-\rho_{global}) + L_{gb}/L_+$$

Figure 3.6 : Mise à jour globale pour l'OCF-3D

Afin de pouvoir mieux observer l'impact de la modification de la structure de la trace de phéromone, la valeur de α est fixée à 4 pour l'OCF-3D. Les valeurs des autres paramètres restent inchangées. Les résultats obtenus par l'OCF-3D pour l'ET1 sont reportés au Tableau 3.7. Étant donné les conclusions concernant les performances de l'OCF-H, l'OCF-3D sera comparé ici à l'OCF-0 afin d'avoir une meilleure idée des gains apportés par la nouvelle structure de la trace. Le lecteur peut toutefois se référer à un récapitulatif à la Section 3.6 pour comparer chaque version d'algorithme qui sera proposé.

On remarque que les performances de l'OCF-3D sont inchangées pour l'ET1 par rapport à l'OCF-0. Le nombre de cycles nécessaires à l'obtention d'une solution sans conflit est négligeable et égal à 1, dans la majorité des cas. Les temps de calcul sont équivalents à ceux de l'OCF-0, ce qui permet de dire que l'ajout d'une troisième dimension à la matrice de trace n'engendre pas de hausse notable.

Groupe d'instances	Meilleure solution connue	OCF-0 Nombre de conflits moyen	OCF-3D Nombre de conflits moyen	OCF-3D Cycle de sortie moyen	OCF-3D Temps de calcul moyen (sec)
60-*	0	0,00	0,00	1,32	0,01
65-*	0	0,00	0,00	1,46	0,02
70-*	0	0,00	0,00	1,66	0,02
75-*	0	0,00	0,00	2,14	0,03
80-*	0	0,00	0,00	2,24	0,03
85-*	0	0,00	0,00	2,18	0,03
90-*	0	0,00	0,00	1,78	0,03

Tableau 3.7 : Résultats obtenus par l'OCF-3D pour l'ET1

Pour l'ET2, les résultats sont donnés au Tableau 3.8. On observe une baisse ou une égalité des nombres de conflits moyens pour chacune des instances par rapport à l'OCF-0, à l'exception du 16_81. Ces diminutions, identifiées en gras dans le tableau, sont de l'ordre de 2% à 23%. On peut affirmer que les résultats de l'OCF-3D sont peu variables, puisqu'il obtient des écarts-types entre 0 et 0,74. L'algorithme parvient à obtenir un nombre minimum de conflits équivalent à la meilleure solution connue pour l'ensemble des instances, mais les nombres de conflits maximum sont légèrement plus élevés que ceux de l'OCF-0. Les cycles de sortie moyens sont légèrement inférieurs à ceux de l'OCF-0 et relativement peu élevés. Les temps de calcul sont généralement moins élevés que ceux de l'OCF-0.

Instance	Meilleure solution connue	OCF-0	OCF-3D					
		Nombre de conflits moyen	Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
10_93	3	4,24	**4,03**	0,74	3	7	284,78	7,00
16_81	0	0,12	0,58	0,59	0	2	234,01	4,76
19_71	2	2,08	**2,04**	0,20	2	3	217,74	6,55
21_90	2	2,63	**2,02**	0,14	2	3	179,90	6,35
26_82	0	0,00	**0,00**	0,00	0	0	40,68	0,27
36_92	2	2,29	**2,03**	0,17	2	3	139,84	6,22
4_72	0	0,00	**0,01**	0,10	0	1	69,15	0,48
41_66	0	0,00	0,00	0,00	0	0	4,34	0,03
6_76	6	6,00	6,00	0,00	6	6	1,00	5,62

Tableau 3.8 : Résultats obtenus par l'OCF-3D pour l'ET2

La tendance à la baisse des nombres de conflits moyens par rapport à l'OCF-0 se confirme pour la majorité des instances de l'ET3 et ces diminutions sont identifiées en gras dans le Tableau 3.9. Les baisses des nombres de conflits moyens se situent entre 10% et 80%. Avec des écarts-types entre 0 et 1,5, on peut confirmer la faible variabilité de l'algorithme. Les valeurs minimale et maximale obtenues sont diminuées ou maintenues dans tous les cas, à l'exception de faibles augmentations pour les valeurs minimales et maximales des instances 300_05, 400_03, 400_07 et 400_08, ainsi que pour la valeur minimale du 400_02 et celle maximale du 400_01. Comme pour l'ET2, les cycles de sortie moyens sont comparables pour la grande majorité des instances.

Instance	Meilleure solution connue	OCF-0	OCF-3D					
		Nombre de conflits moyen	Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
200_01	0	3,80	**2,00**	0,62	1	3	394,76	13,69
200_02	2	4,14	**2,38**	0,49	2	3	332,53	13,78
200_03	4	8,90	**7,45**	0,59	6	9	370,86	13,87
200_04	7	9,86	**7,87**	0,34	7	8	196,39	14,58
200_05	6	8,81	**7,29**	0,46	7	8	313,68	12,95
200_06	6	6,87	**6,03**	0,17	6	7	261,79	13,13
200_07	0	2,99	**0,67**	0,53	0	2	422,98	10,90
200_08	8	8,00	8,00	0,00	8	8	21,91	11,98
200_09	10	11,85	**10,97**	0,17	10	11	95,00	13,91
200_10	19	21,44	**20,19**	0,61	19	21	341,63	12,64
300_01	0	5,33	**3,89**	0,57	3	5	392,36	23,38
300_02	12	13,15	**12,57**	0,50	12	13	230,48	24,08
300_03	13	14,54	**13,85**	0,36	13	14	213,29	24,09
300_04	7	10,33	**8,69**	0,51	7	9	327,08	21,75
300_05	29	40,55	42,54	1,07	40	44	406,08	20,84
300_06	2	7,59	**5,79**	0,56	5	7	406,44	23,03
300_07	0	2,89	**0,97**	0,50	0	2	417,38	19,91
300_08	8	9,17	**8,95**	0,22	8	9	105,82	20,53
300_09	7	9,05	**8,00**	0,20	7	9	297,19	20,04
300_10	21	34,63	**32,56**	1,15	29	35	427,52	19,74
400_01	1	3,01	3,50	0,64	2	5	280,90	28,71
400_02	16	23,28	23,82	0,81	22	25	360,76	27,89
400_03	9	11,65	**13,64**	0,64	12	15	269,08	27,71
400_04	19	21,96	**20,38**	0,51	19	21	343,25	31,05
400_05	0	3,48	**2,68**	0,76	1	5	485,32	28,41
400_06	0	4,20	**1,53**	0,59	0	3	348,61	27,40
400_07	4	7,65	8,68	0,94	6	10	346,20	26,60
400_08	4	11,54	12,67	1,50	8	16	83,66	25,71
400_09	5	17,98	**16,01**	1,17	13	18	430,45	29,36
400_10	0	4,24	**2,66**	0,67	1	4	425,60	28,33

Tableau 3.9 : Résultats obtenus par l'OCF-3D pour l'ET3

En conclusion, il est clair que l'ajout d'une troisième dimension à la matrice de trace est bénéfique pour la résolution des instances test. L'OCF-3D améliore

considérablement ou maintient les performances atteintes par l'OCF-H. On peut également déjà conclure à une amélioration de la performance par rapport à l'OCF-0 pour la majorité des instances, et c'est pourquoi l'OCF-3D est retenu pour la suite de l'analyse. La répartition de l'information de la trace de phéromone en fonction des distances entre les classes de voitures facilite la construction de solutions où se répètent les motifs essentiels à un ordonnancement efficace pour certaines options. Par exemple, pour une option j avec $tu_j = 1$ et $p_j / q_j = 1/2$, on sait que le motif idéalement à répéter est de placer exactement une voiture sur deux requérant j dans la séquence pour éviter les conflits. La matrice de trace à trois dimensions favorise d'elle-même ces motifs grâce aux mises à jour de la trace et tente d'éliminer les combinaisons de classes à des distances défavorables.

3.4 Variation des paramètres en cours d'exécution (OCF-EV)

Établir les valeurs des paramètres d'exécution représente une tâche ardue pour de nombreuses méta-heuristiques. Bien que le choix de ces valeurs soit directement relié à la performance générale de l'algorithme, il n'existe, pour ainsi dire, aucune méthode rapide et/ou optimale pour y arriver. Les auteurs ont parfois recours à d'autres méta-heuristiques dont le problème à résoudre est de trouver la meilleure combinaison possible de paramètres. C'est le cas de Botee et Bonabeau, qui ont travaillé sur un réglage de l'ensemble des paramètres d'un OCF à l'aide d'un algorithme génétique qui a donné des résultats prometteurs [BOT'98]. Mais le plus souvent, les auteurs procèdent à des essais numériques et à des analyses de sensibilité. L'une des faiblesses de cette dernière méthode est qu'il est pratiquement impossible de tester chacune des combinaisons de valeurs.

Afin de pallier à ce problème, des auteurs ont proposé des méthodes où les valeurs des paramètres sont modifiées en cours d'exécution. Battiti et Tecchiolli [BAT'94] ont présenté le *Reactive Tabu Search* dans lequel la longueur de la liste tabou est modifiée dynamiquement en fonction du comportement de la recherche. Vu son efficacité, cette méthode a été appliquée à de nombreux problèmes par la suite.

Les paramètres les plus sensibles pour l'OCF sont probablement les exposants utilisés pour la règle de transition. Leur rôle est de balancer l'importance relative entre la trace de phéromone et la visibilité locale. Le paradoxe entre l'importance des valeurs de ces paramètres et la difficulté de les établir de manière

efficace a mené à l'ajout de paramètres variables en cours d'exécution de l'algorithme. Cette décision s'appuie également sur le fait observé qu'aucune combinaison de ces paramètres n'est la meilleure sur toutes les instances. En poussant certains paramètres à des valeurs extrêmes, on améliore la performance sur des instances particulières, alors qu'on perd en qualité sur d'autres. De plus, il est logique de penser qu'en variant les combinaisons de valeurs des exposants, on aide à l'exploration en amenant l'algorithme à mettre l'accent sur différents éléments au cours de son exécution. Cette variation dans le comportement de l'algorithme pourrait contribuer à atténuer la convergence rapide observée pour l'OCF-3D, où les cycles de sortie moyens sont relativement bas.

Les paragraphes suivants expliquent de quelle manière on a rendu variables les exposants α, β et δ de la règle de transition (Équation 3.3) pour créer l'OCF-EV. La Figure 3.7 montre l'évolution des valeurs des trois exposants sur des échelles indépendantes, en fonction des cycles complétés.

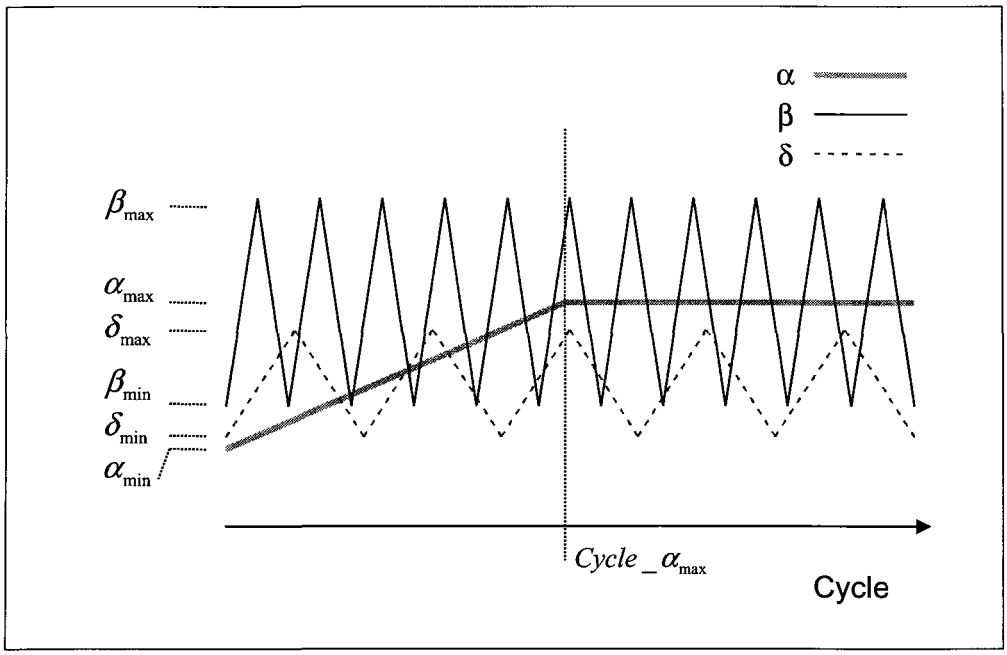

Figure 3.7 : Illustration de la variation générique des exposants de l'OCF-EV en fonction du cycle

On observe la progression constante de la valeur de α de sa valeur minimale à sa valeur maximale dans l'intervalle des cycles 0 à $Cycle\_\alpha_{max}$, puis sa constance pour les cycles suivants. On sait qu'en début de résolution, la trace de phéromone est plus ou moins intéressante étant donné qu'elle doit refléter l'apprentissage. En effet, toutes ses valeurs sont les mêmes au départ de l'algorithme et l'apprentissage augmente avec les cycles complétés. Ceci explique la forme de l'augmentation donnée à α. Deux valeurs limites $\alpha_{\min}$ et $\alpha_{\max}$ sont établies et bornent les valeurs que peut prendre α. On détermine aussi un cycle $Cycle\_\alpha_{max}$ auquel α atteindra sa valeur maximale. Au début de la résolution, α est initialisé à $\alpha_{\min}$ et augmenté de $\left((\alpha_{max} - \alpha_{min})/Cycle\_\alpha_{max}\right)$ à chaque

cycle jusqu'à ce qu'il atteigne $\alpha_{\max}$, alors sa valeur reste constante jusqu'à la fin de la résolution. De cette façon, l'importance de la trace de phéromone augmente en même temps que la pertinence de son contenu. Notons que α peut maintenant prendre des valeurs réelles et non plus uniquement des valeurs entières.

