Université de Montréal

Système de colonie de fourmis GENI pour le problème du voyageur de commerce

par

François-Xavier Le Louarn
Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

mars 2000

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé :

Système de colonie de fourmis GENI pour le problème du voyageur de commerce

présenté par

François-Xavier Le Louarn

a été évalué par un jury composé des personnes suivantes :

Président-rapporteur	Jacques Ferland
Membre du jury	Yoshua Bengio
Directeur de recherche	Michel Gendreau
Codirecteur de recherche	Jean-Yves Potvin

Mémoire accepté le ...

Sommaire

Le système de colonie de fourmis est une méta-heuristique récente présentée par Dorigo *et al.* destinée à la résolution de problèmes d'optimisation combinatoire. Elle résulte de l'observation de la faculté des fourmis réelles à trouver coopérativement un court chemin entre leur fourmilière et une source de nourriture. L'abstraction de ce comportement puis son application au problème du voyageur de commerce (PVC) a mené à un système de fourmis artificielles animées par l'heuristique du plus proche voisin (PPV). L'algorithme résultant est appelé ACS (Ant Colony System). Ce document se veut la présentation d'une nouvelle heuristique basée sur le ACS dans laquelle les fourmis ne construisent plus leur solution grâce au PPV mais en utilisant plutôt l'heuristique GENI, développée par Gendreau, Hertz et Laporte. Nous voulons tester par là 1° le caractère générique de la méta-heuristique inspirée des fourmis et 2° l'effet de l'adjonction d'une mémoire à l'heuristique GENI.

Nous entamons ce document par la revue du PVC et des différents algorithmes que nous manipulerons ; manipulations qui nous permettrons d'obtenir, au chapitre 3, le GACS (GENI ACS). Le chapitre 4 présente les résultats obtenus par le GACS sur quelques instances de PVC. La conclusion de ce mémoire synthétise nos efforts, répond aux questions que nous nous posions en début de document et en pose de nouvelles.

mots-clés : système de colonie de fourmis, système multi-agents, coopération, problème du voyageur de commerce, recherche opérationnelle

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens à remercier mes deux directeurs de mémoire, Michel Gendreau et Jean-Yves Potvin, pour les pistes indiquées, les conseils prodigués et le soutien apporté tout au long de ce projet.

Sincères remerciements aussi à Mathieu Vézeau pour son temps.

Merci à Maxime, ma mère et mon père.

Merci Christelle.

Chapitre 1

Introduction

Le vivant, pour aussi passionnant qu'il soit par lui-même, n'a de commun avec la recherche opérationnelle que la complexité des problèmes à résoudre. C'est pour cette seule mais cruciale similitude que les méthodes de résolution d'inspiration « naturelle » gagnent en popularité depuis une vingtaine d'années parmi la communauté des chercheurs en optimisation combinatoire.

Les fourmis, a-t-on remarqué, parviennent ensemble à trouver un chemin assez direct de la fourmilière à une source de nourriture. Puisque l'on cherche toujours raccourcir les distances parcourues par le fameux voyageur de commerce, pourquoi ne pas extraire de leurs méthodes celles qui peuvent être utiles à la recherche opérationnelle ?

La tâche est entreprise depuis une dizaine d'années par une communauté de chercheurs et il en a résulté plusieurs algorithmes de recherche locale d'efficacité variable. D'affinements en raffinements, ces algorithmes ont été de mieux en mieux adaptés aux problèmes d'optimisation, incorporant les derniers avancements du domaine. Non seulement l'efficacité de ces algorithmes s'en est-elle ressentie, égalant parfois celle des meilleurs, mais la métaphore des fourmis a jeté un éclairage nouveau sur la résolution des problèmes étudiés, le problème du voyageur de commerce en premier.

Quelle est donc cette nouvelle perspective ? Les fourmis accumulent de l'information sur les *meilleures arêtes* d'un chemin plutôt que sur l'*intégralité* des chemins

pour ensuite reconstruire un nouveau chemin avec ces meilleures arêtes. Deux aspects intéressants : d'abord, l'hypothèse que les meilleurs chemins comportent des arêtes communes ; ensuite, la séquence décomposition-recomposition des solutions. Accessoirement, il est intéressant d'observer l'essence de la coopération, le partage des informations, et sa traduction informatique, la mise en commun d'une mémoire globale.

Le premier aspect n'est pas neuf mais trouve ici une expression nouvelle. Le deuxième se démarque par l'importance donnée à l'algorithme de construction. Généralement, en effet, la construction initiale des solutions n'a que peu d'importance et l'emphase est plutôt mise sur l'optimisation des solutions. Le présent projet se penche sur ce deuxième aspect : est-ce que d'autres algorithmes de construction peuvent utiliser efficacement la pondération des arêtes pour construire de bonnes solutions?

Ce projet est une réponse partielle à cette question. Nous présentons dans ces pages une nouvelle instantiation de la méta-heuristique des colonies de fourmis pour le problème du voyageur de commerce ; la construction des solutions est ici confiée à une heuristique plus efficace que celle utilisée traditionnellement : l'heuristique GENI proposée par Gendreau, Hertz et Laporte remplace l'heuristique du plus proche voisin.

Nous commencerons par situer le problème dans son contexte théorique. La revue de littérature qui constitue le premier chapitre présente et définit le problème du voyageur de commerce ainsi que les différentes stratégies de résolution proposées à ce jour. Nous mettons l'accent sur l'heuristique GENI et, bien entendu, sur la méta-heuristique de colonie de fourmis.

La deuxième partie de ce document constitue le coeur de notre projet, la présentation de l'heuristique du système de colonie de fourmis GENI. Nous y décrivons comment nous avons extrait l'algorithme de construction initial pour le remplacer par GENI. Nous y voyons aussi quelles sont les modifications que nous avons apportées à GENI.

Enfin, nous présentons le résultat de notre manipulation. Nous décrivons par le détail les mécanismes de cette nouvelle heuristique et répondons, enfin, à la question

que nous nous posions plus haut.

Chapitre 2

Revue de littérature

Le sentier sur lequel nous nous engageons est déjà creusé de pas : plusieurs, et non les moindres, se sont mesurés au problème du voyageur de commerce ; l'algorithme GENI a été conçu il y a une dizaine d'année ; quant au système de fourmis, bien que relativement jeune, plusieurs articles y font allusion, voire l'utilisent comme matériau principal. Afin de ne pas être redondant par la suite et pour notre propre édification, nous retraçons ici les connaissances pertinentes à ces trois sujets.

Nous commencerons par présenter le problème du voyageur de commerce et son traitement dans la littérature afin de définir le cadre dans lequel se situe le projet. Nous aborderons ensuite plus particulièrement les deux algorithmes qui seront à la base de l'heuristique que nous présenterons dans les chapitres qui suivent : GENI et le système de fourmis.

2.1 Le problème du voyageur de commerce

Pour prolonger la métaphore, le problème du voyageur de commerce (PVC) consiste à trouver un tour qui permettra au dit voyageur de visiter toutes les villes inscrites à son contrat et de revenir à son point de départ tout en minimisant la distance totale parcourue. La simplicité de l'énoncé est trompeuse car, au vu de la quantité de pu-

blications qui le concernent, le PVC est un problème incontournable de la recherche opérationnelle.

Dans ce qui suit, nous énoncerons le PVC et examinerons ses fondements théoriques, déterminerons les causes de sa popularité et présenterons les solutions proposées à ce jour dans la littérature.

2.1.1 Présentation

La formulation que nous avons donnée précédemment, si elle pique la curiosité, ne situe cependant pas le problème dans un cadre mathématique seyant mieux à son analyse. L'énoncé suivant pose plus précisément le problème :

> Soit $G = (N, A)$ un graphe complet consistant d'un ensemble N de n nœuds et d'un ensemble A d'arêtes. À chaque arête $(i, j) \in A$ reliant les nœuds $i, j \in N$ est associé un coût $c(i, j) = c(j, i)$.
> Le PVC consiste à trouver un cycle de coût minimal passant une et une seule fois par chacun des nœuds de N.

Il est à noter que l'énoncé précédent présente le PVC symétrique, qui est une restriction du problème plus général du PVC asymétrique, pour lequel $c(i, j) \neq c(j, i)$. Nous nous concentrerons, tout au long de ce document, sur le PVC symétrique tel qu'énoncé plus haut. Nous le contraindrons davantage encore en ne nous intéressant qu'au PVC se situant dans le plan euclidien, où l'inégalité du triangle est valable. Cette inégalité indique que l'arête d'un nœud à un autre est de coût nécessairement moins élevé que tout autre chemin passant par un nœud intermédiaire. Le coût entre les nœuds i et j de coordonnées (x_i, y_i) et (x_j, y_j) est calculé comme suit : $c(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Dans la pratique, ce coût est souvent approximé par excès à un nombre entier, afin d'uniformiser les résultats obtenus sur des ordinateurs différents ; la précision des ordinateurs lorsqu'ils manipulent des nombres flottants peut en effet être très variable.

Le PVC (euclidien ou non) est un problème d'optimisation combinatoire NP-dur : non seulement n'a-t-on pas encore trouvé d'algorithme permettant sa résolution en temps polynômial, mais on ne peut pas non plus prouver que $PVC \in NP$[1]. Cependant, un proche dérivé du PVC est élément de NP, le PVC-décision. Le PVC-décision consiste à trouver un cycle Hamiltonien dans un graphe $G = (N, A)$ dont le coût est inférieur à une borne k. Il a été montré qu'il existe un algorithme permettant de résoudre le PVC en temps polynômial si et seulement si il existe un algorithme permettant de résoudre le PVC-décision en temps polynômial [28]. Or, évidemment, il n'a pas encore été trouvé d'algorithme permettant de résoudre de problème NP-complet en temps polynômial.

2.1.2 Un problème classique

Le PVC est, comme nous l'avons mentionné, un problème bien connu et très étudié en recherche opérationnelle. Cet état est attribuable à plusieurs caractéristiques qui, combinées, en font sa spécificité.

D'abord, les manifestations concrètes du PVC sont légion. L'intégration à très grande échelle[2], par exemple, débouche sur le PVC lorsqu'il s'agit de tracer au moindre coût les contacts entre les différentes composantes électroniques [30]. La cristallographie [5], l'ordonnancement des tâches et même la fabrication des cibles pour jeu de fléchettes [13] ne sont que quelques autres exemples des applications dans lesquelles se retrouve le PVC.

Sa « simplicité », ensuite, comparée à d'autres problèmes NP-complets n'est certainement pas étrangère à son succès. Celle-ci se manifeste par la bonne performance d'heuristiques simples, comme nous le verrons à la section 2.1.3. Paradoxalement, c'est sans doute cette raison qui a imposé très tôt le PVC comme *le* problème-jouet tout indiqué pour expérimenter de nouvelles heuristiques. Paradoxal puisque l'apport de ces algorithmes ne peut être que marginal. Cependant, il semble que le PVC

1. Comment, en effet, déterminer si un tour est *le* tour optimal du graphe?
2. Very Large Scale Integration, VLSI

reste un point d'entrée pour les heuristiques en devenir. Et, phénomène commun à plusieurs domaines de recherche, le problème-jouet est intéressant pour la similitude qu'il entretient avec d'autres problèmes plus complexes (le problème de confection de routes, par exemple).

Pour les raisons qui précèdent, le PVC offre un environnement de test riche : résultats pratiques et données sur les expériences précédentes abondent. Cette richesse accentue encore la popularité du PVC, sachant qu'une nouvelle étude doit être comparée avec les précédentes pour avoir quelque valeur. Ce cercle vertueux explique en grande partie pourquoi l'heuristique que nous présentons dans ce document est conçue pour le PVC.

2.1.3 Plusieurs approches

Le PVC a été attaqué selon différents angles, donnant chacun un éclairage nouveau sur le problème et, plus généralement, sur les problèmes d'optimisation combinatoire. Deux types de résolution existent : exacte et approximative. Nous nous concentrerons sur ce dernier type mais le lecteur curieux pourra trouver une introduction aux algorithmes de résolution exacts dans l'article de Laporte [31].

Les algorithmes menant à une approximation de la solution optimale du PVC sont relativement nombreux mais se scindent conventionnellement en trois catégories [26] : ceux qui construisent un cycle, ceux qui améliorent un cycle déjà construit et ceux qui effectuent les deux opérations, plus ou moins simultanément. À ces trois types d'heuristiques s'ajoutent les métaheuristiques qui, elles, encadrent les heuristiques plus simples des trois classes précédentes.

Nous présentons aussi une cinquième catégorie proposée récemment regroupant plusieurs métaheuristiques présentant certains points communs, le principal étant l'utilisation d'une mémoire adaptative.

Construction

Il existe plusieurs façons de construire un cycle à partir des nœuds d'un graphe : insèrer successivement tous les nœuds du graphe, fusionner plusieurs chemins pour finalement construire un cycle Hamiltonien ou enfin bâtir un cycle à partir de l'arbre de recouvrement minimal du graphe.

Algorithmes d'insertion Les algorithmes d'insertion se différencient entre eux 1° par leur façon de sélectionner le prochain nœud à insérer et 2° par leur manière d'insérer le noeud en question. Typiquement, une heuristique d'insertion aura la forme suivante :

1. Sélectionner le prochain nœud à insérer ;
2. Insérer le nœud au moindre coût dans la solution courante et revenir à l'étape 1 si tous les nœuds n'ont pas encore été insérés.

La plus simple d'entre elles consiste simplement à choisir un premier nœud arbitrairement puis à construire progressivement un chemin en choisissant systématiquement l'arête la moins coûteuse dont l'origine est le nœud courant et dont l'autre extrémité est un nœud n'ayant pas encore été visité. Lorsque tous les nœuds ont été visités, le chemin est transformé en cycle en reliant simplement le dernier nœud visité au premier. Cette heuristique est celle du *plus proche voisin* (PPV) mais il en existe beaucoup d'autres qui s'appuient sur d'autres critères de sélection (plus grand angle, choix arbitraire, etc.) ou d'insertion (ajouter à la fin du chemin en construction, insérer entre deux nœuds consécutifs, etc.).