Les exposants β et δ, associés aux deux termes de visibilité dans la règle de transition, ont des comportements différents de , mais tous deux basés sur le même principe. Les valeurs de β et δ varient respectivement dans $[\beta_{min}, \beta_{max}]$ et $[\delta_{min}, \delta_{max}]$, par sauts de Inc_β et Inc_δ. Chacun des deux paramètres est initialisé à sa valeur minimale, puis augmenté à chaque cycle de l'incrément Inc correspondant jusqu'à ce qu'il atteigne sa valeur maximale. Il est alors diminué de Inc à chaque cycle jusqu'à ce qu'il égalise sa valeur minimale, puis augmenté de nouveau. Cependant, si la meilleure solution du cycle est meilleure que celle du cycle précédent, β et δ ne sont pas modifiés. L'idée est de conserver pour un cycle de plus une combinaison de β et δ qui s'est montrée efficace. La Figure 3.7 montre une représentation lissée (ne tenant pas compte des cycles où ceux-ci ne sont pas modifiés) de l'évolution des deux paramètres. Les exposants β et δ, comme α, prennent aussi des valeurs réelles. Notons que β_{min}, β_{max}, Inc_β, δ_{min}, δ_{max} et Inc_δ doivent respecter la contrainte de l'Équation 3.6 si l'on veut

éviter que les deux paramètres se dirigent toujours dans le même sens et explorer le plus de combinaisons possible.

$$\frac{\beta_{\max} - \beta_{\min}}{Inc_{\beta}} \neq \frac{\delta_{\max} - \delta_{\min}}{Inc_{\delta}} \tag{3.6}$$

L'algorithme résultant de l'ajout des exposants variables à l'OCF-3D est noté OCF-EV. Afin d'assurer une meilleure comparaison, les valeurs des paramètres ne sont pas modifiées, à l'exception des exposants de la règle de transition. Les valeurs choisies pour l'exécution de l'OCF-EV sont données au Tableau 3.10. Les valeurs précédemment données à α, β et δ, i.e. $\{4,6,3\}$, sont incluses dans les limites des exposants correspondants afin de permettre des combinaisons. Évidemment, les valeurs choisies respectent la contrainte de l'Équation 3.6.

Paramètre	$\alpha_{\min}$	$\alpha_{\max}$	$Cycle\_\alpha_{max}$	$\beta_{\min}$	$\beta_{\max}$	Inc_{β}	$\delta_{\min}$	$\delta_{\max}$	Inc_{δ}
Valeur	3	5	500	2	6	0,1	2	3	0,05

Tableau 3.10 : Valeurs des paramètres pour l'OCF-EV

Le Tableau 3.11 montre les résultats obtenus par l'OCF-EV sur l'ET1. Une fois de plus les performances sur l'ET1 demeurent inchangées et viennent simplement assurer que l'OCF-EV n'entraîne pas de détérioration pour les instances plus faciles.

Groupe d'instances	Meilleure solution connue	OCF-3D	OCF-EV		
		Nombre de conflits moyen	Nombre de conflits moyen	Cycle de sortie moyen	Temps de calcul moyen (sec)
60-*	0	0,00	0,00	1,31	0,01
65-*	0	0,00	0,00	1,48	0,02
70-*	0	0,00	0,00	1,86	0,03
75-*	0	0,00	0,00	2,43	0,03
80-*	0	0,00	0,00	2,48	0,04
85-*	0	0,00	0,00	2,46	0,04
90-*	0	0,00	0,00	2,11	0,03

Tableau 3.11 : Résultats obtenus par l'OCF-EV pour l'ET1

L'effet est tout autre pour l'ET2 dont les résultats sont reportés au Tableau 3.12. L'OCF-EV permet des améliorations du nombre de conflits de 1% à 100% pour 5 des 9 instances (10_93, 19_71, 21_90, 36_92 et 4_72 : en gras dans le tableau). Notons que l'unique amélioration de 100% est obtenue pour l'instance 4_72 et représente une diminution de 0,01 conflit. L'OCF-EV maintient les performances par rapport à l'OCF-3D pour les instances 41_66 et 6_76 (qui sont à l'optimum). On observe cependant une détérioration de l'ordre de 17% pour l'instance 16_81 et de 0,01 conflits pour le 26_82. Les valeurs minimales obtenues sont maintenant toutes égales aux meilleures solutions connues. Les cycles de sortie moyens sont légèrement plus élevés que pour l'OCF-3D, ce qui permet de croire que l'algorithme utilise plus efficacement le temps qui lui est alloué et que l'importance croissante accordée à la trace de phéromone contribue à l'atteinte de meilleures solutions. Les temps de calcul nécessaires demeurent équivalents à ceux des versions précédentes.

Instance	Meilleure solution connue	OCF-3D	OCF-EV					
		Nombre de conflits moyen	Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
10_93	3	4,03	**3,90**	0,50	3	5	297,52	6,83
16_81	0	0,58	0,68	0,66	0	2	128,60	4,39
19_71	2	2,04	**2,00**	0,00	2	2	205,21	6,40
21_90	2	2,02	**2,00**	0,00	2	2	158,52	6,22
26_82	0	0,00	0,01	0,10	0	1	33,54	0,26
36_92	2	2,03	**2,00**	0,00	2	2	146,78	6,13
4_72	0	0,01	**0,00**	0,00	0	0	56,55	0,33
41_66	0	0,00	0,00	0,00	0	0	4,93	0,03
6_76	6	6,00	6,00	0,00	6	6	2,44	5,62

Tableau 3.12 : Résultats obtenus par l'OCF-EV pour l'ET2

Les résultats obtenus par l'OCF-EV pour l'ET3 sont reportés au Tableau 3.13, où les nombres de conflits moyens en gras représentent des améliorations par rapport à l'OCF-3D. L'OCF-EV améliore les performances sur 8 des 30 instances, avec des améliorations des nombres de conflits moyens entre 1% et 48%. On observe cependant des augmentations de nombre de conflits moyen entre 1% et 60% pour 20 des 22 instances restantes. Les valeurs minimales obtenues sont maintenues ou abaissées, à l'exception des instances 300_04, 300_05, 400_01, 400_04, 400_05, 400_07, 400_08, 400_09 et 400_10. On observe que les cycles de sortie moyens sont stables ou en hausse pour la plupart des instances, alors qu'ils sont à la baisse pour celles dont la performance est inférieure à l'OCF-3D. On peut donc penser que l'OCF-EV, lorsqu'il est efficace, trouve ses solutions plus tard dans l'exécution que les versions précédentes. Ceci s'explique probablement

par la plus grande diversification de la recherche en début d'exécution. Finalement, les temps de calcul suivent une tendance légèrement à la baisse.

Instance	Meilleure solution connue	OCF-3D	OCF-EV					
		Nombre de conflits moyen	Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
200_01	0	2,00	2,48	0,77	1	4	341,76	13,43
200_02	2	2,38	**2,09**	0,29	2	3	399,71	13,42
200_03	4	7,45	7,82	0,73	6	9	331,37	13,51
200_04	7	7,87	**7,54**	0,50	7	8	373,73	14,08
200_05	6	7,29	**7,11**	0,35	6	8	415,25	12,76
200_06	6	6,03	6,04	0,20	6	7	255,07	12,85
200_07	0	0,67	**0,35**	0,52	0	2	501,66	8,83
200_08	8	8,00	8,00	0,00	8	8	36,06	11,81
200_09	10	10,97	**10,47**	0,50	10	11	393,99	13,61
200_10	19	20,19	20,59	0,65	19	22	370,54	12,29
300_01	0	3,89	4,35	0,69	2	6	389,58	22,63
300_02	12	12,57	12,99	0,41	12	14	199,01	23,04
300_03	13	13,85	13,91	0,32	13	15	318,12	22,93
300_04	7	8,69	**8,63**	0,51	8	10	369,98	21,20
300_05	29	42,54	44,91	1,22	42	48	288,15	20,30
300_06	2	5,79	6,29	0,74	4	8	417,86	21,93
300_07	0	0,97	**0,68**	0,60	0	2	464,78	17,72
300_08	8	8,95	8,95	0,26	8	10	244,27	20,15
300_09	7	8,00	8,66	0,54	7	10	303,99	19,86
300_10	21	32,56	33,39	1,38	29	36	483,49	19,28
400_01	1	3,50	4,20	0,77	3	6	94,38	28,29
400_02	16	23,82	25,27	1,01	22	27	347,14	27,27
400_03	9	13,64	14,91	0,78	12	16	79,52	27,15
400_04	19	20,38	20,74	0,56	20	22	392,73	30,68
400_05	0	2,68	4,30	1,05	2	7	197,19	26,16
400_06	0	1,53	**1,49**	0,63	0	3	458,20	26,75
400_07	4	8,68	10,68	1,41	7	14	167,35	26,14
400_08	4	12,67	16,32	1,57	13	20	270,36	25,12
400_09	5	16,01	17,30	1,42	14	21	456,08	28,94
400_10	0	2,66	3,72	0,79	2	5	254,37	28,05

Tableau 3.13 : Résultats obtenus par l'OCF-EV pour l'ET3

Afin d'illustrer la variation dans le temps des valeurs des exposants, on a retenu une exécution-type d'une instance, i.e. une exécution pour laquelle la solution finale est trouvée à un cycle proche de la moyenne observée et où la solution trouvée correspond à la meilleure connue. L'instance choisie est le 200_07 parce qu'il s'agit de celle pour laquelle l'OCF-EV a engendré le meilleur pourcentage de gain. À chaque cycle de cette exécution-test, on a noté les valeurs de α, β et δ, le nombre de conflits de la meilleure solution du cycle (L_+) et celui de la meilleure solution globale (L_{gb}). Le graphique de la Figure 3.8 montre la variation de ces éléments en fonction du cycle. On observe d'abord α dont la progression est linéaire jusqu'au cycle 500, à partir duquel il se maintient. En regardant l'évolution de L_+, on voit que les valeurs les plus basses (2 et 3 conflits) sont plus fréquentes une fois que α a atteint sa valeur maximale. En ce qui concerne β et δ, qui oscillent entre leurs bornes respectives, il semble y avoir des combinaisons plus efficaces lorsqu'on regarde les cycles auxquels L_{gb} a été améliorée. En effet, les solutions comportant 2 conflits sont atteintes lorsque β a une valeur plus faible et δ une valeur faible à moyenne. On peut déduire que la force apportée par l'OCF-EV est une diversification de la recherche en permettant à chaque élément de la règle de transition d'être mis plus ou moins à contribution pour le calcul de la règle de transition. Les variations des valeurs de β et δ permettent également à l'algorithme de construire des solutions avec différentes combinaisons pour ces deux paramètres et ainsi de rencontrer celles qui sont les

plus efficaces. Notons cependant que ces conclusions, quoique logiques et confirmant les hypothèses avancées, ont été tirées grâce à l'observation d'une seule exécution qui pourrait ne pas représenter la majorité des cas.

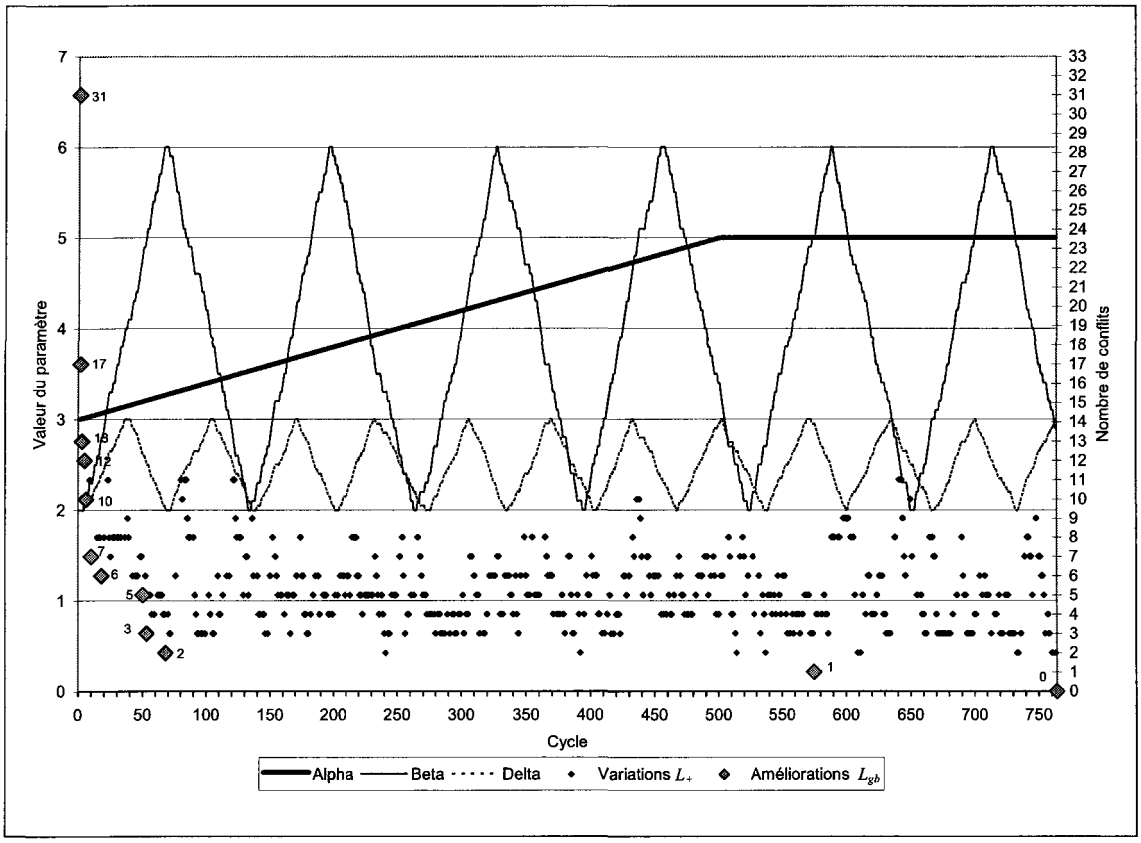

Figure 3.8 : Variation des paramètres α, β et δ et des *meilleures solutions globale et du cycle en fonction* du cycle d'exécution pour une résolution-type de l'instance 200_07

En conclusion, lorsqu'il est efficace, l'ajout des exposants variables entraîne une meilleure diversification de la recherche dans l'espace de solutions et donne une chance à l'algorithme de réunir les valeurs de paramètres les plus efficaces. L'exploration a pour conséquence de repousser le cycle de sortie de la solution

finale. Il existe un risque que l'algorithme ne puisse atteindre un nombre de conflits assez bas dans le temps qui lui est alloué, on peut donc envisager d'autres tests où le nombre de cycles maximal serait augmenté. On observe une perte d'efficacité pour l'OCF-EV pour des instances de grande taille. Ceci permet de conclure qu'il semble risqué d'encourager une trop grande diversification pour ces instances auxquelles sont associés de plus grands espaces de solutions. Finalement, en raison des résultats intéressants obtenus, l'OCF-EV est retenu pour la suite du travail.

3.5 Nouveau mode de construction de solution

3.5.1 La construction dynamique

L'OCF, tel que nous le connaissons, est un algorithme de construction séquentiel qui ne remet jamais en cause les décisions prises. Une nouvelle façon de construire les solutions est proposée dans cette section. Il s'agit du concept de la Construction Dynamique (CD). Selon cette méthode, les fourmis construisent dynamiquement une séquence, dans laquelle elles ajoutent une nouvelle voiture en l'insérant à n'importe quel endroit de la séquence partielle déjà construite. De cette manière, il est possible d'ajouter des conflits à la séquence temporaire, puis de les annuler par l'insertion d'autres voitures plus tard dans la construction. Cette façon de faire s'inspire du fait observé selon lequel il arrive qu'une voiture impossible à placer sans conflit après un certain temps trouve une place avantageuse parmi les positions déjà placées.

Les termes *séquentiel* et *dynamique* sont utilisés comme opposés. Le premier caractérise une méthode selon laquelle la séquence est bâtie de la position 1 à n dans l'ordre. Le second fait référence à une construction où la séquence est bâtie à l'aide d'une chaîne dynamique de voitures dans laquelle on peut insérer à toutes les positions, en décalant les éléments déjà en place.

Pour la CD, la logique de construction des solutions est inversée. À chaque pas, une fourmi doit d'abord choisir une classe de voitures. Ce choix se fera aléatoirement. Ensuite, la position $PosChoisie$ à laquelle une voiture de cette

classe sera insérée est sélectionnée selon une règle de transition pseudo-proportionnelle aléatoire modifiée dont les candidats sont les positions, tel qu'illustré à l'Équation 3.7.

$$PosChoisie = \begin{cases} \arg\max_{Pos' \in [1,y^*]} \left\{ \left[Intérêt^{i}_{Pos'} \right]^{\alpha} \cdot \left[\eta'_{Pos'} \right]^{\beta} \right\} & si\ Q \le Q_0, \\ \psi & sinon \end{cases}$$

où ψ est choisi selon la probabilité suivante: (3.7)

$$P^{k}_{i,\,Pos} = \frac{\left[Intérêt^{i}_{Pos} \right]^{\alpha} \cdot \left[\eta'_{Pos} \right]^{\beta}}{\sum_{Pos' \in [1,y^*]} \left\{ \left[Intérêt^{i}_{Pos'} \right]^{\alpha} \cdot \left[\eta'_{Pos'} \right]^{\beta} \right\}}$$

Trois adaptations sont à noter pour la règle de transition :

(1) L'utilisation de la trace de phéromone à trois dimensions est adaptée au caractère dynamique de l'algorithme. Lorsqu'on envisage une position pour une classe de voitures, on considère l'intérêt d'insérer celle-ci à une position donnée, mais également l'intérêt de ne pas perturber le bloc de voitures déjà en place. Le calcul de la trace de phéromone correspond maintenant à la différence entre ces deux mesures.

L'intérêt de placer une classe i en position Pos dans une séquence S_k est donné par la somme des traces des positions adjacentes à $S_k(Pos)$, tel que montré à l'Équation 3.8.

$$\sum_{y=Pos-p_{\max}}^{Pos+p_{\max}} \tau_{S(y),i,|y-Pos|}, où\ y \neq Pos \qquad (3.8)$$

Le calcul de la valeur du terme de trace de phéromone de la règle de transition modifiée est illustré à la Figure 3.9. La partie supérieure de cette figure montre une sous-séquence de solution intacte et cette même sous-séquence ajoutée de la classe choisie i en position $PosChoisie$ est illustrée dans la partie inférieure. Le terme de la trace $Intérêt^{i}_{Pos}$ est obtenu en soustrayant l'intérêt d'avoir en position Pos la classe qui y est déjà (voir (b)), de l'intérêt d'insérer la classe i (voir (a)), en décalant d'une position par la droite chacune des voitures suivantes. Comme l'élément en position Pos (encerclé de pointillés) est décalé vers la droite si i est insérée, on ne considère que les positions $(Pos - p_{max})$ à $(Pos + p_{max} - 1)$ dans le calcul de l'intérêt de l'insertion.

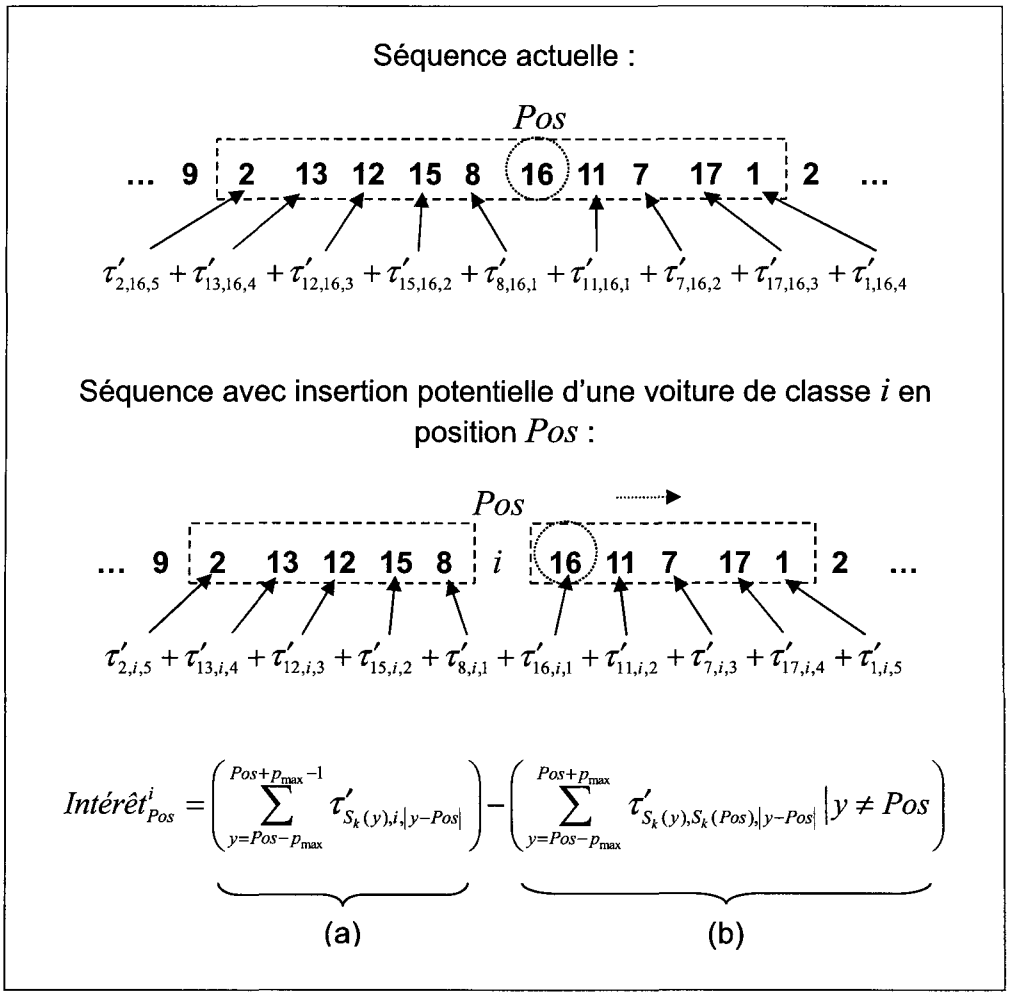

Figure 3.9 : Illustration de la trace de phéromone utilisée dans la règle de transition modifiée de l'OCF-M

(2) L'unique terme de visibilité locale η'_{Pos} de la règle de transition modifiée est fonction de $NouvConflitsPosition^{i}_{Pos}$, le nombre de nouveaux conflits ajoutés par l'insertion de la classe i à la position Pos. Ce nombre peut être négatif si l'insertion annule un ou plusieurs conflits autour de Pos. Afin d'obtenir un terme positif pour chaque position candidate, la valeur de η'_{Pos} est obtenue selon

l'Équation 3.9, où $PPNouvConflits_i$ est le plus petit nombre de nouveaux conflits que i peut ajouter parmi toutes les positions candidates.

$$\eta'_{Pos} = \frac{1}{1 + NouvConflitsPosition^{i}_{Pos} - PPNouvConflits_i} \qquad (3.9)$$

(3) Il n'y a pas de liste de candidats qui restreint les positions pouvant être choisies. Toutes les positions comprises dans $[1, y^*]$ sont candidates.

S'ajoutant à ces modifications, la mise à jour globale de la trace de phéromone est effectuée de la même manière que pour l'OCF-3D. Par contre, la mise à jour locale est faite sur un horizon de $(2p_{max})$, selon la Figure 3.10, où i est la classe insérée et $PosChoisie$ est la position choisie par la fourmi k.

POUR chaque position y entre $(PosChoisie - p_{max})$ et $(PosChoisie + p_{max})$

$$\tau_{i,S_k(y),|PosChoisie-y|} = \rho_{local} \cdot \tau_{i,S_k(y),|PosChoisie-y|} + (1 - \rho_{local}) \cdot \Delta\tau$$

Figure 3.10 : Mise à jour locale pour la CD

Les premiers essais réalisés avec la construction dynamique ont montré des performances inférieures aux résultats connus sur les instances test. Cela s'explique par le fait qu'en début de construction, lorsque la séquence dynamique est courte, la fourmi ajoute de nombreux conflits puisqu'elle ne dispose pas de l'espace nécessaire à un quelconque lissage des conflits. Elle ne parvient pas par la suite à éliminer ces conflits trop nombreux à l'aide des classes de voitures

qu'elle insère. De plus, les temps de calcul nécessaires à son exécution sont très élevés, même pour les plus petites instances.

Le concept n'est cependant pas sans valeur, et c'est ce qui amène à penser qu'une version hybride entre la construction séquentielle et la construction dynamique pourrait être intéressante. L'intégration de la CD à la dernière version de l'algorithme (OCF-EV) pour obtenir une construction de nature mixte est traitée à la section suivante.