Algorithmes de fusion de routes Les algorithmes de fusion de routes construisent plusieurs routes de dimension inférieure à n et les fusionnent ensuite de façon à minimiser le coût du cycle résultant. L'algorithme Clarke-Wright [8] est une heuristique de ce type. On choisit initialement un nœud d (le *dépôt*) et on suppose que chaque nœud i du graphe est desservi par une route unique (d, i, d). Ensuite, pour chaque

nœud i, on répertorie les gains qu'occasionnerait la fusion des routes (d, i, d, j, d) en (d, i, j, d) où j est un nœud du graphe différent de i et de d. Autrement dit, plutôt que de revenir au dépot d après chaque visite à un nœud, on envisage la possibilité de passer directement à un autre nœud. Cela revient à calculer pour chaque arête (i, j) le gain $g(i, j) = [c(i, d) + c(d, j)] - c(i, j)$. On choisit ensuite les arêtes (i, j) par ordre décroissant de gain tout en n'introduisant pas de nœud de degré supérieur à 2.

Construction d'un tour à partir d'un arbre de recouvrement de coût minimal En plus des algorithmes que nous venons de présenter, il est aussi possible de construire un cycle Hamiltonien de coût raisonable à partir d'un arbre de recouvrement de coût minimal. Après avoir construit cet arbre de recouvrement (en temps polynômial, voir par exemple [2]), on le parcourt en profondeur d'abord ; on élimine ensuite du chemin obtenu les occurences multiples d'un nœud. Si l'on procède ainsi, il a été prouvé que le tour obtenu a un coût inférieur ou égal à $2 \cdot C^{opt}(G)$, où $C^{opt}(G)$ est le coût de la solution optimale du TSP du graphe G. Christofides a montré qu'il était possible de trouver une solution dont le coût est inférieur ou égal à $1,5 \cdot C^{opt}(G)$ [7]. Mentionnons que l'on n'a pas encore trouvé d'algorithme offrant de meilleure garantie.

Amélioration

Certains algorithmes ont pour objectif de fouiller l'espace des solutions. Ils sont inaptes à la création d'une solution mais, une fois initialisés, savent se déplacer dans l'espace des solutions. Ces heuristiques sont dites d'*optimisation locale.*

Les algorithmes les plus simples, nous l'avons déjà mentionné, sont étonnamment efficaces sur le PVC. Parmi ceux-ci, les algorithmes *2-opt* et *3-opt* tiennent le haut du pavé quant au rapport qualité-temps. Ils consistent tous deux en plusieurs permutations simples des arêtes d'un cycle : la procédure 2-opt détruit deux arêtes et reconnecte les quatre extrémités différemment (voir figure 2.1) ; la procédure 3-opt détruit trois arêtes et reconnecte de quatre façons différentes les six extrémités. Ces

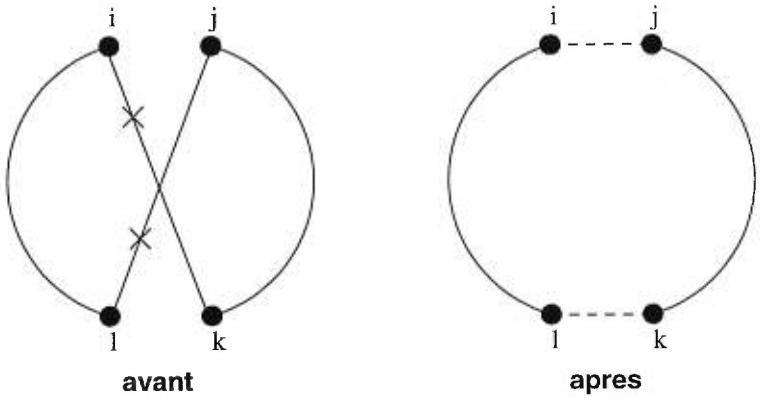

FIG. 2.1 – Échange de (i,k), (j,l) pour (i,j), (k,l). Nous avons indiqué par une croix les arêtes qui disparaissent et tracé en pointillés les nouvelles.

opérations sont reproduites pour toutes les arêtes du cycle et une permutation est conservée si elle mène à une meilleure solution.

Typiquement, l'application de 2-opt à une solution mène ladite solution à 5 % de la borne inférieure de Held-Karp[1] alors que l'application de 3-opt la mène à 3 % [26].

S'ajoutent à ces heuristiques les algorithmes *OR-opt* [36], *2,5-opt* [4] et bien sûr l'algorithme de Lin-Kernighan [33] qui consiste en fait en une procédure r-opt, avec un r variant à mesure de la recherche. Ce dernier algorithme est extrêmement efficace : les solutions trouvées sont aux environs de 2 % de la borne inférieure de Held-Karp.

Hybridation

Les heuristiques hybrides combinent construction et optimisation afin d'obtenir un tour bénéficiant des qualités complémentaires des deux procédures.

CCAO (Convex hull - Cheapest insertion - largest Angle - Or-opt) fut une des premières hybridations réussies [21]. Cet algorithme commence par construire une solution et, après, procède à une amélioration. L'algorithme peut être formulé comme

1. Held et Karp ont proposé un méthode permettant de calculer une borne inférieure sur le coût de la solution optimale d'un problème [22, 23]. Il a été vérifié que cette borne ne peut être inférieure à $(2/3)C^{opt}$ lorsque l'inégalité du triangle est valable.

suit :

1. Initialement, un cycle est construit contenant les nœuds de l'enveloppe convexe[1].
2. Tant qu'un cycle complet n'est pas construit
   (a) Pour tous les nœuds non encore insérés
      i. Calculer le coût du détour que causerait l'insertion de ce nœud
      ii. Calculer l'angle de ce détour
   (b) Insérer dans le cycle le nœud dont l'insertion cause le plus petit détour et dont l'angle de ce détour est le plus grand.
3. Effectuer un optimisation OR-opt sur le cycle trouvé

Une hybridation plus fine fut introduite par Gendreau, Hertz et Laporte [17] avec leur algorithme GENI (GENeralized Insertion) que nous décrivons en détail à la section 2.2.1. Deux particularités permettent à cet algorithme de meilleures performances que CCAO : d'abord, un nœud peut être inséré entre deux nœuds *non consécutifs* ; ensuite, on réoptimise localement après *chaque* insertion et non tout le cycle, en fin d'algorithme. La place était vacante pour une optimisation générale en fin d'algorithme, comme pour CCAO, et elle fut comblée par la procédure US (Unstringing, Stringing), présentée dans le même article. La combinaison de GENI et US, GENIUS, atteint des résultats comparables à ceux des optimisations r-opt. Plus de détails sur ces performances peuvent être trouvés dans la section 2.2.3.

Méta-heuristiques

Le caractère « méta » des méta-heuristiques provient de ce qu'elles offrent un cadre de recherche général qu'il faut adapter au problème considéré. Elles découlent

1. Flood a en effet démontré que les nœuds formant un cycle optimal de PVC euclidien apparaissent dans le même ordre que sur l'enveloppe convexe du graphe[14]. Rappelons que l'enveloppe convexe d'un ensemble de nœuds N est le plus petit ensemble de nœuds qui, lorsque reliés, « encercle » tous les noeud de N.

souvent d'observations faites dans la nature (recuit simulé, algorithmes génétiques, système de fourmis) ou sur l'être humain (recherche avec tabous).

Recherche avec tabous La recherche avec tabous est calquée sur la manière dont procède un être humain pour rechercher une solution à un problème donné. L'idée proposée par Glover consiste à se déplacer autour d'une bonne solution afin d'en trouver une meilleure encore [19, 20]. Dans le cas du PVC, il s'agit d'abord de construire une solution (grâce au PPV, par exemple) et de l'optimiser. Ensuite, il faut chercher dans le voisinage de cette solution la meilleure qui soit[1] et s'y déplacer, *même si cela entraîne une dégradation de la valeur de l'objectif.* De toute façon, la meilleure solution trouvée jusque là est gardée en mémoire. Il est simplement exigé de ne pas revenir sur une solution déjà visitée puisque ces dernières sont « taboues ».

Ce type de recherche locale est particulièrement intéressant par son côté générique : l'algorithme d'optimisation locale peut être 2-opt, 3-opt, OR-opt, Lin-Kernighan, etc. De plus, la recherche avec tabous pose pour hypothèse que les meilleures solutions sont groupées, ce qui est avéré empiriquement.

Recuit simulé Le recuit simulé ressemble un peu à la recherche avec tabous dans la mesure où lui aussi permet de sortir des minima locaux : ici aussi, une solution voisine a une chance d'être acceptée même si elle est moins bonne que la solution initiale. Ce facteur aléatoire est contrôlé par un paramètre, la *température*, diminuant avec le temps. Plus la température est basse, moins la chance de sauter vers une solution de moindre qualité est élevée. De fil en aiguille, les mouvements contre-optimaux se font plus rares et on converge vers un optimum local. Les mouvements peuvent être, comme pour la recherche avec tabous, de type différent (2-opt, 3-opt, etc.).

Système de fourmis Le système de fourmis, que nous aborderons plus en détail dans la section 2.3, tente quant à lui de représenter une portion de l'espace des solutions et d'extraire de cette représentation des solutions intéressantes. Pour le

1. Une solution est voisine si un mouvement simple (2-opt, par exemple) permet d'y accéder.

PVC, par exemple, la région est représentée ici par l'ensemble du graphe dont les arêtes sont pondérées en fonction de leur importance, c'est-à dire de leur fréquence d'occurrence dans les bonnes solutions, par des « fourmis » artificielles. Puisque la représentation est relativement « informe » dans la mesure où elle agglomère plusieurs solutions, il faut faire appel à une heuristique de construction pour les extraire de cette représentation. Dans les moutures actuelles [11, 6], cette heuristique est celle du plus proche voisin, légèrement modifiée (voir la section 2.3.2). Notons aussi qu'une optimisation locale peut être effectuée une fois les solutions construites. Il est pour l'instant question des heuristiques 2-opt ou 3-opt mais il est prévu d'examiner l'effet d'un Lin-Kernighan.

Réseaux de neurones La représentation de l'espace des solutions est une question qui revient aussi dans les heuristiques basées sur les réseaux de neurones. Dans le cas des réseaux de neurones construits selon l'architecture Hopfield-Tank, n^2 neurones sont connectés et leurs liens sont pondérés pour représenter les nœuds du graphe et leur ordre de visite [25]. Il semble toutefois que cette représentation demande une puissance de calcul considérable afin seulement de se placer dans l'espace des solutions adéquat. Trouver des optima locaux demande évidemment une puissance encore supérieure. Les réseaux de neurones basés sur des représentations géométriques du problème, Elastic Net et Self-Organizing Map trouvent généralement des solutions de meilleure qualité en moins de temps. Une étude exhaustive de ces algorithmes se trouve dans l'article de Potvin [37].

Algorithmes génétiques Les algorithmes génétiques, eux, caractérisent une ou des régions par les diverses solutions que l'on peut y trouver. Initialement, la population de solutions représente une portion relativement large de l'espace. Par une succession de croisements entre solutions, l'algorithme rétrécit la région considérée. Un opérateur de mutation permet toutefois de conserver une certaine diversité et, donc, de ne pas focaliser trop rapidement sur une région particulière. Pour chaque solution, une optimisation locale est appliquée, de façon à ne considérer que les meilleures solutions. En effet, le croisement de bonnes solutions, s'il mène à des solutions intéressantes, ne

trouve pas nécessairement les *meilleures* solutions du voisinage et, donc, ne remplace pas une optimisation locale.

Dans la pratique, les algorithmes génétiques parviennent à des résultats intéressants, en particulier quand l'opérateur de mutation ainsi que l'algorithme d'optimisation locale sont adéquatement choisis. Martin, Otto et Felten [34] ont développé un algorithme génétique dégénéré n'ayant pas d'opérateur de croisement, qui est localement optimisé par 3-opt et dont la population n'évolue que grâce à l'opérateur de mutation, un mouvement 4-opt appelé *double bridge*. Johnson a repris l'idée et développé un algorithme utilisant Lin-Kernighan comme algorithme d'optimisation locale. Cela a donné l'algorithme de Lin-Kernighan itéré, qui est, pour le moment, l'heuristique la plus performante sur le PVC [26]. Cette heuristique est intéressante car elle n'altère pas l'algorithme de Lin-Kernighan qui conserve ainsi toute son efficacité (contrairement à d'autres méta-heuristiques qui introduisent un élément stochastique).

Mémoire adaptative

Les bonnes performances que donnent les instances de méta-heuristiques ont mené à leur prolifération : tabou génétique [3], système de fourmis 3-opt [11], génétique Lin-Kernighan [27], etc. ne sont que quelques exemples des différentes approches qui ont été réalisées. Or, chaque nouvelle instance souligne les points communs qui existent entre chacune. D'aucuns ont récemment dénommé cette trame de fond la *programmation à mémoire adaptative* [40]. Les méta-heuristiques qui s'y retrouvent ont la particularité de :

1. mémoriser les solutions trouvées lors de la recherche ;
2. construire une solution temporaire à partir de celles en mémoire ;
3. améliorer la solution temporaire grâce à une recherche locale ;
4. mettre à jour la mémoire grâce à la nouvelle solution.

Sont classés dans cette catégorie les algorithmes génétiques, le système de fourmis, la recherche avec tabous et toute autre méta-heuristique faisant appel à une certaine

TAB. 2.1 – Différentes instances de programmation à mémoire adaptative

	Algorithme génétique	**Système de fourmis**	**Recherche avec tabous**
MÉMORISATION	Population de solutions	Collection d'arêtes pondérées	Listes taboues
CONSTRUCTION	Croisement et mutation	Plus proche voisin	Permutation d'arêtes
AMÉLIORATION	Optimisation locale	Optimisation locale	Recherche avec tabous
MISE À JOUR	Sauvegarde et élimination	Phéromone	Listes taboues

forme de mémoire. L'heuristique peut alors être vue comme une forme de machine à état basant son état suivant sur l'état courant.

Nous admettons sans peine, par exemple, que les algorithmes génétiques *mémorisent* la situation courante en retenant les individus du moment ; *construisent* des solutions temporaires en provoquant une mutation et en croisant, éventuellement, les solutions ; *améliorent* ces solutions en effectuant, par exemple, du Lin-Kernighan et *mettent à jour* la population en évinçant un certain nombre d'individus trop peu performants. Le tableau 2.1 présente l'interprétation de quelques méta-heuristiques sous l'angle de la programmation à mémoire adaptative.