3.5.2 La construction mixte (OCF-M)

L'observation de la construction des solutions de l'OCF-EV permet d'établir que lorsqu'une fourmi arrive en fin de solution, elle a plus de chances d'ajouter un ou plusieurs conflits, étant donné la liste de classes disponibles qui diminue. De cette façon, comme l'algorithme séquentiel fixe définitivement les positions de la première à la dernière, beaucoup de conflits se retrouvent en dernière portion de séquence. Afin de contrer cet effet, la CD est intégrée à l'OCF-EV et l'algorithme résultant est appelé OCF-M pour OCF Mixte.

L'OCF-M présente une méthode de construction mixte, où les fourmis fonctionnent en mode séquentiel, mais peuvent passer en mode dynamique selon deux conditions qui seront présentées dans les paragraphes suivants. Lorsqu'une fourmi passe en mode dynamique, elle inverse sa logique et place une voiture selon la méthode dynamique, puis elle revient en mode séquentiel. Il est à noter

que chaque fourmi est indépendante des autres pour choisir son mode de construction.

La Figure 3.11 montre le pseudo-code du pas d'une fourmi k de l'OCF-M. On observe d'abord en (a) la première condition de passage en mode dynamique pour tenter d'éliminer un ou des conflits lorsque l'évaluation L_k' de la séquence partielle en compte au moins un. En (b), la fourmi fait le choix potentiel d'une classe i' à insérer en position y*, en mode séquentiel. Étant donné la gestion des candidats, si la classe choisie engendre des conflits, cela signifie que l'ensemble des classes restantes génèrent aussi des conflits. On peut alors choisir, en (c), de passer en mode dynamique selon la deuxième condition. La meilleure des deux insertions envisagées est alors appliquée. Le cours normal d'un pas (l'insertion en séquentiel d'une voiture à la fin de la sous-séquence) se retrouve en (d). Après chaque insertion d'une nouvelle voiture, la fourmi repasse en mode séquentiel, tel que montré en (e). Les paragraphes suivants expliquent le détail des deux conditions de passage en mode dynamique.

Tirer un nombre aléatoire ϕ entre 0 et 1

SI ($L'_k > 0$ **ET** $\phi \leq \varphi_{Conflits}$) (a)

Passer en mode dynamique
Choisir i aléatoirement
Choisir $PosChoisie$ entre 1 et y^* selon la règle de transition modifiée de l'Équation 3.7

SI ($NouvConflits_{PosChoisie} < 0$)

Insérer i en $PosChoisie$

SINON

Repasser en mode séquentiel et placer une voiture

SINON

Choisir une voiture de classe i' en mode séquentiel (b)
Tirer un nombre aléatoire ϕ entre 0 et 1

SI ($NouvConflits_i > 0$ **ET** $\phi \leq \varphi_{ToutesGénèrent}$) (c)

Passer en mode dynamique
Choisir i aléatoirement
Choisir $PosChoisie$ entre $(y^* - p_{\max} + 1)$ et y^* selon la règle de transition modifiée de l'Équation 3.7

SI ($NouvConflits_i \leq NouvConflitsPosition^i_{PosChoisie}$)

Repasser en mode séquentiel
Placer une voiture de classe i' en position y^*

SINON

Insérer i en $PosChoisie$

SINON

Placer une voiture de classe i' en position y^* (d)

Repasser en mode séquentiel (e)

Figure 3.11 : Pseudo-code d'un pas pour une fourmi k avec mode de construction dynamique des solutions

La première condition de passage est validée dans une certaine probabilité $\varphi_{Conflits}$ si l'évaluation de la séquence en cours de construction présente au moins un conflit. Dans ce cas, la fourmi passe en mode dynamique avec toutes les positions candidates. L'idée est de retirer un ou plusieurs conflits en insérant des voitures dans la séquence déjà fixée. Une fourmi choisit aléatoirement les classes à tenter d'insérer parmi un ensemble de classes possibles. Cet ensemble est initialisé à toutes les classes disponibles et la fourmi en retire celles qu'elle tente de placer sans succès. On cherche ainsi à ne pas réessayer inutilement les classes qui ne parviennent pas à éliminer les conflits déjà en place. Si le choix fait en mode dynamique ne retire pas de conflit, la fourmi repasse en mode séquentiel pour placer une voiture. La probabilité $\varphi_{Conflits}$ permet d'éviter de construire toujours en mode dynamique à partir du premier conflit. Il est à noter que cette probabilité doit être relativement faible.

La seconde condition de passage en mode dynamique est vérifiée si toutes les classes candidates en mode séquentiel génèrent au moins un conflit (voir Figure 3.11 en (a)). Si elle restait en mode séquentiel, la fourmi serait alors contrainte d'ajouter au moins un conflit. Dans la construction mixte, elle passe en mode de construction dynamique, avec la particularité de considérer comme candidates uniquement les p_{max} dernières positions de la séquence. Ainsi, elle a une chance de perturber la fin de la séquence et de permettre par la suite l'insertion séquentielle d'une voiture sans générer de conflit. Le choix fait en mode

dynamique n'est pas automatiquement appliqué. En effet, la fourmi retient le meilleur des deux choix (séquentiel ou dynamique). Sur le même principe que pour la première condition de passage, une probabilité $\varphi_{ToutesGénèrent}$ a été ajoutée à cette seconde condition.

Pour les exécutions présentées, les valeurs de $\varphi_{Conflits}$ et $\varphi_{ToutesGénèrent}$ sont fixées respectivement à 0,1 (10% de chances) et 0,05 (5% de chances). Plusieurs combinaisons de valeurs ont été testées, ce qui a permis de conclure que ces paramètres doivent être fixés à de petites valeurs. En effet, si l'on utilise toutes les classes intéressantes (celles qui ont moins d'options, les moins contraintes, etc.) en tentant trop souvent de pallier à l'ajout de conflits en début de séquence, on se retrouve en dernière partie de séquence avec des classes difficiles. Ceci entraîne des conflits qui annulent les gains de la construction dynamique.

Le Tableau 3.14 montre les résultats obtenus pour l'ET1. Les nombres de conflits moyens sont maintenus à 0, les temps de calculs et les cycles de sortie moyens restent équivalents à l'OCF-EV.

Groupe d'instances	Meilleure solution connue	OCF-EV Nombre de conflits moyen	OCF-M Nombre de conflits moyen	OCF-M Cycle de sortie moyen	OCF-M Temps de calcul moyen (sec)
60-*	0	0,00	0,00	1,31	0,01
65-*	0	0,00	0,00	1,44	0,02
70-*	0	0,00	0,00	1,73	0,03
75-*	0	0,00	0,00	2,21	0,04
80-*	0	0,00	0,00	2,32	0,04
85-*	0	0,00	0,00	2,25	0,04
90-*	0	0,00	0,00	1,94	0,03

Tableau 3.14 : Résultats obtenus par l'OCF-M pour l'ET1

Les résultats de l'OCF-M sur l'ET2 sont présentés au Tableau 3.15. Les nombres de conflits moyens sont maintenus par rapport à l'OCF-EV pour 5 des 9 instances et des améliorations mineures de 0,07 et 0,01 conflit sont observées pour les instances 16_81 et 26_82 respectivement (en gras dans le tableau). On remarque 2 légères dégradations de 0,02 conflit pour les instances 19_71 et 21_90. Les valeurs minimales et maximales sont maintenues, sauf pour l'instance 26_82 où la valeur maximale est abaissée et pour les instances 10_93, 19_71 et 21_90 où elles sont augmentées. Aucune tendance n'est observée pour les cycles de sortie moyens, qui sont équivalents à la version précédente. Les temps de calculs se maintiennent, mises à part quelques légères augmentations. Bien que le mode de construction dynamique soit relativement lourd en termes de temps de calcul, le temps nécessaire aux exécutions de l'OCF-M n'augmente pas de manière significative par rapport à l'OCF-EV. Cette stabilité est due aux valeurs des probabilités de passage qui limitent l'utilisation de la CD.

Instance	Meilleure solution connue	OCF-EV	OCF- M					
		Nombre de conflits moyen	Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
10_93	3	3,90	3,90	0,56	3	6	270,84	7,80
16_81	0	0,68	**0,61**	0,69	0	2	138,80	4,36
19_71	2	2,00	2,02	0,14	2	3	192,53	7,37
21_90	2	2,00	2,02	0,14	2	3	140,68	6,97
26_82	0	0,01	**0,00**	0,00	0	0	38,11	0,27
36_92	2	2,00	2,00	0,00	2	2	122,81	7,13
4_72	0	0,00	0,00	0,00	0	0	65,40	0,43
41_66	0	0,00	0,00	0,00	0	0	5,48	0,04
6_76	6	6,00	6,00	0,00	6	6	1,00	6,33

Tableau 3.15 : Résultats obtenus par l'OCF-M pour l'ET2

Le Tableau 3.16 contient les résultats obtenus sur l'ET3. L'OCF-M a un effet plus marquant sur l'ET3 que sur l'ET2. On observe des améliorations entre 1% et 69%sur 28 des 30 instances par rapport à l'OCF-EV. Les deux instances qui y font exception sont le 200_02, qui subit une détérioration mineure de 1%, et le 200_08, dont la performance est maintenue. Les valeurs minimales obtenues ont diminué ou sont maintenues pour toutes les instances. On observe même des diminutions de 4 conflits en moyenne pour le 300_05 et de 5 conflits en moyenne pour le 400_09. Le même constat peut être fait pour les valeurs maximales qui sont abaissées ou maintenues dans la totalité des cas. Les cycles de sortie moyens sont encore une fois équivalents à la version précédente. Les temps de calcul augmentent cependant de manière significative. L'OCF-M présente des augmentations de temps de calcul entre 2% et 80% par rapport à l'OCF-EV. Cet

accroissement du temps de calcul s'expliquera plus tard par les statistiques de construction dynamique présentées dans les paragraphes suivants.

Instance	Meilleure solution connue	OCF-EV	OCF- M					
		Nombre de conflits moyen	Nombre de conflits moyen	Écart-type	MIN	MAX	Cycle de sortie moyen	Temps de calcul moyen (sec)
200_01	0	2,48	**2,13**	0,65	1	4	356,65	16,18
200_02	2	2,09	2,11	0,31	2	3	394,16	15,73
200_03	4	7,82	**7,46**	0,56	6	8	312,44	16,04
200_04	7	7,54	**7,33**	0,47	7	8	423,48	18,11
200_05	6	7,11	**6,98**	0,20	6	8	416,64	16,78
200_06	6	6,04	**6,00**	0,00	6	6	223,10	16,68
200_07	0	0,35	**0,11**	0,35	0	2	514,08	9,02
200_08	8	8,00	8,00	0,00	8	8	20,85	15,85
200_09	10	10,47	**10,24**	0,43	10	11	461,50	17,84
200_10	19	20,59	**20,04**	0,63	19	21	363,11	19,88
300_01	0	4,35	**4,14**	0,73	2	6	327,18	29,61
300_02	12	12,99	**12,70**	0,46	12	13	173,24	29,63
300_03	13	13,91	**13,60**	0,49	13	14	282,64	31,33
300_04	7	8,63	**8,60**	0,60	7	10	411,43	30,13
300_05	29	44,91	**41,96**	1,06	38	44	262,16	34,76
300_06	2	6,29	**5,69**	0,73	4	7	445,87	26,09
300_07	0	0,68	**0,62**	0,51	0	2	469,02	22,28
300_08	8	8,95	**8,82**	0,39	8	9	169,10	29,45
300_09	7	8,66	**8,09**	0,40	7	9	238,89	27,43
300_10	21	33,39	**32,17**	1,15	29	35	465,49	35,28
400_01	1	4,20	**4,04**	0,65	3	5	54,21	38,15
400_02	16	25,27	**23,48**	0,83	21	25	339,53	47,66
400_03	9	14,91	**14,25**	0,77	12	16	62,01	33,88
400_04	19	20,74	**20,24**	0,45	19	21	271,00	52,23
400_05	0	4,30	**3,67**	0,95	1	6	155,84	35,26
400_06	0	1,49	**1,35**	0,61	0	3	433,38	35,54
400_07	4	10,68	**9,60**	1,09	7	12	173,44	33,20
400_08	4	16,32	**13,30**	1,70	8	17	44,47	35,51
400_09	5	17,30	**15,88**	1,12	13	18	424,28	52,09
400_10	0	3,72	**3,44**	0,78	2	5	158,60	41,09

Tableau 3.16 : Résultats obtenus par l'OCF- M pour l'ET3

Afin d'évaluer dans quelles proportions l'OCF-M fonctionne en mode dynamique, des statistiques sur les décisions prises par l'algorithme ont été compilées. Trois termes sont utilisés pour ces statistiques : *décision*, *tentative* et *réussite*. Une décision correspond à placer une voiture dans la séquence. Il y a donc exactement $NC_{Final} \cdot m \cdot n$ décisions prises dans une exécution, où NC_{Final} est le nombre de cycles effectués, m est le nombre de fourmis et n, la taille de l'instance. Il y a tentative lorsque l'algorithme passe en mode dynamique. Comme ces passages n'engendrent pas nécessairement le placement d'une voiture en mode dynamique, on considère également les réussites, correspondant aux décisions prises en mode dynamique.