Pour ce qui est de l'algorithme qui nous intéresse, le système de fourmis, le caractère essentiel de la mémoire a été reconnu relativement tôt par ceux qui l'ont proposé [11, 15, 10] lorsqu'ils l'ont comparé aux algorithmes d'apprentissage par renforcement [29]. Ceux-ci construisent une représentation du problème auquel ils sont confrontés en associant à chaque situation le comportement optimal, selon leurs connaissances

de l'instant et l'objectif à atteindre. L'exploration de l'environnement (*i.e.*, du problème) permet d'affiner cette représentation afin d'atteindre la solution plus sûrement et plus rapidement lors de la phase d'exploitation. L'algorithme de *Q-learning* [41], en particulier, est très proche du système de fourmis [15]. Il s'agit dans cet algorithme d'attribuer à chaque situation une valeur, la *Q-value*, indiquant la meilleure situation suivante. Si l'on considère qu'une situation est un nœud, que les options envisageables sont les arêtes émergentes et que les indications sur la meilleure situation suivante sont la combinaison linéaire de la distance et de la phéromone (voir section 2.3.1), on retrouve immédiatement l'heuristique du système de fourmis.

2.2 L'algorithme GENIUS

L'heuristique que nous proposons dans ce document repose sur l'algorithme GENI (GENeralized Insertion) [17]. Celui-ci est un algorithme de construction, insérant les nœuds dans le futur cycle selon une procédure que nous présentons dans ce qui suit. GENI est généralement suivi d'une procédure de post-optimisation appelée US (Unstringing, Stringing) et l'ensemble donne un algorithme hybride appelé GENIUS.

Nous commencerons par énoncer les deux algorithmes, GENI et US, et relèverons ensuite leurs caractéristiques. Nous verrons enfin comment ils se comportent en pratique. Rappelons que nous ne nous intéressons qu'au PVC symétrique et euclidien.

2.2.1 Présentation

La procédure GENI est constructive : elle part avec trois sommets pris au hasard et insère progressivement les autres sommets de manière à former un cycle Hamiltonien. La clé de cette heuristique réside dans la façon dont sont insérés les nœuds du graphe. Nous ferons ici une description, reprise de l'article de Gendreau, Hertz et Laporte [17], rappelant les fondements de l'algorithme GENI. Nous énoncerons aussi la procédure de post-optimisation US, dérivée et complément de GENI.

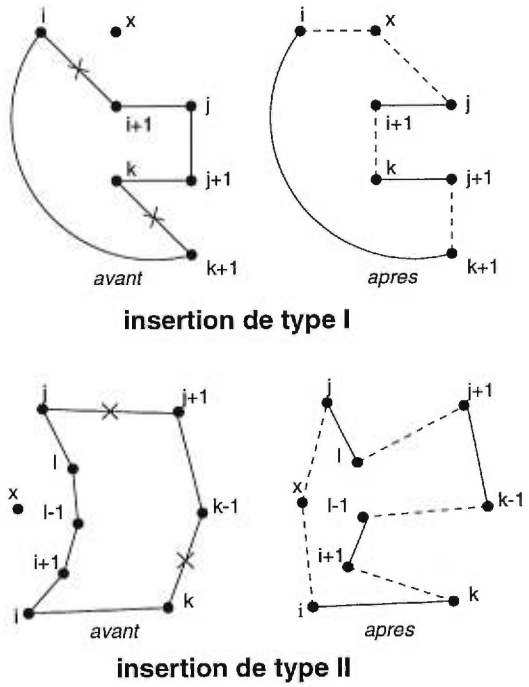

FIG. 2.2 – Les deux types d'insertion GENI lorsque le tour est orienté dans le sens horaire. Nous avons indiqué par une croix les arêtes qui disparaissent et tracé en pointillés les nouvelles.

GENI

Dans GENI, un nœud peut être inséré de *quatre façons différentes* entre deux nœuds qui lui sont *voisins* et l'insertion menant à la meilleur solution temporaire est conservée. Précisons la notion de voisinage et relevons les insertions possibles.

Le voisinage est un élément important de l'algorithme puisqu'il permet de limiter le nombre d'insertions considérées (et donc la complexité de l'algorithme). Les auteurs font appel à un paramètre p qui définit pour un nœud x un voisinage $V_p(x)$ des p nœuds les plus proches, *déjà sur le tour*. Un nouveau nœud ne peut être inséré qu'entre deux nœuds de son voisinage. Un plus grand voisinage permet d'élargir le champ des recherches mais augmente inévitablement le temps de calcul. Ce paramètre oscille généralement entre 3 et 20 selon la qualité de solution voulue (et le temps disponible).

Les insertions sont la force de l'algorithme GENI. Admettons que nous voulions insérer le nœud x entre les nœuds i et $j \in V_p(x)$, que $k \in V_p(i+1)$ se trouve sur

le chemin de j à i pour une orientation donnée du chemin et que $l \in V_p(j+1)$ se trouve, pour la même orientation, sur le chemin de i à j. Les deux types d'insertion, représentées à la la figure 2.2, peuvent avoir lieu pour les deux orientations possibles du graphe (horaire ou trigonométrique) :

- Insertion de x avec mouvement 3-opt (type I)
  $\forall k \in N - \{i, j\}$, l'insertion de x dans le cycle implique le retrait des arêtes $(i, i+1)$, $(j, j+1)$ et $(k, k+1)$. Ces arêtes sont remplacées par (i, x), (x, j), $(i+1, k)$ et $(j+1, k+1)$. Ce réarrangement du cycle est équivalent à une insertion standard suivie d'un mouvement 3-opt.
- Insertion de x avec mouvement 4-opt (type II)
  $\forall k \in N - \{j, j+1\}$ et $\forall l \in N - \{i, i+1\}$, l'insertion de x dans le cycle implique le retrait de $(i, i+1)$, $(l-1, l)$, $(j, j+1)$ et $(k-1, k)$. Ces arêtes sont remplacées par (i, x), (x, j), $(l, j+1)$, $(k-1, l-1)$ et $(i+1, k)$. Ce réarrangement du cycle est équivalent à une insertion standard suivie d'un mouvement 4-opt.

L'algorithme GENI peut être formulé comme suit :

1. Créer un cycle initial en choisissant un ensemble *arbitraire* de trois nœuds. Initialiser le p-voisinage de chacun des nœuds.
2. Choisir *arbitrairement* un nœud $x \in N$ qui ne se trouve pas encore sur le cycle. Insérer x au moindre coût dans le cycle en considérant les deux types et les deux orientations possibles. Mettre à jour le p-voisinage de chacun des nœuds en tenant compte que x est maintenant sur le cycle.
3. Si tous les nœuds font maintenant partie du cycle, arrêter. Sinon, retourner à l'étape 2.

US

L'algorithme GENI peut être suivi par un algorithme d'optimisation qui a été conçu pour lui, US. Celui-ci consiste à considérer pour chaque nœud son retrait selon

une procédure particulière puis sa réinsertion selon GENI ; si ces deux opérations successives mènent à un cycle de moindre coût, elles sont effectuées et le cycle résultant est conservé. Le retrait (*unstringing*) peut être effectuée de deux façons différentes, symétriques aux insertions GENI (*stringing*) :

- Retrait de type I

  Soient $j \in V_p(i+1)$ et, pour une orientation particulière du cycle, $k \in V_p(i-1)$ un nœud sur le chemin de $i+1$ à $j-1$. Le retrait de i implique le retrait de $(i-1,i)$, $(i,i+1)$, $(k,k+1)$ et $(j,j+1)$. Ces arêtes sont remplacées par $(i-1,k)$, $(i+1,j)$, $(k+1,j+1)$.
- Retrait de type II

  Soit $j \in V_p(i+1)$. Pour une orientation particulière du tour, soient $k \in V_p(i-1)$ un nœud sur le chemin de $j+1$ à $i-2$ et $l \in V_p(k+1)$ sur le chemin de j à $k-1$. Le retrait de i implique le retrait de $(i-1,i)$, $(i,i+1)$, $(j-1,j)$, $(l,l+1)$ et $(k,k+1)$. Ces arêtes sont remplacées par $(i-1,k)$, $(l+1,j-1)$, $(i+1,j)$ et $(l,k+1)$.

L'algorithme US peut être formulé comme suit :

1. Soit un cycle κ de coût z. Poser $\kappa^* \leftarrow \kappa$, $z^* \leftarrow z$ et $t \leftarrow 1$.
2. Appliquer les procédures de retrait puis d'insertion du nœud v_t [1] au cycle κ, en considérant à chaque fois les deux types possibles d'insertion et les deux orientations possibles de κ. Soient κ' le meilleur cycle obtenu et z' son coût. Poser $\kappa \leftarrow \kappa'$ et $z \leftarrow z'$.
   - Si $z < z^*$, poser $\kappa^* \leftarrow \kappa$, $z^* \leftarrow z$ et $t \leftarrow 1$; répéter l'étape 2.
   - Si $z \geq z^*$, poser $t \leftarrow t+1$.
   - Si $t = n+1$, arrêter : le meilleur cycle est κ^* et son coût est z^*. Sinon, répéter l'étape 2.

1. v_t indique le t^{ieme} nœud du cycle, comme si les nœuds étaient numérotés de 1 à n. On considère donc tous les nœuds en séquence.

Comme on peut le constater, l'algorithme consiste en fait en une descente locale dans l'espace des modifications US. Le coeur de GENIUS reste évidemment la procédure GENI et c'est sur cette dernière que nous nous attarderons dans ce qui suit.

2.2.2 Un algorithme bivalent

L'algorithme GENI a la particularité de pouvoir insérer un nœud entre deux sommets non-consécutifs et réoptimiser ensuite le chemin entre ces deux sommets. C'est cette double opération qui permet à GENI d'être le meilleur algorithme constructif fait à ce jour, en terme de qualité de solution. Nous reviendrons sur les performances effectives de l'algorithme à la section 2.2.3. Attardons-nous pour le moment sur les mécanismes qui permettent ce dédoublement.

Les quatre types d'insertion (deux pour chaque orientation) constituent évidemment la base de la performance de GENI. Il est possible de les assimiler à des insertions standards (*i.e.*, entre deux nœuds consécutifs), chacune suivie d'un échange d'arêtes 3-opt ou 4-opt, selon le type d'insertion. La figure 2.3, représente une insertion standard suivie d'un échange 3-opt, mouvements équivalents à une insertion GENI de type I. La figure identifie ainsi les deux tâches, insertion et optimisation, dont s'acquitte GENI.

Cette double opération est évidemment coûteuse en temps, d'où l'idée de réduire le champ des possibles en introduisant la notion de p-voisinage. La complexité tombe ainsi à $O(np^4 + n^2)$ puisque l'étape 2, de complexité $p^4 + n$ est répétée $n - 3$ fois. Si l'on garde un p peu élevé, la complexité pratique de GENI est intéressante.

L'hybridation pratiquée par Gendreau, Hertz et Laporte est très fine dans la mesure où les procédures impliquées, l'insertion standard et les échanges 3-opt et 4-opt, sont parmi les plus simples. Elles sont, si l'on peut dire, « petites » et l'enchâssement ne peut se faire que d'une seule façon. GENI est une forme d'hybride minimal, irréductible, quasi-atomique.

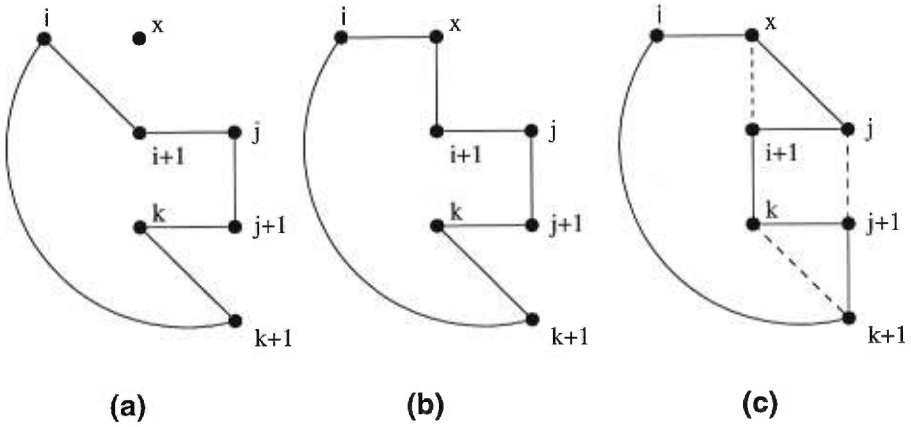

FIG. 2.3 – Insertion standard suivie d'un échange 3-opt équivalente à une insertion GENI de type I. (a) cycle original (b) insertion standard (c) échange 3-opt

Théoriquement, GENI a de quoi se mesurer aux heuristiques simples, tout au moins à celles permettant la construction de cycles. Mais voyons quels sont les résultats effectifs de GENI lorsqu'appliqué à de véritables problèmes.

2.2.3 Un engrenage puissant

L'article [17] présentant GENI tente de comparer l'algorithme avec ceux déjà existants. Il en ressort que GENI trouve des solutions presqu'optimales en un temps réduit par rapport aux autres algorithmes. En particulier, GENI trouve de meilleurs solutions que CCA, lui-même meilleur que toute autre méthode constructive. Il est même compétitif avec certaines heuristiques qui optimisent un chemin déjà construit. Il débouche, par exemple, sur des solutions meilleures que 2-opt dans des problèmes de taille moyenne : GRID100 et G442, présentant respectivement 100 et 442 nœuds. Il va même jusqu'à dépasser Lin-Kernighan pour ce dernier problème. Nous devons admettre que ces deux dernières comparaisons sont délicates, sachant que la puissance des ordinateurs en jeu n'est pas la même pour toutes les expériences. De plus, l'implantation des différents algorithmes n'est pas nécessairement totalement optimisée (structures de données, topologie du problème, etc.).

L'article de Johnson et McGeoch [26] est une étude exhaustive des différents al-

gorithmes de recherche locale. Les comparaisons que l'on y retrouve ont l'avantage inestimable d'être effectuées dans les mêmes conditions. C'est ainsi que l'on y confirme que GENI génère des solutions de très bonne qualité. D'après les résultats obtenus par les auteurs, il dépasse effectivement toute autre heuristique de construction : les solutions trouvées par GENI ne dépassent la borne inférieure de Held-Karp que de 9,1 %, là où l'algorithme de Christofides est à 9,8 %. Quant aux heuristiques d'optimisation locale, dont il possède certains traits, il est meilleur que la plus simple, 2-opt, pour $p \geq 5$ sur des problèmes de « petite » taille[1]. En général, leur performance est proche puisque l'algorithme GENI, avec $p = 20$ se situe en moyenne à 5,6 % au-dessus de la limite de Held-Karp, là où 2-opt est à 4,9 %. Les cycles trouvés par GENI ne sont néanmoins pas de la qualité de ceux trouvés par l'heuristique 3-opt. Sa vitesse moins élevée que celle de ces deux heuristiques est pour les auteurs un facteur limitant.

Mais GENI montre sa valeur lorsque guidé par une méta-heuristique. Par exemple, GENI est utilisé dans l'algorithme TABUROUTE, développé par Gendreau, Hertz et Laporte [18], destiné à donner une solution approximative aux problèmes de confection de routes (PCR). TABUROUTE construit sa solution en échangeant certains nœuds de deux tournées : un nœud est retiré d'une tournée puis inséré dans une autre selon la procédure GENI. Les résultats obtenus sont excellents puisqu'ils ne sont apparemment surpassés que par l'algorithme de Taillard [39] en terme de qualité de solution. Ce dernier, semblable à TABUROUTE en certains aspects, n'utilise cependant pas les insertions GENI et se satisfait d'insertions standards suivies à intervalle régulier d'optimisations locales. Il semblerait que la performance de l'algorithme de Taillard soit essentiellement due à une plus judicieuse décomposition du problème. De plus, il est difficile de comparer les deux algorithmes quant au temps de calcul requis puisque, dans l'article [39], l'algorithme de Taillard s'exécute de façon parallèle.

La jeunesse de GENI réduit le nombre d'utilisation que nous pouvons répertorier. Force est de reconnaître, pourtant, que les résultats qu'il obtient dénote un potentiel élevé. S'il est utilisé dans le cadre d'une méta-heuristique pour le PCR, il semble raisonnable d'en faire de même pour le PVC. Il serait particulièrement intéressant

1. Les auteurs donnent l'exemple d'un problème de 100 nœuds.

pour un algorithme reposant fortement sur la construction des solutions ; or le système de fourmis est précisément ce type d'algorithme. Nous le présentons dans la section qui suit.

2.3 L'algorithme Ant Colony System

Une fourmilière présente des caractéristiques intéressantes, que cela soit pour l'entomologiste, le chercheur en intelligence artificielle ou encore le chercheur en optimisation combinatoire. Pour l'un, elle présente des caractéristiques uniques dans le monde des insectes : esclavagisme, présence de mercenaires, communication, robuste adaptation aux nouvelles conditions, etc. [24]. Pour l'autre, la fourmilière peut être vue comme un tout, comme une entité vivante individuelle ayant des besoins et des objectifs particuliers, habitée de plusieurs « agents » plus simples d'importance relativement égale dont émerge une certaine intelligence [35]. Pour le dernier, les fourmis sont capables de déterminer simplement un chemin quasi-optimal d'un point à un autre.

Le système de fourmis (Ant System, AS) est une contribution récente au domaine des méta-heuristiques [9]. Comme son nom l'indique, il s'agit de résoudre des problèmes d'optimisation combinatoire à l'aide de « fourmis » artificielles dont le comportement est vaguement inspiré de celui de fourmis réelles. Dans les paragraphes qui suivent, nous présentons l'algorithme Ant Colony System (ACS), dernière mouture en date du système de fourmis, explicitons ses spécificités et relevons certains résultats pratiques.

2.3.1 Présentation

L'ACS s'inspire de la manière dont les fourmis explorent leur environnement à la recherche de nourriture et trouvent collectivement un court chemin d'une source de nourriture à la fourmillière. Lorsqu'elles recherchent de la nourriture, les fourmis

réelles partent de la fourmilière de manière à peu près aléatoire. Elles s'éloignent progressivement de la fourmillière jusqu'à ce qu'elles trouvent une source de nourriture. Elles prennent alors un peu de cette nourriture et reviennent à la fourmillière. Chemin faisant, elles déposent des traces odorantes, la phéromone, indiquant la qualité de la source (déterminée par la quantité de nourriture) et la distance de la source à la fourmillière. Plus la phéromone indiquera une source intéressante (qualité et distance), plus les autres fourmis exploratrices y seront attirées.

Ainsi, les nouvelles fourmis partant à la recherche de nourriture effectueront leur recherche de façon moins aléatoire : elles suivront les traces de phéromone déjà déposées par les premières exploratrices. En particulier, elles suivront plus probablement les traces de phéromone menant à une source de nourriture intéressante. De fil en aiguille, les fourmis convergeront vers le chemin le plus intéressant. Lorsque cette source-là sera épuisée, d'autres exploratrices pourront drainer les fourmis vers une nouvelle source intéressante par le même processus de communication indirecte.

L'algorithme ACS peut s'énoncer comme suit :

1. Initialiser : la quantité initiale de phéromone, τ_0, est la même pour toutes les arêtes.
2. Tant qu'une condition de fin n'est pas satisfaite
   (a) Toutes les fourmis sont positionnées sur un nœud initial
   (b) Pour toutes les fourmis
      i. Utiliser une fonction de transition d'état probabiliste pour construire incrémentalement une solution
      ii. Mettre à jour la phéromone localement (voir l'équation 2.3)
   (c) Mettre à jour la phéromone globalement (voir l'équation 2.4)

Les fourmis construisent chacune leur propre solution, itérativement. Pour ce faire, elles suivent une fonction de transition d'état (*state transition rule*, expression utilisée

par Dorigo *et al.*). Celle-ci indique le nœud j vers lequel se diriger lorsque la fourmi se trouve sur le nœud i :

$$j = \begin{cases} \text{argmin}_{k \in R}\left(\frac{c(i,k)^\beta}{\tau(i,k)}\right), & \text{si } q < q_0 \\ J, & \text{sinon} \end{cases} \tag{2.1}$$

avec

$c(i,j) \geq 1$	le coût de l'arête de i à j
$\tau(i,j)$	la phéromone déposée sur l'arête allant de i à j,
$\beta > 0$	le paramètre déterminant l'importance relative du coût par rapport à la phéromone,
$q \in [0;1]$	un nombre aléatoire distribué uniformément,
$q_0 \in [0;1]$	un paramètre,
R	l'ensemble des nœuds non encore insérés,
J	une variable aléatoire tirée de la distribution suivante :

$$p(i,j) = \begin{cases} \frac{\frac{\tau(i,j)}{[c(i,j)]^\beta}}{\sum_{k \in R} \frac{\tau(i,k)}{[c(i,k)]^\beta}} & \text{si } j \in R \\ 0, & \text{sinon} \end{cases} \tag{2.2}$$

Cette fonction favorise les arêtes les moins coûteuses et comportant le plus de phéromone. Plus β est élevé, plus on favorise les coûts réels des arêtes. Si, au contraire, on pondère moins fortement le coût, on favorise l'exploitation des chemins déjà tracés antérieurement.

La mise à jour locale de phéromone consiste à modifier la quantité de phéromone sur les arêtes visitées par une fourmi. La modification peut consister en l'ajout de phéromone, pour favoriser l'exploration locale, ou en son retrait, pour mener à une plus grande diversification des solutions. Dorigo *et al.* ont privilégié la mise à jour additive dans les premières moutures de l'algorithme. Plus récemment, le retrait de phéromone a été envisagé et c'est ce que nous avons choisi ici. La mise à jour locale s'exprime alors de la façon suivante :

$$\tau(i,j) \leftarrow (1-\rho)\cdot\tau(i,j) + \rho\cdot\tau_0, \text{ pour toute arête } (i,j) \text{ visitée par la fourmi} \quad (2.3)$$

avec τ_0 la quantité initiale de phéromone sur les arêtes et $0 < \rho < 1$ un paramètre permettant de contrôler la diversification des recherches. Plus ρ est élevé, moins les fourmis se dirigeront vers les endroits de l'espace de solution où sont déjà passées d'autres fourmis. Une diversification trop élevée mène évidemment à une dispersion des fourmis dans l'espace des solutions.

La mise à jour globale consiste, elle, à favoriser le meilleur chemin en y déposant un peu plus de phéromone que sur les autres, comme si plusieurs fourmis *élitistes*[1] y étaient passées. Cette mise à jour consiste aussi à retirer un peu de phéromone de tous les chemins (*évaporation*). Ces deux opérations s'expriment de la façon suivante :

$$\tau(i,j) \leftarrow (1-\alpha)\cdot\tau(i,j) + \alpha\cdot\Delta\tau(i,j) \quad (2.4)$$

avec

$$\Delta\tau(i,j) = \begin{cases} (C^*)^{-1}, & \text{si } (i,j) \in \text{meilleur cycle trouvé} \\ 0 & \text{sinon} \end{cases}$$

$0 < \alpha < 1$ un paramètre et C^* le coût du meilleur cycle trouvé jusqu'à cette mise à jour.

Cette double opération a deux buts : entraîner les fourmis vers une région intéressante de l'espace des solutions et réduire peu à peu le champ d'investigation.

2.3.2 Une décomposition des solutions

L'ACS s'articule autour 1° d'une représentation particulière de l'espace des solutions et 2° de la construction de solutions à partir de cette représentation. D'un côté, les solutions sont décomposées selon les arêtes qui les forment ; de l'autre, de nouvelles solutions sont reconstruites arête par arête, en tenant compte de l'importance

1. Élitistes dans la mesure où elles choisissent invariablement le meilleur chemin.

relative de chacune. Cette importance est directement proportionnelle à la qualité des solutions auxquelles chaque arête a appartenu.

La décomposition des solutions par l'attribution aux arêtes d'une pondération fonction de leur fréquence d'apparition dans les meilleures solutions est relativement neuve. Elle est inspirée de l'hypothèse répandue voulant que les optima locaux soient proches les uns des autres[1] [26]. Cette hypothèse a le double avantage de s'avérer dans la pratique [12] et permet de restreindre le nombre de solutions à considérer.

Afin que les poids des arêtes guident correctement la recherche, il est bien sûr nécessaire de les mettre à jour à mesure de l'avancement de la recherche. Cette mise à jour constitue un élément-clé de la performance de l'algorithme. La méthode utilisée ici, nous l'avons vue, consiste à attribuer un poids élevé aux arêtes des meilleures solutions. Cette opération de mise à jour doit cependant être mitigée pour ne pas s'enferrer trop tôt dans un minimum local, d'où le phénomène d'évaporation. À l'extrême, les poids peuvent même être altérés pour *élargir* le champ de recherche. TABUROUTE, par exemple, comprend un terme favorisant le mouvement des arêtes qui n'ont été que peu déplacées lors de la recherche. Cette considération se retrouve dans la mise à jour locale de l'ACS : les arêtes empruntées subissent une *diminution* de leur pondération.

Étant donné que la représentation que garde l'ACS de l'espace des solutions, une liste d'arêtes pondérées, est agglomérée, il est nécessaire d'en extraire une solution. Ceci est accompli grâce à un algorithme de construction sensible aux pondérations attribuées lors de la décomposition. L'algorithme retenu et présenté dans le papier de Dorigo *et al.* est une variante stochastique de l'heuristique du plus proche voisin (voir section 2.1.3). L'aléatoire y est introduit lors du choix de la prochaine ville à visiter, dans le but diversifier les solutions générées. En effet, si l'heuristique du plus proche voisin avait été utilisée sans modification, il est clair qu'il n'y aurait eu que n solutions générées (une par nœud de départ).

1. La proximité de deux solutions est déterminée, ici comme dans les autres heuristiques, par le nombre d'arêtes les différenciant : un échange 2-opt est un pas plus court que deux échanges 2-opt différents.

2.3.3 Une performance prometteuse

Le système de fourmis présente une façon novatrice de combiner diverses solutions afin de guider les algorithmes de construction vers les régions intéressantes de l'espace des solutions. En pratique, cette technique se comporte bien, trouvant de façon répétée l'optimal de problèmes simples ou s'en approchant pour les problèmes de dimension plus élevée. Les évolutions récentes du système de fourmis présentent des caractéristiques permettant d'en améliorer la performance, qualité et temps de calcul confondus.

Les premières versions de l'algorithme n'étaient pas, nous en convenons, destinées à entrer en compétition avec Lin-Kernighan. Il s'agissait de proposer une nouvelle méta-heuristique de recherche locale. Cette considération se retrouve dans le premier article de Colorni, Dorigo et Maniezzo [9] où les temps de calcul ne sont pas présentés. Les résultats obtenus sont cependant encourageants puisque le système de fourmis trouve de meilleures solutions que l'optimisation 2-opt et atteint la performance de Lin-Kernighan.

Plus tard, les auteurs ont eu le temps de comparer leur heuristique avec la recherche avec tabous et le recuit simulé [12]. Ils constatent que le système de fourmis ne s'en laisse pas imposer quant à la qualité des solutions trouvées : il égale la recherche avec tabous et surpasse le recuit simulé. Cependant, les temps de calcul ne sont toujours pas indiqués et le problème sur lequel ont été effectués les tests ne compte que 30 nœuds.

Enfin, récemment, les auteurs ont présenté un algorithme issu du premier, l'ACS, capable d'égaler la performance des algorithmes génétiques. Ici, la mise à jour locale est légèrement différente, permettant une meilleure diversification et, surtout, une optimisation locale est effectuée sur les cycles construits par chaque fourmi. Cette fois-ci, les performances sont assez bonnes pour que les auteurs indiquent les temps de calcul. Il en ressort que l'ACS-3-opt, c'est-à-dire un ACS dont les solutions sont localement optimisées par 3-opt, atteint des solutions de qualité légèrement inférieure à un algorithme génétique basé sur une optimisation Lin-Kernighan mais en beaucoup

TAB. 2.2 – Résultats comparés de l'ACS post-optimisé et d'un algorithme génétique. La colonne t indique le temps de résolution en secondes ; la colonne ϵ indique l'écart relatif entre la solution trouvée et la solution optimale. Ce tableau est tiré de [11]

Problème	ACS-3-opt		SSTP-GA	
	t	ϵ	t	ϵ
d198	238	0,01 %	253	0,00 %
lin318	537	0,00 %	2054	0,00 %
att532	810	0,11 %	11780	0,03 %
rat783	1280	0,36 %	21210	0,01 %

moins de temps. Les résultats obtenus sont répertoriés dans le tableau 2.2, où l'on retrouve, pour quatre problèmes, les temps de résolution et l'écart relatif par rapport à la solution optimale. On remarque aussi que les instances de PVC qui ont constitué les tests sont de dimension plus élevée que ce qui avait été présenté auparavant. Notons enfin que les auteurs désirent remplacer l'optimisation locale 3-opt par une optimisation Lin-Kernighan, plus performante.