Les Tableaux 3.17, 3.18 et 3.19 regroupent ces statistiques pour l'ET1, l'ET2 et l'ET3 et présentent la colonne « Groupe d'instances » ou « Instance » et pour chaque condition de passage les colonnes « Nombre de tentatives », « % d'efficacité » et « % des décisions ». La colonne « Groupe d'instances » ou « Instance » indique le nom de l'instance ou du groupe. Le *nombre de tentatives* est le nombre moyen de fois au cours d'une résolution où l'algorithme passe en mode dynamique, alors que le *pourcentage d'efficacité* représente la moyenne des ratios réussites-tentatives (ramené sur 100). Le *pourcentage des décisions* donne la moyenne des ratios réussites-décisions, il indique quel pourcentage des décisions ont été prises en mode dynamique. Pour le cas de l'ET1, les moyennes présentées sont calculées sur l'ensemble des statistiques de toutes les exécutions d'un même groupe d'instances.

Groupe d'instances	Première condition de passage : « Il y a au moins 1 conflit »			Deuxième condition de passage : « Toutes génèrent des conflits »		
	Nombre de tentatives	% d'efficacité	% des décisions	Nombre de tentatives	% d'efficacité	% des décisions
60-*	114,40	1,24	0,0248	2,92	16,45	0,0133
65-*	130,07	0,12	0,0035	4,22	12,91	0,0115
70-*	147,17	0,13	0,0039	6,73	10,40	0,0109
75-*	168,13	0,18	0,0047	11,37	8,55	0,0113
80-*	185,82	0,14	0,0044	12,31	8,44	0,0123
85-*	148,79	0,06	0,0016	10,36	4,54	0,0062
90-*	119,17	0,05	0,0015	8,59	5,55	0,0066

Tableau 3.17 : Statistiques moyennes de construction dynamique pour l'ET1

Instance	Première condition de passage : « Il y a au moins 1 conflit »			Deuxième condition de passage : « Toutes génèrent des conflits »		
	Nombre de tentatives	% d'efficacité	% des décisions	Nombre de tentatives	% d'efficacité	% des décisions
10_93	54248,46	0,07	0,003	13386,65	2,65	0,02
16_81	23414,12	0,13	0,004	4111,91	3,28	0,02
19_71	53858,35	0,02	0,001	9201,67	5,22	0,03
21_90	44800,52	0,03	0,001	7632,22	5,64	0,03
26_82	1402,55	0,04	0,001	260,69	3,11	0,02
36_92	58224,45	0,08	0,003	9958,62	3,32	0,02
4_72	2479,53	0,24	0,006	384,10	6,06	0,03
41_66	135,63	0,06	0,001	31,62	4,42	0,02
6_76	32771,79	0,03	0,001	10099,02	5,83	0,04

Tableau 3.18 : Statistiques moyennes de construction dynamique pour l'ET2

Instance	Première condition de passage : « Il y a au moins 1 conflit »			Deuxième condition de passage : « Toutes génèrent des conflits »		
	Nombre de tentatives	% d'efficacité	% des décisions	Nombre de tentatives	% d'efficacité	% des décisions
200_01	103261,21	0,13	0,0045	16462,07	2,57	0,01
200_02	90548,32	0,00	0,0001	11784,81	5,09	0,02
200_03	100758,51	0,20	0,0066	19042,72	3,02	0,02
200_04	121059,92	0,02	0,0009	19957,60	3,70	0,02
200_05	130088,07	0,11	0,0049	18743,01	5,38	0,03
200_06	119736,28	0,01	0,0005	15931,60	6,62	0,04
200_07	67211,83	0,02	0,0009	7934,59	2,59	0,01
200_08	131758,94	0,02	0,0010	18114,79	6,95	0,04
200_09	119583,32	0,04	0,0016	22744,28	3,92	0,03
200_10	140122,52	0,07	0,0033	40450,99	4,80	0,06
300_01	196444,76	0,02	0,0008	20282,46	3,44	0,02
300_02	191928,35	0,01	0,0003	30629,14	3,00	0,02
300_03	195621,81	0,03	0,0015	30472,84	4,57	0,03
300_04	210454,75	0,02	0,0011	25845,16	4,51	0,03
300_05	202584,25	0,04	0,0018	67205,08	1,22	0,02
300_06	159204,62	0,01	0,0005	23387,25	3,90	0,02
300_07	153486,34	0,02	0,0009	13335,44	5,19	0,02
300_08	194647,95	0,03	0,0013	26416,18	5,14	0,03
300_09	189342,31	0,13	0,0054	25150,47	5,85	0,03
300_10	209967,66	0,17	0,0080	66648,33	2,27	0,03
400_01	230856,70	0,02	0,0007	25027,10	2,13	0,01
400_02	279781,37	0,04	0,0018	53934,69	4,02	0,04
400_03	208149,69	0,00	0,0001	34904,65	1,38	0,01
400_04	276919,69	0,04	0,0021	45530,96	4,25	0,03
400_05	252850,63	0,10	0,0042	25831,86	5,59	0,02
400_06	234543,49	0,01	0,0002	17524,92	5,62	0,02
400_07	205188,17	0,00	0,0001	31250,00	3,11	0,02
400_08	246884,03	0,04	0,0016	41650,10	2,60	0,02
400_09	275381,97	0,11	0,0049	45741,20	2,33	0,02
400_10	252330,52	0,01	0,0005	21070,41	4,16	0,01

Tableau 3.19 : Statistiques moyennes de construction dynamique pour l'ET3

En observant les pourcentages de décisions pour l'ET1, on remarque que le mode de construction dynamique n'est pas très utilisé, ce qui peut s'expliquer par

la facilité de résolution relative des instances. L'algorithme prend un pourcentage négligeable de ses décisions en mode dynamique, soit entre 0,001 et 0,025% par la condition « Il y a au moins un conflit » et entre 0,006% et 0,013% par la condition « Toutes génèrent ». On remarque cependant que lorsqu'il est utilisé, le mode dynamique est relativement efficace, comme le montrent les pourcentages d'efficacité. Nous verrons que les pourcentages d'efficacité sont plus élevés pour l'ET1 que pour l'ET2 et l'ET3, ce qui s'explique encore par la facilité relative des instances.

Des constats semblables peuvent être faits pour l'ET2 et l'ET3, avec des pourcentages d'efficacité et de décisions légèrement plus faibles dans l'ensemble. Les pourcentages d'efficacité peuvent sembler faibles, mais ils pourraient signifier simplement que l'algorithme en mode séquentiel est en mesure de construire d'excellentes solutions et que le mode dynamique est utilisé sporadiquement pour tenter d'améliorer ces solutions déjà bonnes. Cette théorie est appuyée par des pourcentages d'efficacité intéressants qui sont toutefois moins élevés pour le passage par « Il y a au moins un conflit ». Ceci s'explique par le fait qu'il est plus facile de perturber la fin de la séquence partielle (même si cela entraîne des conflits) que d'éliminer des conflits dans l'ensemble de la solution.

On observe que les classes qui sont placées lors des essais profitables (correspondant aux réussites) sont à peu près toujours les mêmes pour l'ensemble des résolutions d'une même instance. Pour chaque instance, il existe quelques classes (une ou deux) que la construction dynamique utilise toujours. Les classes

profitables pour le passage par « Il y a au moins un conflit » sont toujours des classes requérant une seule option et pour lesquelles on a une demande suffisamment grande. Ces mêmes classes sont également très avantageuses pour le passage par « Toutes génèrent des conflits », où elles laissent place aux autres classes qui se partagent toutefois une maigre part des réussites.

L'observation du déroulement de l'algorithme et des choix faits permet de déduire que pour être efficace, l'OCF-M doit être appliqué à des instances pour lesquelles il existe des classes qui ont à la fois peu d'options (idéalement une seule), une demande suffisante, mais surtout, des difficultés d pour les options requises qui laissent une marge de manœuvre, donc qui n'approchent pas 1,0. Ces conditions sont remplies par l'ET2 et l'ET3, et c'est sur ce dernier que l'OCF-M a été le plus bénéfique.

On peut donc dire que dans la forme où elle a été présentée, avec les valeurs de paramètres données, la construction mixte est profitable pour les instances qui remplissent les conditions énumérées plus haut, et qu'elle maintient le niveau de performance général pour les autres instances.

3.6 Récapitulation des différentes versions d'algorithme présentées

Trois nouvelles versions de l'OCF ont été présentées dans ce chapitre. Il serait maintenant pertinent de faire une récapitulation de leur évolution, afin d'identifier les tendances et de tirer les conclusions appropriées par rapport à l'algorithme de départ, l'OCF-0.

L'algorithme de Gravel *et al.* [GRA'05], l'OCF-0, a été présenté à la Section 3.2. Il s'agit d'un OCF, auquel ont été ajoutés un second terme de visibilité locale et une gestion plus restrictive des candidats. La trace de phéromone associe à chaque paire de classes de voitures un bénéfice potentiel à les placer l'une à côté de l'autre dans la séquence. Les résultats obtenus étaient très encourageants et sont reportés aux Tableaux 3.20 et 3.21 pour l'ET2 et l'ET3 respectivement. Les résultats obtenus par Gravel *et al.* avec le même OCF ajouté d'une procédure de recherche locale, que l'on nommera OCF-0+, sont également présentés dans les dernières colonnes des mêmes tableaux. Ces derniers sont donnés à titre indicatif et ne peuvent pas être directement comparés aux autres versions d'algorithmes présentées étant donné la force de calcul plus grande donnée à l'algorithme par la recherche locale.

À partir de l'OCF-0, l'OCF-3D a été obtenu en ajoutant une dimension à la matrice de trace de phéromone. Nous ne considérons pas l'OCF-H pour cette récapitulation, étant donné son statut de phase préliminaire et ses résultats peu concluants. De là, une variation en cours d'exécution des exposants des trois termes de la règle de transition a été intégrée pour arriver à l'OCF-EV. Finalement,

l'OCF-M a été présenté, pour lequel un nouveau concept de construction mixte des solutions est proposé. Les nombres de conflits moyens des trois versions de l'algorithme sont reportés dans les Tableaux 3.20 et 3.21, où ils peuvent être comparés entre eux. Les moyennes en gras correspondent aux meilleurs nombres de conflits moyens, toutes versions confondues (à l'exception de l'OCF-0+). Les résultats pour l'ET1 ont été omis, considérant que les moyennes ont été maintenues à 0 conflit et qu'elles n'offrent pas d'intérêt pour cette analyse.

Instance	Meilleure solution connue	OCF-0	OCF-3D	OCF-EV	OCF-M	OCF-0+ [GRA'05]
10_93	3	4,24	4,03	**3,90**	**3,90**	3,80
16_81	0	**0,12**	0,58	0,68	0,61	0,00
19_71	2	2,08	2,04	**2,00**	2,02	2,00
21_90	2	2,63	2,02	**2,00**	2,02	2,00
26_82	0	**0,00**	**0,00**	0,01	**0,00**	0,00
36_92	2	2,29	2,03	**2,00**	**2,00**	2,00
4_72	0	**0,00**	0,01	**0,00**	**0,00**	0,00
41_66	0	**0,00**	**0,00**	**0,00**	**0,00**	0,00
6_76	6	**6,00**	**6,00**	**6,00**	**6,00**	6,00

Tableau 3.20 : Récapitulatif des nombres de conflits moyens obtenus par chaque version de l'OCF pour l'ET2

Instance	Meilleure solution connue	OCF-0	OCF-3D	OCF-EV	OCF-M	OCF-0+ [GRA'05]
200_01	0	3,80	**2,00**	2,48	2,13	1,00
200_02	2	4,14	2,38	**2,09**	2,11	2,41
200_03	4	8,90	**7,45**	7,82	7,46	6,04
200_04	7	9,86	7,87	7,54	**7,33**	7,57
200_05	6	8,81	7,29	7,11	**6,98**	6,40
200_06	6	6,87	6,03	6,04	**6,00**	6,00
200_07	0	2,99	0,67	0,35	**0,11**	0,00
200_08	8	**8,00**	**8,00**	**8,00**	**8,00**	8,00
200_09	10	11,85	10,97	10,47	**10,24**	10,00
200_10	19	21,44	20,19	20,59	**20,04**	19,09
300_01	0	5,33	**3,89**	4,35	4,14	2,15
300_02	12	13,15	**12,57**	12,99	12,70	12,02
300_03	13	14,54	13,85	13,91	**13,60**	13,06
300_04	7	10,33	8,69	8,63	**8,60**	8,16
300_05	29	**40,55**	42,54	44,91	41,96	32,28
300_06	2	7,59	5,79	6,29	**5,69**	4,38
300_07	0	2,89	0,97	0,68	**0,62**	0,59
300_08	8	9,17	8,95	8,95	**8,82**	8,00
300_09	7	9,05	**8,00**	8,66	8,09	7,46
300_10	21	34,63	32,56	33,39	**32,17**	22,60
400_01	1	**3,01**	3,50	4,20	4,04	2,52
400_02	16	**23,28**	23,82	25,27	23,48	17,37
400_03	9	**11,65**	13,64	14,91	14,25	9,91
400_04	19	21,96	20,38	20,74	**20,24**	19,01
400_05	0	3,48	**2,68**	4,30	3,67	0,01
400_06	0	4,20	1,53	1,49	**1,35**	0,33
400_07	4	**7,65**	8,68	10,68	9,60	5,44
400_08	4	**11,54**	12,67	16,32	13,30	5,30
400_09	5	17,98	16,01	17,30	**15,88**	7,63
400_10	0	4,24	**2,66**	3,72	3,44	0,95

Tableau 3.21 : Récapitulatif des nombres de conflits moyens obtenus par chaque version de l'OCF pour l'ET3

Pour l'ET2, on observe que l'OCF-3D obtient des nombres de conflits moyens égaux ou meilleurs que l'OCF-0 sur toutes les instances, à l'exception du 16_81. L'OCF-EV amène par après de meilleures performances moyennes, à l'exception

d'une légère dégradations pour le 16_81 et de l'instance 26_82, où l'OCF-EV a obtenu une moyenne de 0,01 conflit. Finalement, l'OCF-M améliore les performances pour certaines instances, alors qu'elle entraîne de légères dégradations sur d'autres. Lorsque l'on observe la meilleure moyenne toutes versions confondues pour chaque instance (en gras dans le tableau et excluant l'OCF-0+), on s'aperçoit que l'OCF-EV et l'OCF-M dominent leurs prédécesseurs. Seule l'instance 16_81 fait exception. En résumé, dès l'OCF-3D, les performances de l'algorithme de départ sont égalées ou améliorées et le niveau des résultats sera maintenu pour toutes les versions présentées.