Nous restreignons notre étude au PVC mais il est intéressant de constater que cette méta-heuristique s'applique également bien à d'autres problèmes réputés plus difficiles. Par exemple, le système de fourmis original a été appliqué récemment au problème d'élaboration de tournées [6]. Dans ce cas, il semble que le système de fourmis, aidé de l'optimisation 2-opt, arrive dans les temps à des solutions similaires à celles trouvées par les autres méta-heuristiques performantes (recherche avec tabous, recuit simulé, etc.). Il n'obtient cependant pas la meilleure performance. Une version de AS a aussi été proposée pour le problème de l'affectation quadratique (Quadratic Assignment Problem, QAP) [16, 38]. Les résultats obtenus sont, là aussi, prometteurs en terme de qualité de solution.

Nous avons présenté, dans ce chapitre, le contexte dans lequel se situe ce projet. Nous avons commencé par présenter le problème sur lequel nous travaillons, le PVC symétrique dans le plan euclidien, afin de restreindre notre champ de recherche, fixer

les contraintes à respecter et situer notre travail dans la mouvance actuelle, en l'occurence, la coopération basée sur une mémoire partagée. Nous avons ensuite présenté les deux heuristiques sur lesquelles seront basées notre travail : GENI et le système de fourmis. Nous avons mis l'emphase sur le fait que GENI est une heuristique hybride quasi-atomique, particulièrement performante lorsqu'utilisée dans le cadre de méta-heuristiques. ACS se présente comme une méta-heuristique récente mais prometteuse, représentant une agglomération de l'espace des solutions que dépêtre un algorithme de construction.

Nous avons tenté de faire ressortir les crochets que nous utiliserons dans le prochain chapitre pour imbriquer les algorithmes ACS et GENI. En une phrase : nous voulons établir une collaboration, par le biais d'un algorithme ACS, non pas entre algorithmes du plus proche voisin (comme dans Dorigo *et al.*) mais entre algorithmes GENI. Ou encore : nous souhaitons mettre en œuvre plusieurs algorithmes GENI doués d'une mémoire commune.

Chapitre 3

Le système de colonie de fourmis GENI

NOUS PROPOSONS ici une extension de l'algorithme GENI, à la lumière de ce que nous avons présenté dans le chapitre précédent. En particulier, nous y avons vu qu'un algorithme de construction simple, l'heuristique du plus proche voisin (PPV), pouvait apprendre à mieux résoudre un problème de voyageur de commerce (PVC), s'il s'appuyait sur une représentation adaptée du problème, sa mémorisation et sa mise à jour au fil des expériences. L'objectif de ce chapitre et, plus généralement, de ce projet est de fournir à GENI le même type de tremplin.

L'adaptation ne s'effectue cependant pas de but en blanc et nous devons d'abord identifier les éléments qui seront affectés. Pour ce faire, nous « déconstruisons » l'algorithme du système de fourmis. Nous reprenons ensuite ses éléments essentiels et les adaptons, si besoin est, à l'utilisation que nous souhaitons en faire. Nous présentons en dernier lieu le résultat de l'hybridation que nous aurons ainsi effectuée.

3.1 Décomposition

Dans cette section, nous décortiquons l'algorithme du système de fourmis. Nous suivons pour ce faire un processus connu : la simplification. Nous isolons autant que faire se peut les composantes de l'algorithme original. Cette démarche nous amène à une meilleure compréhension du tout, compréhension indispensable pour notre tâche.

3.1.1 Collaboration

L'algorithme original du système de fourmis s'appuie sur plusieurs fourmis semblables travaillant toutes sur le même problème et partageant toutes la même mémoire. Simplifions : intéressons-nous à un système mono-fourmi.

Nous obtenons l'algorithme suivant :

1. Initialiser
2. Tant qu'une condition de fin n'est pas satisfaite
   (a) Utiliser la fonction de transition d'état pour construire incrémentalement une solution
   (b) Mettre à jour la phéromone globalement

Non seulement n'y a-t-il plus de référence qu'à une seule fourmi mais la mise à jour locale a disparu. Rappelons que son rôle est de réduire l'intérêt (*i.e.*, la quantité de phéromone) des arêtes qui viennent d'être parcourues, selon l'équation (2.3) de la page 26. Rappelons aussi que Dorigo *et al.* ont bâti plusieurs algorithmes basés sur la métaphore des fourmis et ont proposé plusieurs équations différentes de mise à jour locale. Celle que nous utilisons ici est celle présentée dans l'algorithme ACS [11].

La mise à jour locale sert donc à limiter les recherches redondantes. Les fourmis subséquentes seront moins « tentées » de suivre un chemin déjà exploré et essaieront d'autres combinaisons, diversifiant ainsi la recherche. La mise à jour locale est un

élément qui assure la cohérence de la recherche, lorsque celle-ci est effectuée par plusieurs agents aux mécanismes de recherche semblables.

Nous avons vu quelle était la stratégie qui permet d'obtenir une collaboration efficace entre fourmis. Focalisons-nous maintenant sur une seule fourmi, afin d'en extraire les éléments essentiels.

3.1.2 Fourmi PPV

Reprenons tout d'abord le processus que suit une fourmi pour évoluer dans son environnement, *i.e.* un graphe complet symétrique dans le cas qui nous intéresse. Étant donnée un nœud, la fourmi PPV perçoit dans son environnement les éléments qui lui permettent de passer à un autre nœud. Le système perceptuel de la fourmi PPV est formé de deux « sens » : le premier détecte le coût des arêtes émergentes du sommet courant et le deuxième la phéromone qui y a été déposée. Ces deux éléments d'information sont combinés et transmis à un algorithme qui détermine en conséquence la marche à suivre. Dans le cas de la fourmi PPV, il s'agit de l'algorithme du plus proche voisin stochastique. La décision prise, la fourmi change de nœud. Lorsque la construction d'un tour est terminée, la meilleure solution se voit attribuer une certaine quantité de phéromone et les autres voient leur phéromone diminuer.

Description parlante mais qui pose plus de questions qu'elle n'en résoud. Qu'est ce que cette phéromone, quelle justification à sa combinaison avec le coût, pourquoi le PPV stochastique, pourquoi le renforcement? Approfondissons.

Architecture

L'environnement joue un double rôle : celui de donner à la fourmi son état externe de l'instant *et* celui de lui rappeler les fortunes diverses rencontrées en suivant tel ou tel chemin. La phéromone est ainsi une instanciation de la mémoire de la fourmi. La dispersion de la mémoire à même le graphe est un moyen commode de permettre

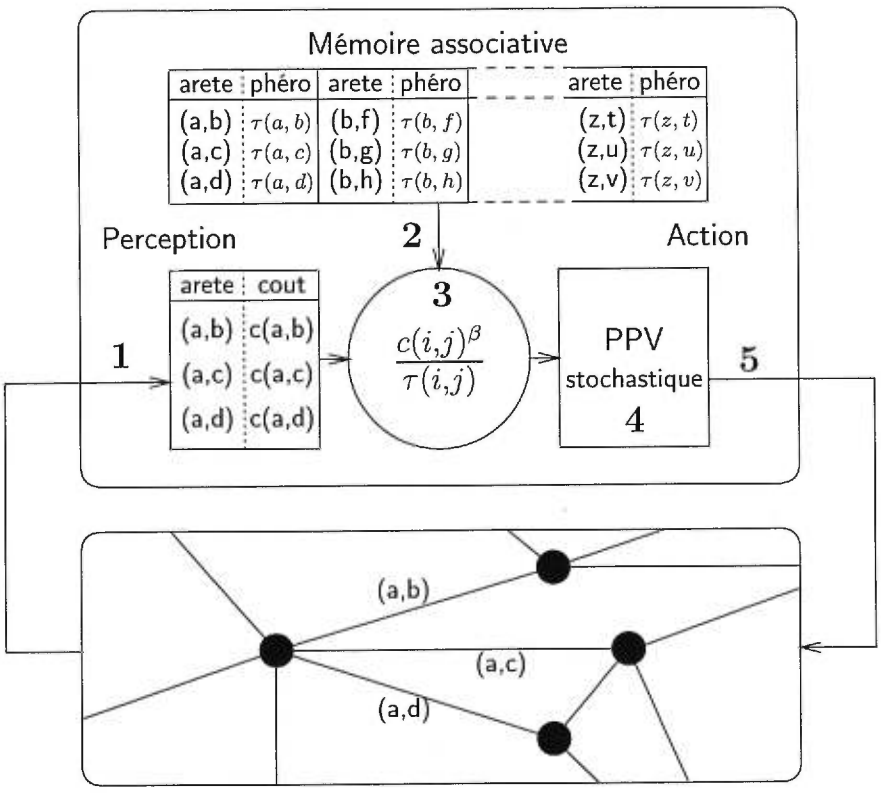

FIG. 3.1 – Composantes de l'heuristique du PPV avec mémoire en interaction avec son environnement (un graphe). La fourmi *perçoit* d'abord son environnement en extrayant le coût des arêtes adjacentes (**1**). Après avoir retrouvé dans sa mémoire la pondération attribuée à ces arêtes (**2**), elle les combine aux coûts (**3**). Le résultat de cette combinaison permet au PPV stochastique de *décider* de la marche à suivre (**4**). La fourmi peut ensuite *agir* sur son environnement en passant à un autre nœud (**5**).

à d'autres fourmis de profiter de son expérience. Nous ne nous intéressons qu'à une seule fourmi, restituons lui donc sa mémoire pour en faire une fourmi individuelle et autosuffisante. Avant d'aller plus loin, regardons la figure 3.1 qui représente schématiquement l'architecture de notre fourmi PPV avec mémoire.

Au delà la mise en évidence de la mémoire, la figure 3.1 montre clairement la parenté de la fourmi PPV et des algorithmes d'apprentissage par renforcement. Par exemple, l'*Adaptative Heuristic Critic* (AHC), schématisé à la figure 3.2, présente de troublants points communs. Essentiellement, l'idée d'AHC, est d'avoir un module-critique (étiqueté *AHC* sur le schéma) qui suggère à un algorithme d'apprentissage (étiqueté *RL*) la meilleure marche à suivre globale. On peut imaginer que, dans la fourmi PPV, le module-critique est constitué de la mémoire et de l'opération combi-

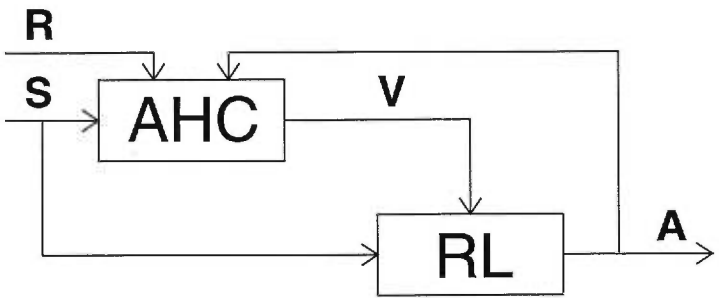

FIG. 3.2 – Architecture de l'algorithme *Adaptative Heuristic Critic*. Le module *AHC* représente le « critique » et *RL* est une instance d'algorithme d'apprentissage par renforcement. Le critique reçoit des signaux de renforcement R (le coût de la solution, par exemple) et un signal S indiquant l'état du moment (le nœud sur lequel la fourmi se trouve, par exemple). Le critique envoie une valeur heuristique V à l'algorithme d'apprentissage *RL* et lui suggère ainsi une marche à suivre (V peut être vu comme le résultat de la combinaison dans notre cas). Enfin, l'algorithme d'apprentissage effectue une action A. Ce schéma est tiré de l'article [29].

nant coût et pondération tandis que le PPV stochastique est l'algorithme d'apprentissage *RL*. Le *Q-learning*, dont émane le système de fourmis [15], est d'ailleurs très directement inspiré de l'AHC. La différence réside essentiellement dans la façon de mettre à jour les poids attribués aux suggestions [29].

Ce rapprochement répond du coup à toutes les autres questions que nous nous posions. La combinaison est une *heuristique*, qui ne peut se justifier que par sa qualité empirique ; l'aléatoire est utile pour l'*exploration* de l'environnement ; le renforcement est une composante essentielle des algorithmes d'apprentissage...par renforcement. Et, plus important, nous avons identifié avec précision les deux éléments essentiels de l'architecture et leur rôle : la *mémoire* pour influer le cours des évènements et le *PPV stochastique* pour explorer et exploiter les informations contenues dans ladite mémoire.

Mémoire

La mémoire contient, nous l'avons vu au chapitre précédent, une représentation particulière de l'espace des solutions. Elle est l'agglomération des différentes expériences « vécues », des différentes solutions trouvées antérieurement. L'intérêt de cette agglomération réside dans la décomposition des solutions en leurs éléments essentiels,

leurs arêtes.

La mémoire influence le cours de l'algorithme en émettant un « avis » sur l'intérêt d'une arête particulière. Cet avis se traduit par un poids numérique, la quantité de phéromone, qui est combiné au coût puis transmis à l'algorithme de recherche. Nous appelons le résultat de la combinaison le *coût ajusté* $c_a(i,j)$ de l'arête, par opposition à son coût brut $c(i,j)$:

$$c_a^{ACS}(i,j) = \frac{[c(i,j)]^\beta}{\tau(i,j)} \tag{3.1}$$

Le coût ajusté est moins élevé pour les arêtes les moins coûteuses comportant le plus de phéromone ; le PPV favorise donc ce type d'arête. La sensibilité à la mémoire est contrôlée par le paramètre β. Plus il est élevé, moins la mémoire a d'influence sur les choix de la fourmi PPV[1].

La façon dont évolue la quantité de phéromone sur une arête est une question cruciale. La méthode retenue consiste à renforcer systématiquement les arêtes composant la meilleure solution trouvée, selon l'équation (2.4) de la page 26.

Le renforcement s'effectue par une petite augmentation de l'intérêt présenté par les arêtes composant la meilleure solution et par la diminution des autres. Il a pour objectif de « tirer » l'algorithme vers une région « intéressante » de l'espace des solutions. C'est ce qui est appelé la « mise à jour globale ».

Exploration

L'algorithme du choix de l'action[2] de la fourmi PPV est un dérivé stochastique de l'algorithme du plus proche voisin. Comme tout algorithme, il dépend 1° de sa

1. Il est intéressant de noter que l'interprétation que nous avons de la combinaison de la quantité de phéromone et du coût diffère selon le domaine d'étude. Si l'on se place dans le domaine de la recherche opérationnelle, c'est la phéromone qui biaise le coût. Si on s'intéresse plutôt aux algorithmes d'apprentissage, c'est au contraire le coût qui constitue un biais (l'inverse du coût est parfois appelé heuristique par Dorigo *et al.*[11]).

2. *action selection algorithm* dans la littérature d'intelligence artificielle.

structure interne et 2° de son initialisation.

PPV stochastique À l'origine, l'heuristique du plus proche voisin choisit en chaque nœud l'arête la moins coûteuse parmi celles qui sont disponibles. Le PPV stochastique spécifie d'ajouter, après le nœud i, le nœud j tiré de la distribution aléatoire définie à l'équation (2.2) (page 25).

Afin d'avoir un certain contrôle sur le degré de hasard mis en jeu, Dorigo *et al.* ont ajouté un paramètre permettant de donner une plus ou moins grande importance à l'arête la moins coûteuse [11]. Le PPV stochastique restreint, que nous appelons simplement le PPV stochastique, spécifie, lui, d'ajouter les nœuds en les choisissant selon la règle (2.1) (page 25).

Par rapport au PPV original, le PPV stochastique donne une plus grande diversité de résultats. Ceux-ci ne seront pas nécessairement meilleurs mais seront plus variés (si $0 \leq q_0 < 1$). Donc, utilisé seul, le PPV stochastique n'offre que peu d'intérêt. Il montre sa valeur, par contre, quand il agit conjointement avec une mémoire. La mémoire n'est utile en effet que lorsqu'elle est la synthèse de diverses expériences. Un PPV classique se dirige directement vers la solution qu'il pense être la meilleure. Il est inévitable qu'il retombe presque toujours sur les mêmes solutions, privant la mémoire de cette diversité sans laquelle elle est inutile ; le PPV stochastique n'a pas ce défaut. La fourmi PPV suit grâce à lui une stratégie d'*exploration aléatoire* de l'espace des solutions[1].

Un constat : q_0 n'est pas augmenté au fil des itérations. Ceci va à l'encontre de ce que l'on retrouve généralement dans les algorithmes d'apprentissage. Les preuves de convergence ne sont en effet valables que lorsque l'exploration s'estompe progressivement au profit de l'exploitation des données. En fait, ceci est compensé par la mise à jour de la mémoire et des pondérations attribuées à chaque arête. On peut voir en effet à l'équation (2.4) (page 26) que le coût des arêtes peu intéressantes est systématiquement diminué au profit de la meilleure solution.

1. *randomized exploration strategy* [29]

Initialisation Initialiser une fourmi PPV consiste à la placer, avant même que sa recherche ne commence, sur un nœud particulier. Ce faisant, on oblige la recherche à s'effectuer dans une région particulière de l'espace des solutions. Une bonne initialisation est le complément indispensable de toute stratégie de diversification.

Si l'algorithme est déterministe, il reproduira d'une itération à l'autre le même comportement, pour une condition initiale donnée. Ajouter un élément stochastique amplifie certes la diversification ; mais, sans variation dans l'initialisation, elle ne pourra être que limitée.

Dans le cas du système de fourmis, différents types d'initialisation ont été expérimentés. Il n'est pas étonnant que la meilleure soit celle qui consiste à placer chacune des fourmis sur une ville différente. Ainsi, chacune des fourmis explore une région différente de l'espace des solutions.

3.2 Composition

Après avoir décomposé le système de fourmis PPV, nous avons une meilleure connaissance des éléments qui seront nécessaires à la construction d'un système de fourmis GENI. Puisque nous avons utilisé une approche « top-down » pour la décomposition, il nous a semblé naturel d'utiliser une approche « bottom-up » pour la...composition.

3.2.1 Fourmi GENI

La fourmi GENI a, *a priori*, la même architecture que la fourmi PPV que nous venons de voir : une mémoire et sa mise à jour, un algorithme pour le choix de l'action à effectuer.

Mémoire

Dans le cas de la mémoire, il est bien évident que nous ne changerons en rien la représentation interne des solutions : c'est une des caractéristiques de base du système de fourmis ! De même, rien, *a priori*, ne nous empêche d'utiliser une combinaison identique pour calculer le coût ajusté.

Il nous est toutefois apparu, lors d'expériences préliminaires, que nous obtenions de meilleurs résultats lorsque nous utilisions une autre définition du coût ajusté. Celle-ci 1° donne une grande importance au coût original et 2° conserve le même ordre de grandeur que le coût original.

$$c_a^{GENI}(i,j) = c_a(i,j) = \frac{c(i,j)}{1 + \beta \cdot \tau_r(i,j)} \tag{3.2}$$

où $\tau_r(i,j)$ est la quantité relative de phéromone sur l'arête (i,j), c'est-à-dire

$$\tau_r(i,j) = \frac{\tau(i,j)}{\max_{k,l \in N}(\tau(k,l))} \tag{3.3}$$

Cette équation possède deux propriétés supplémentaires. D'abord, elle normalise la quantité de phéromone sur les arêtes, donnant ainsi des résultats plus robustes, *i.e.* plus semblables d'un problème à l'autre. Et, ensuite, elle n'élimine pas d'office la possibilité qu'une fourmi sélectionne une arête très peu visitée, contrairement à l'équation (3.1) (qui donne à ces arêtes un coût extrêmenent élevé).

Une dernière remarque sur cette nouvelle heuristique : la quantité initiale de phéromone sur les arêtes, τ_0, peut ici être égale à 0. Dans ce cas, le coût ajusté d'une arête est égal à son coût réel.

Quant à la mise à jour globale des traces mémorielles, la raison qui est à son origine, attirer vers une région intéressante de l'espace des solutions, reste valable pour GENI. Nous la conservons donc sans modification.

Exploration

L'algorithme GENI donne toujours, pour une condition initiale donnée, la même solution, la meilleure à laquelle il peut parvenir et, pour cela, il est dit *déterministe.* Cette caractéristique est une qualité indispensable lorsque le temps est une ressource comptée. Cependant, puisque GENI devient un algorithme d'exploration, nous devons lui permettre une certaine marge d'erreur.

GENI stochastique Le PPV stochastique altère aléatoirement le choix du prochain nœud à visiter : plutôt que de choisir *systématiquement* le plus proche parmi ceux non encore visités, on choisit *probablement* le plus proche, mais pas nécessairement. Nous voudrions altérer GENI de la même façon.

À chaque itération, GENI choisit le type d'insertion menant au cycle de moindre coût. Nous pouvons perturber ce choix en affectant à chaque insertion possible une probabilité fonction de la « qualité » de l'insertion : plus court est le tour résultant d'une insertion, meilleure est cette insertion. Imaginons que I insertions GENI soient possibles[1] et que nous les pondérions en fonction de leur qualité : l'insertion de meilleure qualité, i_1, a un poids $\pi(i_1) = I$, la deuxième un poids $\pi(i_2) = I - 1$, etc. et la moins bonne un poids $\pi(i_I) = 1$. Ainsi, la probabilité de sélectionner une insertion GENI i est :

$$p(i) = \frac{\pi(i)}{\sum_{j \in [1;I]} \pi(j)} = \frac{\pi(i)}{\frac{I \cdot (I+1)}{2}} \tag{3.4}$$

Les insertions menant à un tour plus court et dont les arêtes ont été plus utilisées sont ainsi plus probablement sélectionnées.

Comme pour le PPV stochastique, nous introduisons un paramètre permettant de contrôler le degré de hasard. Ainsi, le GENI stochastique choisit l'insertion θ à effectuer grâce à la règle suivante :

1. $I < 4 \cdot p^4$: pour les 4 types d'insertion, on examine $p \cdot p \cdot p \cdot p$ combinaisons de nœuds pour les nœuds i, j, k et l, avec p le paramètre définissant le voisinage de la procédure GENI (section 2.2)

$$\theta = \begin{cases} i_1, & \text{si } q < q_0 \\ \iota, & \text{sinon} \end{cases} \tag{3.5}$$

où q est un nombre aléatoire distribué uniformément entre 0 et 1, $q_0 \in [0;1]$ est le paramètre de contrôle et ι est une variable aléatoire tirée de la distribution définie en (3.4).

Initialisation Nous avons vu comment nous pouvons perturber aléatoirement une exécution de GENI. Nous favorisons grâce à cela l'exploration des différentes arêtes du graphe. Cependant, si nous ne changeons pas de condition initiale, nous n'aurons qu'un échantillon réduit de solutions.

Où, dans l'algorithme, fait-on le choix de la région explorée? Si nous nous référons à l'algorithme de GENI (voir la section 2.2.1, page 17), nous constatons que ce choix s'effectue en deux temps. D'abord, à l'étape 1, nous devons choisir « un ensemble *arbitraire* de trois nœuds » pour construire un cycle initial. Ensuite, à l'étape 2, il s'agit de choisir « *arbitrairement* un nœud qui ne se trouve pas encore sur le cycle » pour l'y insérer. En somme, l'ordre d'insertion des nœuds dans le cycle est laissé à la discrétion de celui qui implante l'algorithme. Cet ordre constitue la condition initiale de GENI.

Le nœud à insérer peut être choisi de diverses manières: selon une distribution aléatoire uniforme parmi les nœuds qui ne sont pas encore sur le cycle, selon la quantité de phéromone sur les arêtes émergentes, etc. Pour notre part, nous nous limiterons à tester l'influence d'un ordre d'insertion aléatoire.

Voici donc définis les éléments constituants de la fourmi GENI: une mémoire, calquée sur celle utilisée par la fourmi PPV, et un algorithme exploratoire, le GENI stochastique. Nous sommes en mesure de proposer un algorithme pour un système de fourmis GENI ne comportant qu'une seule fourmi. Reste à définir l'aspect collaboratif.

3.2.2 Collaboration

Nous n'apportons ici aucun changement, comme pour tout ce qui touche la mémoire. Les raisons qui poussent à diversifier la recherche sont encore valables : dans le système de fourmis GENI, nous voulons que les fourmis empruntent avec moins d'entrain les arêtes déjà visitées par des fourmis de la même génération. Nous conservons donc la mise à jour locale telle que nous l'avons présentée à la section 3.1.1. Remarquons tout de même que, puisque nous choisirons $\tau_0 = 0$, l'équation se réduit à :

$$\tau(i,j) \leftarrow (1-\rho) \cdot \tau(i,j), \text{ pour toute arête } (i,j) \text{ visitée par la fourmi} \qquad (3.6)$$

Nous avons maintenant tous les éléments qui nous permettent de construire un système de fourmis GENI.

3.3 Le système de colonie de fourmis GENI

3.3.1 Algorithme GACS

La structure générale du système de colonie de fourmis GENI (GACS) est calquée sur l'algorithme du système de fourmis original de Dorigo et Gambardella [11]. Il est présenté à la table 3.1.

3.3.2 Complexité

Si nous comparons cet algorithme à celui présenté à la section 2.3.1 de la page 23, on constate que seules les étapes de la construction d'une solution sont modifiées. La complexité de l'algorithme s'en ressent évidemment un peu. L'introduction de GENI dans l'algorithme (boucle [2(a)iii]) fait passer la complexité de $O(NC \cdot m \cdot n^2)$

1. Initialiser : lire le problème, initialiser les structures de données, etc.
2. Tant qu'une condition d'arrêt n'est pas satisfaite
   (a) Pour les m fourmis
      i. Créer un cycle initial en choisissant un ensemble arbitraire de trois nœuds.
      ii. Initialiser le p-voisinage de chacun des nœuds. $(O(N^2))$
      iii. Tant que le cycle ne contient pas tous les nœuds du graphe
         A. Choisir arbitrairement un nœud x qui ne se trouve pas encore sur le cycle.
         B. Répertorier toutes les insertions GENI possibles de x dans le cycle. $(O(p^4))$
         C. Ordonner les insertions selon le coût requis pour les effectuer. (algorithme QuickSort : $O(p^4 \cdot \log(p^4))$)
         D. Insérer x dans le cycle selon la règle de transition d'état (3.5).
         E. Mettre à jour le p-voisinage de chacun des nœuds en tenant compte que x est maintenant sur le cycle. $(O(N))$
      iv. Faire la mise à jour locale de la mémoire globale : appliquer l'équation (3.6) aux arêtes du cycle tout juste construit. $(O(N))$
   (b) Faire la mise à jour globale de la mémoire globale : appliquer l'équation (2.4) à toutes les arêtes du graphe. $(O(N^2))$

TAB. 3.1 – Algorithme du système de colonie de fourmis GENI (GACS).

à $O(NC \cdot m \cdot (n \cdot p^4 + n^2))$, avec NC le nombre d'itérations après lequel l'algorithme est arrêté et p le p-voisinage de l'algorithme GENI.

3.3.3 Post-optimisation

Comme pour le ACS, il nous est possible de post-optimiser les solutions obtenues par les fourmis GENI. Ainsi, après la boucle 2a et juste avant la mise à jour globale (étape 2b), nous pouvons insérer une procédure de post-optimisation, pour donner l'algorithme de la table 3.2. Nous choisirons évidemment la procédure US, complément naturel de GENI.

Nous avons tenté, dans ce chapitre, de présenter l'architecture du système de fourmis GENI comme une émanation logique du système de fourmis original. Nous avons donc commencé par mettre ce dernier en morceaux pour en identifier les composantes. Nous avons ensuite remplacé certains éléments pour reconstruire un algorithme modifié. Nous nous retrouvons ainsi avec un algorithme hybride dont nous étudierons les résultats dans le prochain chapitre.

En particulier, il sera intéressant de voir comment réagissent les paramètres inchangés du système de fourmis: ρ, α. Et, inversement, nous pourrons voir comment l'adjonction d'une mémoire à GENI influence ses résultats.

1. Initialiser : lire le problème, initialiser les structures de données, etc.
2. Tant qu'une condition de fin n'est pas satisfaite
   (a) Pour les m fourmis
      i. Créer un cycle initial en choisissant un ensemble arbitraire de trois nœuds.
      ii. Initialiser le p-voisinage de chacun des nœuds. $(O(N^2))$
      iii. Tant que le cycle ne contient pas tous les nœuds du graphe
         A. Choisir arbitrairement un nœud x qui ne se trouve pas encore sur le cycle.
         B. Répertorier toutes les insertions GENI possibles de x dans le cycle. $(O(p^4))$
         C. Ordonner les insertions selon le coût requis pour les effectuer. (algorithme QuickSort : $O(p^4 \cdot \log(p^4))$)
         D. Insérer x dans le cycle selon la règle de transition d'état (3.5).
         E. Mettre à jour le p-voisinage de chacun des nœuds en tenant compte que x est maintenant sur le cycle. $(O(N))$
      iv. Faire la mise à jour locale de la mémoire globale : appliquer l'équation (3.6) aux arêtes du cycle tout juste construit. $(O(N))$
      v. **Pour chacune des m fourmis, optimiser la solution obtenue grâce à la procédure US.** $(O(m \cdot (n \cdot p^4 \cdot n^2)))$
   (b) Faire la mise à jour globale de la mémoire globale : appliquer l'équation (2.4) à toutes les arêtes du graphe. $(O(N^2))$

TAB. 3.2 – Algorithme du système de colonie de fourmis GENI dont les solutions sont post-optimisées par US (GUACS).