Les résultats de l'ET3 permettent de tirer des conclusions similaires quant aux apports des différentes versions. Considérant que les instances de l'ET3 sont reconnues comme très difficiles à résoudre, les versions proposées offrent des performances très intéressantes. L'OCF-3D présente une fois de plus des résultats qui sont meilleurs que ceux de l'OCF-0, pour 23 des 30 instances. L'OCF-EV améliore les nombres de conflits moyens sur plusieurs instances de 200 et 300 voitures, alors qu'il entraîne généralement des dégradations sur les instances de plus grande taille. Cette réaction peut s'expliquer par la taille grandissante de l'espace de solutions. En effet, si l'espace de solutions est plus vaste, il peut être désavantageux d'employer des mécanismes de diversification trop puissants. En contre partie, c'est pour cet ensemble test que l'OCF-M s'est montré le plus utile, avec des améliorations entre 3% et 96% par rapport à l'algorithme de départ.

Afin de comparer entre elles les versions d'algorithme proposées dans ce travail, les meilleurs nombres de conflits moyens parmi les trois versions (OCF-3D, OCF-EV et OCF-M) sont mis en évidence dans les cases en gris du Tableau 3.20 et du Tableau 3.21. Pour l'ET2, l'exercice ne permet pas de conclure qu'une version domine les autres, étant donné les faibles écarts entre les différentes versions. Cependant, pour l'ET3 on constate que l'OCF-M obtient la majorité des meilleures performances, qu'il partage avec l'OCF-3D qui performe légèrement mieux sur quelques instances. Ces conclusions confirment le choix de retenir l'OCF-M comme algorithme final.

En terminant, lorsqu'on compare les performances de l'OCF-M à celles de l'OCF-0+ (bonifié d'une procédure de recherche locale) présentées dans les tableaux précédents, les conclusions se révèlent encourageantes. Bien qu'il dispose d'une force de calcul inférieure à celle de l'OCF-0+, l'OCF-M atteint des résultats proches de ceux de ce dernier. En effet, les moyennes des deux algorithmes pour l'ET2 sont comparables, à l'exception de l'instance 16_81 pour laquelle l'OCF-M obtient en moyenne 0,61 conflit de plus que l'OCF-0+. Quant à l'ET3, l'écart est un peu plus grand et augmente avec la taille des instances. L'OCF-M arrive à de meilleurs résultats que l'OCF-0+ pour 2 instances de taille 200 et à des résultats égaux sur deux autres instances de même taille, alors qu'on remarque des augmentations pour toutes les autres instances de l'ensemble. Considérant que l'OCF-M est désavantagé par rapport à l'OCF-0+, ces résultats sont encourageants et permettent de supposer que l'OCF-M pourrait surpasser les

résultats de l'OCF-0+ si une procédure de recherche locale semblable lui était ajoutée.

3.7 Conclusion

Nous avons vu au cours de ce chapitre un algorithme d'OCF évoluer. À partir d'un algorithme de départ, différentes modifications ont été faites et conservées d'une version à l'autre.

La première étape a été la modification de la structure de la trace de phéromone afin d'y intégrer les informations relatives à la distance entre les voitures. Cette opération s'est faite en passant par une première phase où une somme des traces était utilisée pour la règle de transition. Les résultats peu concluants de cette première phase ont amené l'ajout d'une troisième dimension à la trace de phéromone pour obtenir une information sur la profitabilité de placer deux classes de voitures en fonction de la distance qui les sépare. Pour l'OCF-3D, l'insertion de cette nouvelle dimension a permis d'encourager la répétition de motifs dans la solution. Cette première modification de l'algorithme de départ s'est montrée plus efficace ou équivalente à l'algorithme de départ sur la grande majorité des instances test.

Suite à ces résultats intéressants, l'OCF-EV a été créé en rendant variables les valeurs données aux exposants dans la règle de transition. Ces variations ont permis une plus grande diversification de la recherche dans l'espace de solutions et une meilleure utilisation des cycles disponibles.

Comme dernière modification, le concept de construction dynamique a été intégré à l'algorithme pour obtenir l'OCF-M, qui présente une construction mixte des solutions. Ce nouveau concept a permis d'améliorer les performances de

l'algorithme. Les temps de calcul nécessaires pour l'exécution de l'OCF-M sont cependant plus élevés que pour les versions précédentes et cette augmentation est encore plus marquée pour les instances de taille plus grande.

Notons finalement que des tests statistiques pourraient être conduits sur les résultats des différents algorithmes afin de mieux évaluer les impacts des bonifications proposées. En conclusion, la démarche empruntée a permis de prouver que les éléments intégrés à l'algorithme d'OCF de départ permettent une amélioration de la performance générale. Les résultats obtenus pour l'ET2 sont très encourageants lorsque comparés à ceux de la littérature présentés au Chapitre 2. En ce qui concerne l'ET3, peu de résultats sont disponibles dans la littérature. Les résultats obtenus sont prometteurs lorsqu'on considère qu'ils sont meilleurs que ceux de l'algorithme de départ sur plusieurs instances.

CHAPITRE 4 :

CONCLUSION

Depuis quelques années, les méta-heuristiques représentent un domaine en pleine expansion. La communauté scientifique accorde un intérêt grandissant à ces méthodes, dont les mécanismes s'adaptent aussi bien aux problématiques théoriques, qu'à celles rencontrées par les praticiens. Depuis leur apparition il y a une quinzaine d'années, les algorithmes de fourmis ont été adaptés à de nombreux problèmes et leur efficacité sur plusieurs d'entre eux a été démontrée. De pair avec ces adaptations, différentes variantes de l'AS ou de l'OCF ont été présentées. Ces variantes proposent des modifications qui touchent principalement l'utilisation de la trace de phéromone dans la règle de transition et ses mises à jour. On constate que peu de gens se sont intéressés à la structure de la trace de phéromone ou au comportement de construction des fourmis.

Quant au POV, la littérature nous le présente comme une référence dans le domaine des CSP et comme un nouveau défi dans celui des méthodes heuristiques. Il s'agit d'un problème NP-difficile au sens fort, qui représente une version simplifiée d'une problématique industrielle concrète et d'actualité.

L'objectif global de ce travail était de « Proposer un algorithme d'OCF efficace en termes de qualité de solution pour la résolution du POV comprenant des éléments adaptés spécifiquement pour ce dernier » et a été atteint à l'aide de trois objectifs spécifiques.

Le premier de ces objectifs était de « Proposer de nouvelles structures de trace de phéromone pour un algorithme d'OCF et en évaluer les impacts sur la qualité des solutions, pour la résolution du POV ». Le fait que les contraintes de

capacité impliquent des blocs de voitures a inspiré l'utilisation de la trace de phéromone sur un horizon. La première phase proposée, l'OCF-H, a produit des résultats peu convaincants. Il a été conclu que la somme de trace utilisée était imprécise, puisqu'elle correspondait à une addition d'éléments confondus. La distance entre les voitures devait être prise en compte pour mieux refléter l'intérêt des différentes combinaisons de classes de voitures. Ces observations ont conduit à l'ajout d'une dimension à la matrice de phéromone, pour obtenir l'OCF-3D. Avec une matrice de phéromone tridimensionnelle, il était possible d'associer chaque paire de classes de voitures aux différentes distances possibles entre elles. Cette nouvelle structure de trace a permis des améliorations considérables par rapport aux performances de l'OCF-H, et a même produit des résultats généralement meilleurs que ceux de l'algorithme de départ. Le premier objectif spécifique a donc été atteint, puisqu'on a établi que l'utilisation de la trace dans l'OCF-H était imprécise et que l'OCF-3D contribuait à la répétition des motifs d'options importants dans la construction de bonnes solutions.

Le second objectif spécifique de ce travail était d'« Identifier des mécanismes permettant de diversifier la recherche de solutions pour un OCF et en évaluer les impacts sur la qualité des solutions, pour la résolution du POV ». Lorsqu'il a été observé que l'OCF-3D entraînait une convergence plus rapide de l'algorithme et donc une sous-utilisation des cycles disponibles, l'ajout d'une certaine variabilité dans les paramètres de décision a été envisagé. Les exposants utilisés dans la règle de transition qui donnent l'importance relative des différents termes de trace

de phéromone ou de visibilité locale ont été choisis pour deux raisons majeures. Ils sont parmi les seuls paramètres d'OCF dont les valeurs ont été montrées comme dépendantes du problème par l'expérience de la communauté scientifique, et il n'existe pas de méthode efficace et rapide pour en régler les valeurs. On a donc établi des bornes à l'intérieur desquelles les valeurs de ces exposants devaient varier de manière régulière au cours de l'exécution. La version d'OCF en résultant, l'OCF-EV, a permis une amélioration des performances de l'OCF-3D. En observant le comportement de l'algorithme et l'augmentation générale des cycles moyens auxquels la meilleure solution est trouvée, on a conclu à une plus grande diversification de la recherche. L'ajout des exposants variables contribue donc à l'atteinte du second objectif spécifique.

Le troisième et dernier objectif spécifique était de « Proposer un nouveau mode de construction des solutions dans un algorithme d'OCF afin de permettre la modification des éléments déjà fixés et en évaluer les impacts sur la qualité des solutions, pour la résolution du POV ». Les algorithmes de fourmis sont des algorithmes de construction séquentielle qui ne remettent jamais en question les éléments déjà placés pour une solution. Considérant la flexibilité d'une séquence de solution pour le POV, un nouveau mode de construction a été proposé selon lequel les fourmis insèrent les voitures dans une chaîne dynamique : la construction dynamique. Cette méthode de construction prise seule a produit des résultats de moindre qualité que ceux publiés dans la littérature. Cependant, en hybridant le concept de la construction dynamique et la dernière version

séquentielle de l'algorithme, les performances de l'OCF-EV ont pu être améliorées. En effet, l'OCF-M s'est montrée profitable pour l'ET2 et l'ET3. Si l'OCF-M apporte des gains, il en coûte en revanche une augmentation générale des temps de calcul, plus marquée pour les instances de plus grande taille. La construction mixte des solutions présente donc un potentiel intéressant et permet de rencontrer le troisième objectif spécifique.

En résumé, l'objectif global de ce travail a pu être atteint grâce aux effets bénéfiques des trois bonifications proposées. En effet, les performances observées pour la version finale de l'algorithme (OCF-M) sont généralement équivalentes ou supérieures à celles de l'OCF de départ pour l'ensemble des instances test. On observe cependant que l'algorithme de départ demeure plus performant que l'OCF-M sur quelques-unes des instances de plus grande taille, à savoir les instances de 300 et 400 voitures de l'ET3. Nous avons vu que cette baisse de performance peut être attribuable à une trop grande diversité engendrée par l'OCF-EV. Ajoutons que ces résultats s'obtiennent dans des temps de calcul raisonnables, même s'il ne s'agissait pas d'une priorité.

Parmi les points forts du présent travail, soulignons le caractère novateur des éléments proposés. Les bonifications réalisées touchent la structure et l'utilisation de la trace de phéromone et la logique de construction des fourmis, qui sont des éléments peu abordés dans la littérature. S'ajoute à cette originalité le constat que la résolution du POV à l'aide d'un OCF est peu courante. De plus, dans le cas des

deux premières bonifications présentées (l'OCF-3D et l'OCF-EV), les concepts introduits sont simples et faciles à insérer dans un algorithme de fourmi existant.