Chapitre 4

Résultats

Au chapitre précédent, nous avons décrit un nouvel algorithme, dérivé hybride de l'heuristique GENI et du système de colonie de fourmis (Ant Colony System, ACS). Afin de déterminer 1° si notre raisonnement est valable et 2° si l'algorithme possède un quelconque intérêt pratique, nous avons observé son comportement face au problème pour lequel il a été conçu, le PVC symétrique.

Pour ce faire, nous avons d'abord manipulé de diverses manières l'algorithme présenté au chapitre précédent, afin de mieux saisir le rôle de chacun de ses paramètres et d'en déterminer les ordres de grandeur. Une fois que nous avons su utiliser adéquatement l'algorithme, nous avons comparé ses performances à celles de ses parents : GENI, d'une part, et l'ACS, d'autre part.

Notons que l'algorithme a été implanté en C et que toutes les expérimentations ont eu lieu sur une station de travail Sparc Ultra 1 dont le microprocesseur est cadencé à 140 MHz.

4.1 Contexte

Avant que nous ne présentions les résultats, quelques remarques s'imposent. Celles-ci ont trois objectifs : expliquer, d'abord, la provenance des chiffres apparaissant dans les tableaux et graphiques de la section suivante ; faire preuve de transparence scientifique, ensuite, et permettre ainsi au lecteur d'ajouter foi aux conclusions que nous émettrons ; faciliter la tâche, enfin, de ceux qui souhaiteraient reproduire nos résultats.

4.1.1 Robustesse

Nous avons effectué nos tests sur quatre instances de PVC symétrique : deux générées aléatoirement (50 et 100 nœuds) que nous appellerons alea50 et alea100 et deux tirées de la librairie de problèmes TSPLIB (berlin52 et kroA100, comportant respectivement 52 et 100 nœuds). De cette manière, nous nous sommes assuré que les paramètres que nous avons par la suite choisis restaient valides pour plusieurs problèmes, *i.e.* qu'ils permettaient à l'algorithme de « généraliser » correctement.

Pour ne pas baser nos conclusions sur des résultats atypiques, chaque expérience a été répétée dix fois. D'une expérience à l'autre, seule l'initialisation du générateur de nombres pseudo-aléatoires a changé[1]. Nous avons ensuite fait la moyenne de ces dix résultats et ainsi obtenu une « expérience moyenne ». Les conclusions que nous avons tirées par la suite sont basés sur ces résultats moyens.

Plus un problème est complexe, plus le nombre de ses solutions est élevé et plus large est le champ d'investigation d'une heuristique. Pour cette raison, chaque problème est exploré plus ou moins longtemps selon sa taille : les résultats présentés sont ceux que nous avons trouvés après N/m, $10^{0,5} \cdot N/m$ et $10 \cdot N/m$ itérations , où N est le nombre de nœuds du graphe et m est le nombre de fourmis du GACS.

1. Le germe (*seed*) du générateur de nombre pseudo-aléatoire était égal à 1 pour la première expérience, à 2 pour la deuxième, etc.

4.1.2 Paramètres

Nous avons vu au chapitre précédent que la coopération peut être vue comme la mise à jour parallèle d'une mémoire globale et que seule la mise à jour locale de la phéromone influence le degré de cette coopération. Pour mieux isoler les deux composantes du GACS, la mémoire et la coopération, nous avons donc annulé dans un premier temps cette mise à jour locale et n'avons considéré l'action que d'une seule fourmi ($m = 1$ et $\rho = 0$). Après avoir étudié α, β et q_0, nous nous sommes attardé à m et ρ. Il est à noter que les problèmes sur lesquels nous avons ajusté nos paramètres sont différents de ceux sur lesquels nous avons fait nos tests.

Le choix de p, la taille du voisinage dans l'heuristique GENI [17], s'est fait indépendamment des autres paramètres. Il est clair en effet que si l'on connaît une bonne valeur de p pour un problème particulier, nous n'avons qu'à réutiliser cette valeur pour le GACS. Nous avons déterminé empiriquement que $p = 3$ était un choix raisonnable pour les problèmes que nous souhaitions étudier.

Il n'est ni nécessaire ni même de bon aloi de considérer toutes les combinaisons de paramètres possibles. Lorsque nous en étudiions un, nous fixions les autres à des valeurs prédéterminées : $\alpha = 0,5$, $\beta = 0,5$ et $q_0 = 0,95$. Lors des expériences initiales, $m = 1$ et $\rho = 0$, comme l'avons mentionné. Lorsque nous étudiions la coopération, m avait pour valeur par défaut 10 et ρ, 0,5. Ajoutons que l'ordre d'insertion des nœuds de la procédure GENI était, par défaut, aléatoire.

Enfin, pour toutes les expériences, nous avons naturellement choisi $\tau_0 = 0$, c'est-à-dire une quantité de phéromone initiale nulle. Ainsi, les coûts ajustés étaient initialement égaux aux coûts réels puis ensuite diminués s'ils s'avéraient intéressants.

4.2 Expérimentations

Maintenant que le cadre de nos expériences est défini, nous pouvons présenter les résultats obtenus. Nous commençons par présenter les résultats d'une fourmi in-

dividuelle douée de mémoire. Nous enchaînons avec ceux d'une colonie de fourmis travaillant coopérativement.

4.2.1 Fourmi individuelle

Dans ce qui suit, nous étudions les paramètres qui ne concernent qu'une fourmi individuelle douée de mémoire. Nous verrons d'abord les paramètres qui permettent de contrôler la mémoire, β et α. Nous nous attarderons ensuite aux paramètres qui dirigent la stochastique de notre fourmi GENI, que nous avons présentée à la section 3.2, page 38.

Sensibilité à la mémoire, β

La sensibilité à la mémoire, contrôlée par le paramètre β, est sans doute l'aspect le plus important des fourmis, qu'elles soient PPV ou GENI. Rappelons que par ce paramètre, on attribue plus ou moins d'importance au coût réel d'une arête par rapport à la quantité de phéromone qui y est déposée. Commençons donc notre étude par ce paramètre.

Considérons le tableau 4.1. Il présente la moyenne des coûts après N, $10^{0,5} \cdot N$ et $10 \cdot N$ itérations, où N est le nombre de nœuds du problème considéré, pour plusieurs valeurs de β. Que peut-on y voir?

La mémoire est utile L'information essentielle que nous livre ce tableau est la suivante : les traces mémorielles aident GENI à se diriger plus rapidement vers de bonnes solutions. On constate en effet que l'on obtient de meilleures solutions moyennes lorsque $0 < \beta < 1$ que lorsque $\beta = 0$. En revanche, on remarque que lorsque la phéromone prend une trop grande importance, les solutions obtenues par GENI sont mauvaises : le biais mémoriel devient trop grand et GENI ne distingue plus les coûts réels. En fait, la représentation de la réalité que suggère la phéromone ne colle plus du tout avec ladite réalité...

TAB. 4.1 – Influence de β sur la moyenne des coûts. En gras, les meilleures moyennes.

Problème	It.	β					
		0	0,25	0,5	0,75	1	5
berlin52	N	7661,3	**7574,5**	7646,7	7677,2	7675,3	8091,1
	$10^{0,5} \cdot N$	7579,9	**7556,5**	7561,8	7595,8	7607,8	7893,8
	$10 \cdot N$	7548,6	7545,4	7545,1	**7542,0**	7549,4	7684,5
kroA100	N	21590	**21438**	21542	21544	21603	22156
	$10^{0,5} \cdot N$	21452	**21356**	21365	21387	21402	21680
	$10 \cdot N$	21376	21306	21313	**21302**	21303	21452
alea50	N	272,6	**272,5**	273,8	273,5	274,1	282,1
	$10^{0,5} \cdot N$	271,8	271,9	**271,8**	272,1	273,8	278,5
	$10 \cdot N$	271,4	**271,2**	271,3	271,6	272,6	273,8
alea100	N	802,2	**800,0**	803,0	804,3	810,0	827,8
	$10^{0,5} \cdot N$	798,1	**795,9**	796,4	798,2	800,9	815,2
	$10 \cdot N$	792,3	789,3	**788,2**	788,5	791,9	807,9

TAB. 4.2 – Améliorations des solutions trouvées grâce à la phéromone par rapport à celles trouvées sans.

Problème	Améliorations relatives		
	après N it.	après $10^{0,5} \cdot N$ it.	après $10 \cdot N$ it.
berlin52	1,1 %	0,3 %	0,1 %
kroA100	0,7 %	0,4 %	0,3 %
alea50	0,0 %	0,0 %	0,1 %
alea100	0,3 %	0,3 %	0,5 %

Bien que moins marquée sur les problèmes générés aléatoirement, l'amélioration de la qualité de la solution grâce à la mémoire est tout de même sensible. Un rapide calcul montre que les différences relatives entre le coût moyen lorsque $\beta = 0$ et le meilleur coût moyen sont légèrement plus faibles lorsque les problèmes sont générés aléatoirement (voir tableau 4.2).

La sensibilité optimale évolue au fil des itérations Le tableau 4.1 laisse supposer que la meilleure valeur de β croît avec le nombre d'itérations. Considérons les graphiques de la figure 4.1. Ils donnent à chaque itération la valeur de β ayant donné le meilleur résultat, β_{opt}[1]. Pour construire ces courbes, nous avons compilé les résultats obtenus par le GACS en faisant varier β de 0 à 1,0 par incrément de 0,05. À chaque itération, nous avons relevé la valeur de β pour laquelle la longueur du tour était la moindre.

Pour chacun des problèmes, nous avons fait une régression linéaire. Lorsque celle-ci donne un résultat probant (*i.e.* une corrélation suffisamment éloignée de 0), nous traçons le résultat de cette régression en pointillés et indiquons la corrélation (r^2) ainsi que la statistique F_0 mesurant l'adéquation de la régression[2].

1. Le β optimal est la valeur de β pour laquelle on obtient le tour le plus court, pour une itération donnée : $\beta_{opt} = \text{argmin}_\beta(L(\beta))$, avec $L(\beta)$ la longueur du tour obtenu avec les paramètres α, ρ et q_0 fixés à leur valeur par défaut.

2. Observons que la somme des carrés de la régression $SS_R = \sum_{i=1}^{n_{obs}} (\hat{\beta}_i - \bar{\beta})^2$ a un seul degré de liberté et que la somme des carrés de l'erreur $SS_E = \sum_{i=1}^{n_{obs}} (\beta_i - \hat{\beta}_i)^2$ en a $n_{obs} - 2$, avec n_{obs} le nombre

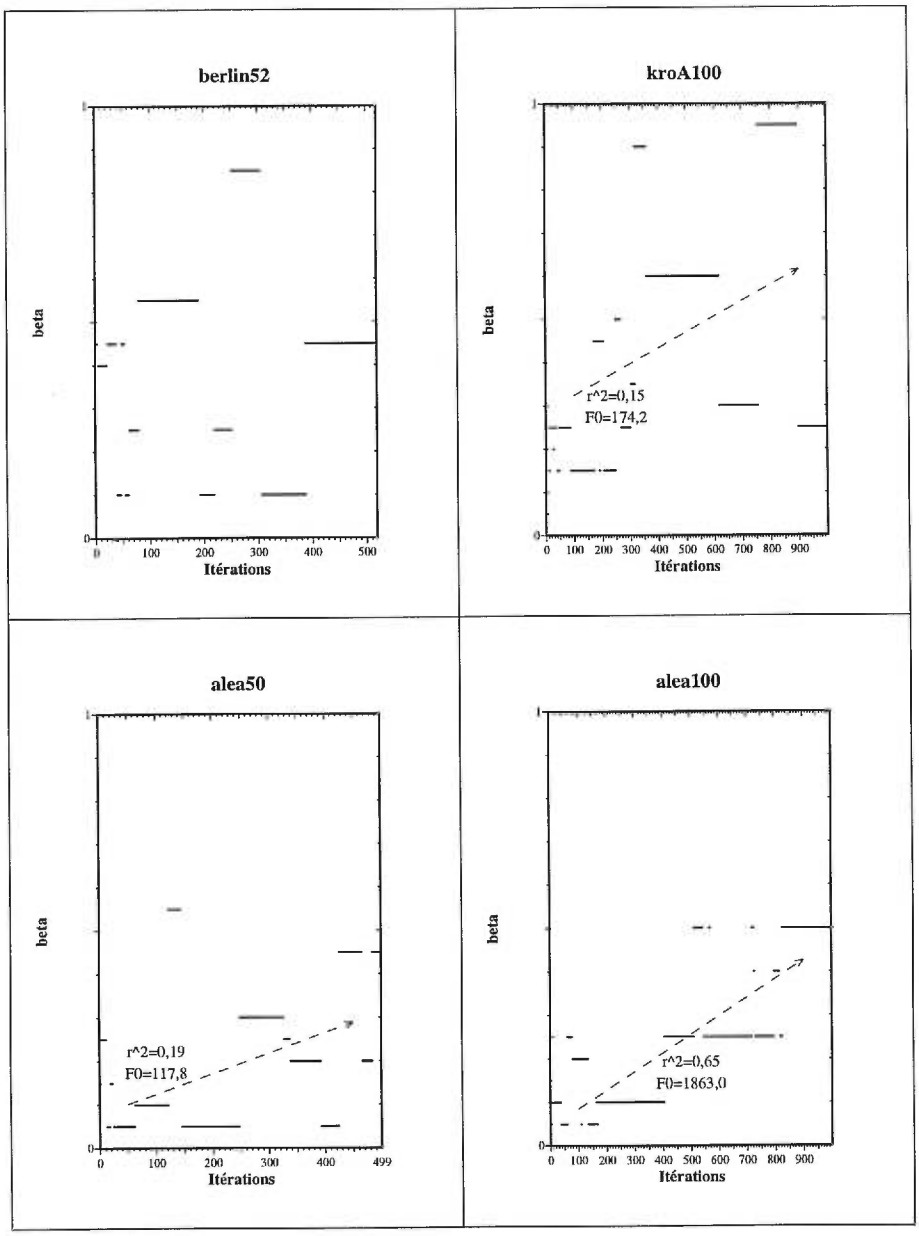

FIG. 4.1 – Pour les quatre problèmes-tests, l'évolution du β optimal au fil des itérations. Voir le texte pour des détails quant à la régression linéaire, r^2 et F_0.