Ce travail comporte également des limites. Notons que les deux premiers ensembles d'instances test choisis, incontournables dans le domaine, ont fait l'objet de nombreux travaux. Les résultats rencontrés dans la littérature laissent peu de place à l'amélioration, compte tenu de l'excellence des performances atteintes avec l'expérience de la communauté scientifique. Quant au troisième ensemble, la comparaison avec la littérature est difficile étant donné le peu de résultats disponibles. Le présent travail vient donc s'insérer dans une lignée de travaux qui auront le potentiel d'améliorer la connaissance des mécanismes du problème théorique, en passant par l'étude de ces nouvelles instances. Notons également que les algorithmes proposés sont adaptés pour un problème spécifique. En conséquence, les impacts des nouveaux concepts sur d'autres problèmes du même type et même pour d'autres instances de POV ne sont pas mesurés.

Certaines idées générées au cours de ce travail, mais qui n'ont pu être concrétisées, présentent un potentiel intéressant. Premièrement, une analyse plus détaillée du comportement des fourmis avec la trace de phéromone à trois dimensions serait pertinente. Une étude des instances utilisées serait également intéressante pour expliquer les bonnes performances de l'OCF-3D sur certaines instances et ses résultats moins concluants sur d'autres. De plus, considérant que la demande n'est pas la même pour chaque classe de voitures et que les mises à

jour sont faites sans tenir compte de la fréquence des combinaisons de classes dans la séquence, des quantités plus importantes de phéromone sont observées pour les combinaisons où interviennent les classes avec le plus grand nombre de voitures. L'impact de ces chances inégales d'accumulation n'a pas été calculé. Il serait donc pertinent de conduire une étude plus approfondie afin d'en connaître l'ampleur. Deuxièmement, il aurait été intéressant de tester différentes variantes de la construction mixte. Par exemple, au lieu d'insérer une voiture en mode dynamique, la fourmi pourrait plutôt remplacer une voiture déjà placée, ou même retirer des voitures sans les remplacer. Les conditions de passage en mode dynamique pourraient également être modifiées pour obtenir, par exemple, une majorité des voitures placées en mode séquentiel, puis un passage en mode dynamique pour la fin de la construction. En ce qui concerne les constructions dynamique et mixte, les possibilités sont nombreuses.

Au cours de ce travail, il a été question d'un transfert technologique vers une version industrielle du problème, dont une variante a été formulée au Chapitre 2. L'adaptation de l'algorithme est effectivement possible, moyennant certaines modifications. La structure de la trace de phéromone à 3 dimensions est transférable sans difficulté. Cependant, on devra considérer que des contraintes de capacité couvrent de plus grands blocs de voitures et sont plus nombreuses que celles du problème théorique. Il serait lourd et peut être moins approprié de considérer une trace de phéromone sur un horizon trop grand. De leur côté, les exposants variables représentent un concept général qui est adaptable à n'importe

quel problème résolu par un algorithme de fourmi, et le POV+ n'y fait pas exception. Quant à la construction mixte des solutions, il est clair que la liste des positions candidates devra être établie de manière plus restrictive. En effet, pour la version industrielle du problème, la quantité de positions candidates en mode dynamique sera beaucoup plus importante et ce, pour les deux conditions de passage. De plus, lorsqu'une voiture sera insérée, il est probable qu'elle entraîne une détérioration sur les objectifs d'ordre inférieur. Il sera peut-être pertinent d'ajouter des mécanismes de réparation des solutions, notamment pour les rafales de couleurs. En résumé, la grande taille des instances, les intervalles plus importants couverts par les contraintes de capacité, ainsi que la considération de plusieurs objectifs contradictoires seront les principales sources de difficulté.

Finalement, comme ce travail de recherche cible un domaine actif de la recherche opérationnelle et de l'informatique, quelques avenues de recherche futures à plus long terme sont suggérées.

Il serait intéressant d'aborder le problème industriel dans une version multi-objectifs au sens habituel du terme. Le POV+ présenté dans ce travail est un problème multi-objectifs sans compensation possible entre les objectifs. Cette manière de traiter le problème n'est pas unanime dans l'industrie ; certains producteurs utilisent des méthodes qui cherchent le meilleur compromis entre les objectifs, appliquant les principes de l'*ensemble Pareto-optimal*. Dans cette optique, certaines modifications sont envisageables, telle l'utilisation d'un matrice de phéromone pour chaque type de contraintes.

Évaluer l'impact de l'ajout d'une procédure de recherche locale à l'algorithme pourrait être à considérer étant donné la profitabilité observée pour de nombreuses méta-heuristiques. L'expérience n'a pas été tentée pour le présent travail dans le but d'avoir une idée exacte de l'impact des bonifications apportées.

Les auteurs d'algorithmes de fourmis adaptent presque toujours le problème à la matrice bidimensionnelle originale proposée pour le TSP. La modification de la structure de trace présentée pour ce travail permet un pas vers l'adaptation de la phéromone à des problématiques précises. De manière générale, travailler sur de nouvelles structures de trace de phéromone pour la résolution de problèmes spécifiques représente un bon potentiel pour des recherches futures.

À l'ère des algorithmes parallèles, il pourrait être intéressant de concevoir une version parallèle de la méthode présentée. Considérant les limites de temps et l'effort de calcul demandé par la construction dynamique des solutions, la construction mixte y trouverait un avantage de taille. Dans une autre optique, une architecture parallèle permettrait à des colonies fonctionnant sur différents modes de construction d'échanger de l'information via une matrice de phéromone globale.

Un projet d'hybridation entre les algorithmes de fourmis et les méthodes de CSP serait également envisageable. Dans les deux domaines, et dans des buts différents, les algorithmes ont atteint d'excellents résultats et des niveaux de perfectionnement impressionnants. Il serait à considérer d'unir les forces et les compétences des deux spécialités afin de concevoir des algorithmes encore plus efficients.

En terminant, ce travail n'a pas la prétention d'être une fin en soi. Il représente plutôt un premier pas sur un terrain encore à explorer. Il encourage la poursuite de la recherche et l'approfondissement des concepts amenés.

[BAL'81] M. Ball et M. Magazine (1981). *The design and analysis of heuristics.* Networks, Vol. 11, p. 215-219.

[BAT'94] R. Battiti et G. Tecchiolli (1994). *The reactive tabu search.* ORSA Journal on Computing, Vol. 6 (2), p. 126-140.

[BLA'98] P. E. Black (1998). *Dictionary of Algorithms and Data Structures*, http://www.nist.gov/dads/.

[BOI'05] S. Boivin (2005). Résolution d'un problème de satisfaction de contraintes pour l'ordonnancement d'une chaîne d'assemblage automobile. Université du Québec à Chicoutimi: Chicoutimi. p. 118.

[BOT'98] H. M. Botee et E. Bonabeau (1998). *Evolving Ant Colony Optimization.* Advanced Complex Systems, Vol. 1, p. 149-159.

[BUK'02] J. Bukchin, E. M. Dar-El et J. Rubinovitz (2002). *Mixed model assembly line design in a make-to-order environment.* Computerers & Industrial Engineering, Vol. 41 (4), p. 405-421.

[BUL'98] B. Bullnheimer, G. Kotsis et C. Strauss (1998). *Parallelization strategies for the Ant System.* Dans *High Performance Algorithms and Software in Nonlinear Optimization*, R. De Leone, A. Murli, P. Pardalos et G. Toraldo, Editors, Kluwer Academic Publishers: Dordrecht, NL, p. 87-100.

[BUL'99a] B. Bullnheimer, R. F. Hartl et C. Straub (1999a). *A new rank based version of the Ant System - A compurational study.* Central European Journal for Operations Research and Economics, Vol. 7 (1), p. 25-38.

[BUL'99b] B. Bullnheimer, R. F. Hartl et C. Strauss (1999b). *An Improved Ant system Algorithm for the Vehicle Routing Problem.* Annals of Operations Research, Vol. 89, p. 319-328,.

[BUL'99c] B. Bullnheimer, R. F. Hartl et C. Strauss (1999c). Applying the Ant System to the Vehicle Routing Problem. Dans Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Voss S., Martello S., Osman I.H. et Roucairol C., Editors: Kluwer, Boston.

[BUR'97] R. E. Burkardy, S. E. Karischz et F. Rendl (1997). *QAPLIB - A Quadratic Asignment Problem Library.* Journal of Global Optimization, Vol. 10, p. 391-403.

[CHE'92] T.-L. Chew, J.-M. David, A. Nguyen et Y. Tourbier (1992). Solving constraint satisfaction problems with simulated annealing; the car sequencing problem revisited. Dans le proceeding de 12th International Conference on Artificial Intelligence, Expert Systems and Narual Language, Avignon, France, p. 405-416.

[COL'92a] A. Colorni, M. Dorigo et V. Maniezzo (1992a). *An investigation of some properties of an "Ant algorithm"*. Dans le proceeding de *Parallel problem solving from nature conference (PPSN92)*, Brussels, Belgique, Elsevier Publishing, p. 509–520.

[COL'92b] A. Colorni, M. Dorigo et V. Maniezzo (1992b). *Distributed Optimization by Ant Colonies.* Dans le proceeding de *ECAL91 - European Conference on Artificial Life*, Paris, France, Elsevier Publishing, p. 134-142.

[CUN'01] V.-D. Cung, S. Martins, C. Ribeiro et C. Roucairol (2001). *Strategies for the parallel implementation of metaheuristics.* Dans *Essays and Surveys in Metaheuristics*, C.C. Ribeiro et P. Hansen, Editors, Kluwer Academis Pub, p. 263-308.

[DAV'94] A. Davenport et E. Tsang (1994). GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. Dans le proceeding de AAAI, p. 325-330.

[DAV'99] A. Davenport et E. Tsang (1999). Solving constraint satisfaction sequencing problems by iterative repair. Dans le proceeding de First international conference on the practical applications of constraint technologies and logic programming (PACLP), p. 345-357.

[DEN'03] B. Denby et S. L. Hegarat-Mascle (2003). *Swarm intelligence in optimisation problems.* Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Vol. 502 (2-3), p. 364-368.

[DIC'97] G. DiCaro et M. Dorigo (1997). *AntNet: A Mobile Agents Approach to Adaptive Routing*, Rapport technique no. IRIDIA/97-12. Université Libre de Bruxelles: Belgium.

[DIC'98a] G. DiCaro et M. Dorigo (1998a). *Mobile Agents for Adaptive Routing.* Dans le proceeding de *31st Hawaii International Conference on System*, Los Alamitos, CA, IEEE Computer Society Press, p. 74-83.

[DIC'98b] G. DiCaro et M. Dorigo (1998b). *AntNet: Distributed Stigmergetic Control for Communications Networks.* Journal of Artificial Intelligence Research (JAIR), Vol. 9, p. 317-365.

[DIN'88] M. Dincbas, H. Simonis et P. v. Hentenryck (1988). *Solving the car-sequencing problem in constraint logic programming.* Dans le proceeding de *ECAI-88*, p. 290-295.

[DOR'91a] M. Dorigo, V. Maniezzo et A. Colorni (1991a). *Ant System: An Autocatalytic Optimizing Process*, Rapport technique no. 91-016. Dipartimento di Elettronica e Informazione, Politecnico di Milano: Milano, Italy.

[DOR'91b] M. Dorigo, V. Maniezzo et A. Colorni (1991b). *Positive feedback as a search strategy*, Rapport technique no. 91-016. Dip. Elettronica, Politecnico di Milano, Italy.

[DOR'96a] M. Dorigo, A. Colorni et V. Maniezzo (1996a). *Ant System: Optimization by a Colony of Cooperating Agents.* IEEE transactions on Systems, Man, and Cybernetics, Part-B, Vol. 26 (1), p. 1-13.

[DOR'96b] M. Dorigo et L. M. Gambardella (1996b). *A Study of Some Properties of Ant-Q.* Dans le proceeding de *PPSN IV-Fourth International Conference on Parallel Problem Solving From Nature*, Berlin, Springer-Verlag, Germany, p. 656-665.

[DOR'96c] M. Dorigo, V. Maniezzo et A. Colorni (1996c). *The Ant System : Optimization by a colony of cooperating agents.* Cybernetics–Part B, Vol. 26 (1), p. 1-13.

[DOR'97a] M. Dorigo et L. M. Gambardella (1997a). *Ant Colonies for the Traveling Salesman Problem*, Rapport technique no. TR/IRIDIA/1996-3. IRIDIA, Université Libre de Bruxelles.

[DOR'97b] M. Dorigo et L. M. Gambardella (1997b). Ant Colony system: A Cooperative learning approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, Vol. 1 (1), p. 53-66.

[DOR'99a] M. Dorigo et G. D. Caro (1999a). *Chapter 2 : The Ant Colony Optimization Meta-Heuristic.* Dans *New Ideas in Optimization*, D. Corne, M. Dorigo et F. Glover, Editors: London, p. 11-32.

[DOR'99b] M. Dorigo, G. D. Caro et L. M. Gambardella (1999b). *Ant Algorithms for Discrete Optimization.* Artificial Life, Vol. 5 (2), p. 137-172.

[EGG'03] J. Eggers, D. Feillet, S. Kehl, M. Oliver Wagner et B. Yannou (2003). *Optimization of the keyboard arrangement problem using an Ant Colony algorithm.* European Journal of Operational Research, Vol. 148 (3), p. 672-686.

[FRI'96] B. Freisleben et P. Merz (1996). New Genetic Local Search Operators for the Traveling Salesman Problem. Dans le proceeding de The Fourth Conference on Parallel Problem Solving from Nature (PPSN), Berlin, Springer, p. 890-899.

[FRE'92] E. C. Freuder et R. J. Wallace (1992). *Partial constraint satisfaction.* Artificial Intelligence, Vol. 58 (1-3 (Special Volume on Constraint Based Reasoning)), p. 21-70.

[GAG'01] C. Gagné, M. Gravel et W. Price (2001). A look-ahead addition to the ant colony optimization metaheuristic and its application to an industrial scheduling problem. Dans le proceeding de The 4th Metaheuristics International Conference, Porto, Portugual,

[GAG'02a] C. Gagné, M. Gravel et W. L. Price (2002a). Algorithme d'optimisation par colonie de fourmis avec matrices de visibilité multiples pour la résolution d'un problème d'ordonnancement industriel. INFORMS Journal on Computing, Vol. 40 (2), p. 259-276.

[GAG'02b] C. Gagné, W. Price et M. Gravel (2002b). Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence dependent setup times. Journal of the Operational Research Society, Vol. 53 (8), p. 895-906.

[GAG'05] C. Gagné, M. Gravel et W. L. Price (2005). *Solving real car sequencing problems with ant colony optimization.* European Journal of Operational Research (à paraître).

[GAM'95] L. M. Gambardella et M. Dorigo (1995). Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem. Dans le proceeding de ML-95, Twelfth International Conference on Machine Learning, Tahoe City, CA, p. 252-260.

[GAM'96] L. M. Gambardella et M. Dorigo (1996). Solving Symmetric and Asymmetric TSPs by Ant Colonies. Dans le proceeding de IEEE Intenational Conference on Evolutionary Computation, IEEE-EC 96, Nagoya, Japan, p. 622-627.

[GAM'97] L. M. Gambardella et M. Dorigo (1997). *HAS-SOP: An Hybrid Ant System for the Sequential Ordering Problem*, Rapport technique no. IDSIA 97-11. IDSIA: Lugano, Switzerland.

[GAM'99] L. M. Gambardella, É. Taillard et M. Dorigo (1999). *Ant Colonies for the Quadratic Assignment Problem.* Journal of the Operational Research Society, Vol. 50, p. 167-176.

[GEN'98] I. P. Gent (1998). *Two Results on Car-sequencing Problems*, Rapport technique no. APES-02-1998. University of St Andrews, School of Computer Science, APES Group.

[GEN'99] I. P. Gent et T. Walsh (1999). CSPlib: A benchmark library for constraints. Dans le proceeding de The Fifth International Conference on Principles and Practice of Constraint Programming, Alexandria, Virginia, p. 480-481.

[GLO'89] F. Glover (1989). *Tabu Search - Part I.* ORSA Journal on Computing, Vol. 1 (3), p. 190-206.

[GLO'90] F. Glover (1990). *Tabu Search - Part II.* ORSA Journal on Computing, Vol. 2 (1), p. 4-32.

[GOS'89] S. Goss, S. Aron, J. L. Deneubourg et J. M. Pasteels (1989). *Self-organized shortcuts in the Argentine ant.* Naturwissenschaften, Vol. 76, p. 579-581.

[GOT'03] J. Gottlieb, M. Puchta et C. Solnon (2003). A study of greedy, local search ans ant colony optimization approches for car sequencing problems. Computer Science, (2611), p. 246-257.

[GRA'02] M. Gravel, W. L. Price et C. Gagné (2002). *Scheduling continuous casting of aluminium using a multiple-objective ant colony optimization metaheuristic.* European Journal of Operational Research, Vol. 143 (1), p. 218-229.

[GRA'05] M. Gravel, C. Gagné et W. Price (2005). *Review and comparison of three methods for the solution of the car-sequencing problem.* Journal of the Operational Research Society, accepté pour publication en octobre 2004 (à paraître).

[GUN'02a] M. Guntsch et M. Middendorf (2002a). A Population Based Approach for ACO. Dans Applications of Evolutionary Computing, Proceedings de EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim, Stefano

Cagnoni and Jens Gottlieb and Emma Hart and Martin Middendorf and Gunther Raidl, Editor Springer-Verlag: Kinsale, Ireland, p. 71-80.

[GUN'02b] M. Guntsch et M. Middendorf (2002b). Applying population based ACO to dynamic optimization problems. Dans le proceeding de Ant Algorithms: 3rd International Workshop, ANTS2002, Springer-Verlag, p. 111-122.

[HAO'99] J. K. Hao, P. Galinier et M. Habib (1999). *Metaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes.* Revue d'Intelligence Artificielle, Vol. 13 (2), p. 283-324.

[HOL'75] J. H. Holland (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press.

[HYU'98] C. J. Hyun, Y. Kim et Y. K. Kim (1998). *A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines.* Computers Operation Researches, Vol. 25 (7/8), p. 675-690.

[KIR'83] S. Kirkpatrick, J. C. D. Gelatt et M. P. Vecchi (1983). *Optimization by Simulated Annealing.* Science, Vol. 220 (4598), p. 671-680.

[KIS'04] T. Kis (2004). *On the complexity of the car sequencing problem.* Operations Research Letters, Vol. 32 (4), p. 331-335.

[LAG'02] M. Laguna (2002). Global optimization and meta-heuristics. Dans Optimization and Operations Research (Encyclopedia of Life Support Systems (EOLSS)), Ulrich Derigs, Editor: Oxford, UK, p. Theme 6.5, Topic 2.

[LEE'95] J. H. M. Lee, H. F. Leung et H. W. Won (1995). Extending GENET for non-binary constraint satisfaction problems. Dans le proceeding de 7th International Conference on Tools with Articial Intelligence, p. 338-342.

[LOP'00] P. Lopez et F. Roubellat (2000). Chapitre 8 : Ordonnancement sur une ligne multimodèle. Dans Ordonnancement de la production, HERMES, Editor.

[LOV'00] R. H. Lovgren et M. J. Racer (2000). *Algorithms for mixed-model sequencing with due date restrictions.* European Journal of Operational Research, Vol. 120 (2), p. 408-422.

[MAN'99a] V. Maniezzo (1999a). Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. INFORMS Journal on Computing, Vol. 11 (4), p. 358-369.

[MAN'99b] V. Maniezzo et A. Carbonaro (1999b). Ant Colony Optimization : An overview. Dans le proceeding de MIC'99 III Metaheuristics International Conference, Brézil,

[MAN'99c] V. Maniezzo et A. Colorni (1999c). *The Ant System Applied to the Quadratic Assignment Problem.* IEEE Transactions on Knowledge and Data Engineering, Vol. 11 (5), p. 769-778.

[MER'00] D. Merkle et M. Middendorf (2000). An ant algorithm with a new pheromone evaluation rule for total tardiness problems. Dans le proceeding de Real-World Applications of Evolutionary Computing, Springer, Berlin, p. 287-296.

[MIN'93] S. Minton, M. D. Johnson, A. Philips et P. Laird (1993). *Minimizing conflicts: a heuristic repair method for constraint satisfaction problems and scheduling problems.* Journal of Artificial Intelligence Research, Vol. 58 (1), p. 161-205.

[NGU'03] A. Nguyen (2003). Challenge ROADEF' 2005 : Problème du "Car Sequencing".

[OSM'96] I. H. Osman et G. Laporte (1996). *Metaheuristics: A bibliography.* Annals of Operations Research, Vol. 65, p. 513-623.

[PAR'86] B. D. Parrello, W. C. Kabat et L. Wos (1986). *Job-shop scheduling using automated reasoning: a case study of the car-sequencing problem.* Journal of Automated Reasoning, Vol. 2, p. 1-42.

[REG'97] J.-C. Régin et J.-F. Puget (1997). A Filtering Algorithm for Global Sequencing Constraints. Dans le proceeding de Principles and Practice of Constraint Programming LNCS 1330, Springer, p. 32-46.

[ROB'49] J. B. Robinson (1949). *On the Hamiltonian game (a traveling-salesman problem).* RAND Research Memorandum RM-303.

[SCH'04] B. Scheuermann, K. So, M. Guntsh, M. Middendorf, O. Diessel, H. ElGindy et H. Schmeck (2004). *FPGA implementation of population-based ant colony optimization.* Applied soft computing, Vol. 4, p. 303-322.

[SCH'97a] R. Schoonderwoerd, O. Holland et J. Bruten (1997a). *Ant-like Agents for Load Balancing in Telecommunications Networks*. Dans le proceeding de *Agents'97*, Marina del Rey, CA, ACM Press, p. 209-216.

[SCH'97b] R. Schoonderwoerd, O. Holland, J. Bruten et L. Rothkrantz (1997b). *Ant-based Load Balancing in Telecommunications Networks.* Adaptive Behavior, Vol. 5 (2), p. 169-207.

[SMI'97] B. Smith (1997). Succeed-first or Fail-first: A Case Study in Variable and Value Ordering Heuristics. Dans le proceeding de PACT'97, Third International Conference on the Practical Applications of Constraint Technology, p. 321-330.

[SOL'00] C. Solnon (2000). Solving Permutation Constraint Satisfaction Problems with Artificial Ants. Dans le proceeding de The 14th European Conference on Artificial Intelligence, Amsterdam, The Netherlands, p. 118-122.

[SOL'02] C. Solnon (2002). *Ants can solve Constraint Satisfaction Problems.* EEE Transactions on Evolutionary Computation, Vol. 6 (4), p. 347-357.

[STU'97a] T. Stützle et H. Hoos (1997a). Improvements on the Ant System: Introducing the MAX-MIN Ant System. Dans le proceeding de ICANNGA97 - Third International Conference on Artificial Neural Networks and Genetic Algorithms, University of East Anglia, Norwich, UK, Wien: Springer Verlag,

[STU'97b] T. Stützle et H. Hoos (1997b). The MAX-MIN Ant System and local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Global Optimization. Dans le proceeding de 2nd Metaheuristics International Conference (MIC-97), Sophia-Antipolis, France,

[STU'99] T. Stützle et M. Dorigo (1999). *ACO Algorithms for the Traveling Salesman Problem*. Dans *Evolutionary Algorithms in Engineering and Computer Science*, K. Miettinen, M. Makela, P. Neittaanmaki et J. Periaux, Editors, Wiley.

[STU'00] T. Stützle et H. H. Hoos (2000). *MAX-MIN ant system.* Future Generation Computer Systems Journal, Vol. 16 (8), p. 889-914.

[STU'02] T. Stützle et S. Linke (2002). *Experiments with variants of ant algorithms.* Mathware & Soft Computing, Vol. 7, p. 193-207.

[TAI'97a] É. Taillard et L. M. Gambardella (1997a). An Ant Approach for Structured Quadratic Assignment Problems. Dans le proceeding de 2nd Metaheuristics International Conference (MIC-97), Sophia-Antipolis, France,

[TAI'97b] É. Taillard et L. M. Gambardella (1997b). *Adaptive memories for the quadratic assignment problem*, Rapport technique no. IDSIA-87-97. IDSIA: Lugano, Suisse.

[TAL'01] E.-G. Talbi, O. Roux, C. Fonlupt et D. Robillard (2001). *Parallel ant colonies for the quadratic assignment problem.* Future Generation Computer Systems, Vol. 17 (4).

[TSA'93a] E. Tsang (1993a). Chapter 1 : Introduction. Dans Foundations of constraint satisfactionAcademic Press, p. 1-30.

[TSA'93b] E. Tsang (1993b). Chapter 2 : CSP Solving - An overview. Dans Foundations of constraint satisfactionAcademic Press, p. 31-52.

[TSA'93c] E. Tsang (1993c). Chapter 3 : Fundamental concepts in the CSP. Dans Foundations of constraint satisfactionAcademic Press, p. 53-78.

[WAR'95] T. Warwick et E. Tsang (1995). *Tackling car sequencing problem using a generic genetic algorithm.* Evolutionary Computation, MIT Press, Vol. 3 (3), p. 267-298.

[WAT'92] C. J. C. H. Watkins et P. Dayan (1992). *Technical Note Q-Learning.* Machine Learning, Vol. 8, p. 279-292.

[ZAN'81] S. H. Zanakis et J. R. Evans (1981). *Heuristic "optimization": why, when, and how to use it.* INTERFACES, Vol. 11 (5), p. 84-91.

[ZAN'89] S. H. Zanakis, J. R. Evans et A. A. Vazacopoulos (1989). *Heuristic methods and applications : A categorized survey.* European Journal of Operational Research, Vol. 43, p. 88-110.