Il apparaît que la sensibilité optimale change effectivement à mesure des itérations : le β optimal croît généralement avec le nombre d'itérations. Comment expliquer ce phénomène?

En début de processus, GENI explore le domaine des solutions et, pour être efficace, cette exploration se doit d'être la plus large possible. Or une forte sensibilité à la phéromone incite à l'exploitation des meilleures solutions ; mais, en début de processus, priviliégier une solution plutôt qu'une autre est illusoire car la base de solutions est encore trop réduite. Tomber dans un minimum local est inévitable. C'est pour cette raison que le choix d'un β faible apparaît comme un meilleure stratégie pour un nombre peu élevé d'itérations (de l'ordre de N itérations).

Chaque itération permet de découvrir une nouvelle solution. Si celle-ci est meilleure que la meilleure solution courante, elle remplacera cette dernière et sera renforcée lors des itérations suivantes (pendant que l'autre sera progressivement oubliée). C'est ainsi que la fourmi au β élevé comble son « inexpérience », itération après itération. Au bout d'un moment, elle est avantagée car elle a déterminé où se trouvent les meilleures solutions, exclusivement.

Les problèmes aléatoires nécessitent un biais moins élevé que les problèmes « réels » La figure 4.1 nous donne une indication supplémentaire sur β : il doit être moins élevé pour les problèmes générés aléatoirement.

Ceci peut s'expliquer par la structure-même de ces problèmes : les meilleures solutions ne sont pas nécessairement « proches » les unes des autres. C'est-à-dire que de petits mouvements (comme un échange 3-opt) ne donneront pas plus de chance

---

d'observations, égal ici au nombre d'itérations. Nous pouvons montrer que $E(SS_E/(n-2)) = \sigma^2$ et que $E(SS_R) = \sigma^2 + \gamma_1^2 S_{xx}$, avec σ la variance de l'erreur, γ_1 la pente de la droite de régression et $S_{xx} = \sum_{i=1}^{n_{obs}} (x_i - \bar{x})^2$ la somme corrigée des carrés de x. Avec les sommes SS_R et SS_E indépendantes, si $\gamma_1 = 0$, la statistique $F_0 = \frac{SS_R/1}{SS_E/(n_{obs}-2)}$ suit une distribution F avec 1 degré de liberté au numérateur et $n_{obs} - 2$ degrés de liberté au dénominateur, abrégée $F_{1;n_{obs}-2}$. Ainsi, si la statistique F_0 est supérieure à $F_{0,99;1;n_{obs}-2} \approx 6,63$, on peut rejeter avec 99% de certitude l'hypothèse voulant que cette relation n'existe pas.

de trouver une meilleure solution qu'un changement radical. Dans ces conditions, la mémoire est moins bénéfique puisqu'elle concentre la recherche sur une région particulière de l'espace des solutions.

Renforcement de la meilleure solution, α

Le renforcement des meilleures solutions (*i.e.*, la mise à jour globale) permet, en théorie, d'amener la fourmi GENI à se diriger plus rapidement vers les meilleures régions de l'espace des solutions. Le renforcement est contrôlé par le paramètre α et le tableau 4.3 présente les solutions moyennes obtenues pour différentes valeurs de ce paramètre et pour les quatre problèmes-tests.

Rappelons que, pour calculer le coût ajusté de l'arête (i, j), nous devons diviser la quantité de phéromone présente sur une arête (i, j), $\tau(i, j)$, par la quantité de phéromone présente sur l'arête qui en possède le plus, $\tau_{max}(i, j)$. Lorsque $\alpha = 0$, la quantité de phéromone est constante puisqu'il n'y a alors ni renforcement, ni « évaporation ». Ainsi $\tau_{max} = \tau_0 = 0$ et la division pose problème. Nous pouvons le contourner en remarquant que la situation pour laquelle $\alpha = 0$ (quantité constante de phéromone sur les arêtes) équivaut à celle pour laquelle $\beta = 0$ (la phéromone n'influence pas la fourmi). Nous avons par conséquent indiqué la qualité des solutions trouvées lorsque $\beta = 0$ dans le tableau 4.3.

Le tableau 4.3 montre qu'il n'existe apparemment pas de « meilleure » valeur pour α mais que, de façon générale, le renforcement de la meilleure solution trouvée permet de trouver de meilleures solutions que lorsque la phéromone reste à niveau constant.

Bien que moins marqué, le phénomène que nous décrivions plus haut (*i.e.* plus le nombre d'itérations permis est grand, plus l'influence de la phéromone peut l'être aussi) se retrouve. Examinons, par exemple, les graphiques de la figure 4.2 qui représentent les meilleures valeurs de α, itération après itération. Ici, nous avons relevé les solutions obtenues par le GACS à chaque itération pour $\alpha \in [0.05; 1]$, à intervalle de 0,05. Ici encore, nous avons indiqué les régressions probantes ainsi que les coefficients

TAB. 4.3 – Influence de α sur la moyenne des coûts. Nous indiquons aussi les valeurs obtenues lorsque $\beta = 0$ puisque cela correspond à la situation où le niveau de phéromone reste constant ($\alpha = 0$).

Problème	It.	$\beta = 0$	α				
			0,1	0,25	0,5	0,75	1
berlin52	N	7661,3	**7603,8**	7648,8	7646,7	7658,3	7618,2
	$10^{0,5} \cdot N$	7579,9	7555,6	7561,0	7561,8	7561,4	**7554,2**
	$10 \cdot N$	7548,6	**7542,1**	7546,9	7545,1	7552,3	7544,6
kroA100	N	21590	21480	21498	21542	**21478**	21511
	$10^{0,5} \cdot N$	21452	21371	21407	**21365**	21371	21416
	$10 \cdot N$	21376	21314	21314	21313	**21296**	21311
alea50	N	**272,6**	273,0	273,4	273,8	272,9	273,5
	$10^{0,5} \cdot N$	271,8	272,0	**271,7**	271,8	271,8	271,8
	$10 \cdot N$	271,4	**271,2**	271,6	271,3	271,3	271,3
alea100	N	**802,2**	804,5	805,8	803,0	807,2	806,2
	$10^{0,5} \cdot N$	798,1	798,1	796,1	796,4	798,7	**795,1**
	$10 \cdot N$	792,3	790,0	790,0	**788,2**	790,1	788,3

TAB. 4.4 – Influence de l'ordre d'insertion sur la moyenne des coûts. En gras, les meilleures moyennes.

Problème	It.	Ordre d'insertion	
		constant	aléatoire
berlin52	$10 \cdot N$	7876,0	**7545,1**
kroA100	$10 \cdot N$	21449	**21313**
alea50	$10 \cdot N$	276,0	**271,3**
alea100	$10 \cdot N$	813,4	**788,2**

de corrélation et la statistique d'adéquation de régression (voir la note 2, à la page 51).

Nous ne pouvons que constater la similitude de ces courbes avec celles présentées à la figure 4.1 (page 52). Ceci confirme ce que nous indiquions précédemment sur le rapport entre l'influence de la phéromone et le nombre d'itérations du GACS.

Condition initiale

Nous avons vu au chapitre précédent que l'ordre d'insertion des nœuds détermine la région de l'espace des solutions qui est explorée. Nous avons voulu déterminer s'il est bien important de changer de condition initiale d'une construction à l'autre. Pour ce faire, nous avons comparé les résultats obtenus par la fourmi GENI lorsque les nœuds sont systématiquement insérés dans le même ordre[1] avec ceux obtenus lorsque l'ordre d'insertion est aléatoire. Le tableau 4.4 synthétise ces résultats.

Il est important de faire varier l'ordre d'insertion des nœuds La conclusion est simple : une bonne variabilité de l'ordre d'insertion est importante pour obtenir de bons résultats.

1. Les problèmes sont décrits comme une séquence de nœuds auxquels sont associées leurs coordonnées. Lorsque nous spécifions que l'ordre d'insertion est constant, les nœuds sont insérés dans l'ordre dans lequel ils apparaissent dans la description du problème.

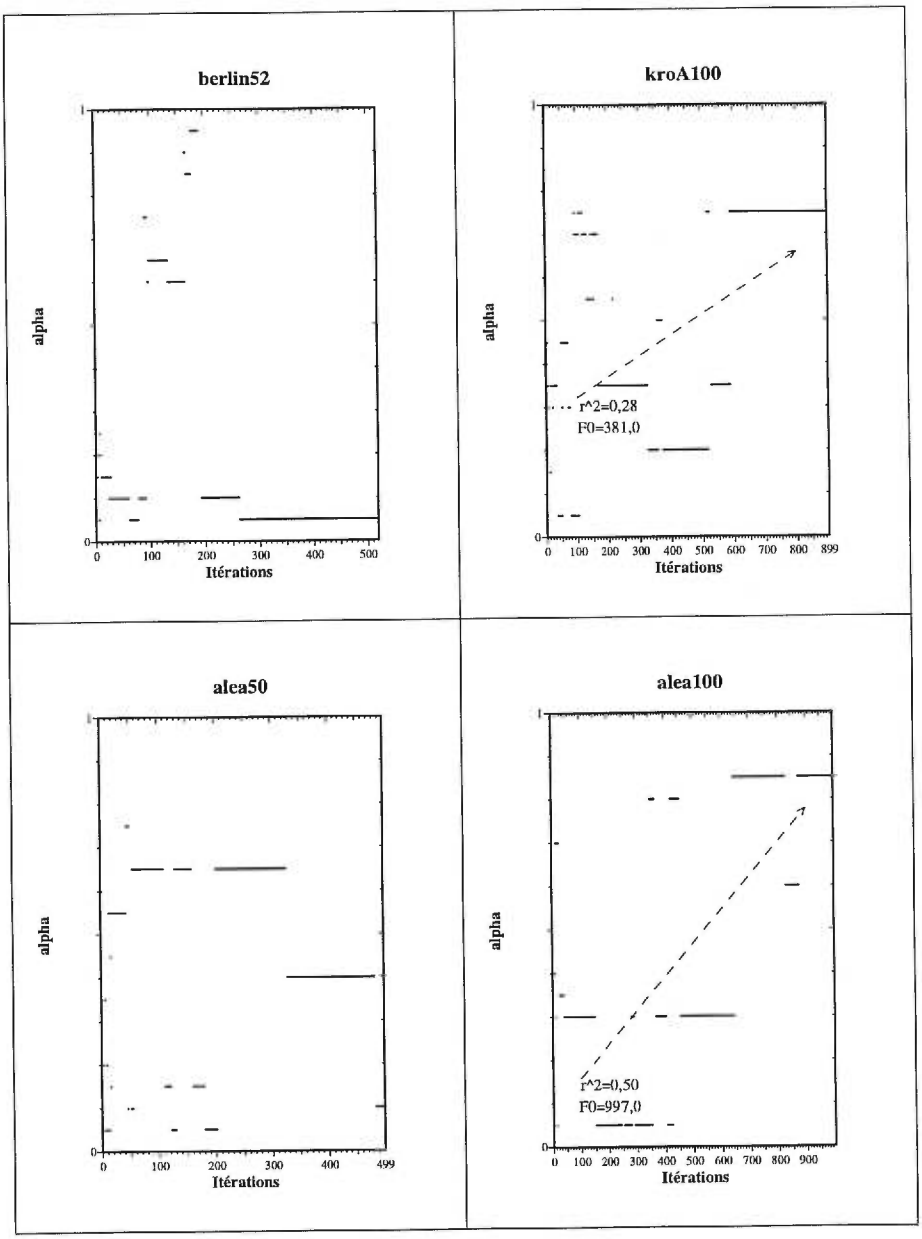

FIG. 4.2 – Pour les quatre problèmes-tests, l'évolution du α optimal au fil des itérations. Voir le texte pour des détails quant à la régression linéaire, r^2 et F_0.

TAB. 4.5 – Influence de q_0 sur la moyenne des coûts. En gras, les meilleures moyennes.

Problème	It.	q_0					
		0	0,5	0,9	0,95	0.98	1
berlin52	N	9473,5	8027,1	7674,6	7646,7	7643,1	**7639,6**
	$10^{0,5} \cdot N$	9032,6	7872,3	7576,3	**7561,8**	7593,9	7564,6
	$10 \cdot N$	8858,2	7719,6	**7543,9**	7545,1	7551,6	7551,6
kroA100	N	28552	23122	21538	21542	21428	**21394**
	$10^{0,5} \cdot N$	27765	22766	21447	21365	21349	**21333**
	$10 \cdot N$	27187	22628	21352	21313	**21307**	21308
alea50	N	323,3	284,7	275,0	273,8	273,2	**272,6**
	$10^{0,5} \cdot N$	317,2	282,3	272,7	271,8	271,6	**271,4**
	$10 \cdot N$	311,3	276,9	271,3	271,3	271,3	**271,2**
alea100	N	1044,6	868,8	804,7	803,0	799,0	**797,9**
	$10^{0,5} \cdot N$	1014,3	855,5	798,7	796,4	791,3	**788,1**
	$10 \cdot N$	988,3	841,7	791,3	788,2	**785,6**	785,8

Degré d'exploration, q_0

Le tableau 4.5 donne la moyenne des coûts obtenus pour chaque valeur de q_0 testée et pour chaque problème. q_0 est le degré d'exploration de la fourmi étudiée : $q_0 = 1,0$ implique que la fourmi ne prend pas de risque en cherchant une solution à un problème et opte toujours pour le mouvement qui lui « coûte » le moins.

L'aléatoire a un effet bénéfique à long terme Durant les premières itérations, il est plus intéressant de fixer q_0 à 1 et, ainsi, de ne permettre aucun mouvement contre-optimal. Par contre, lorsque le nombre d'itérations permis est suffisamment élevé, il devient intéressant de perturber très légèrement GENI. Pourquoi ?

Le déterminisme de GENI est une arme efficace en début de processus et lui permet de se diriger rapidement là où se trouvent les meilleurs résultats. Ce faisant, il place les jalons qui guideront ses futures recherches mais qui, par là-même, les limiteront.

La convergence du GACS combinée à celle du GENI déterministe amène la fourmi à reprendre les mêmes chemins, itération après itération. Choisir $q_0 < 1$ permet de conserver, même dans cette situation, une certaine variabilité des solutions. Cette opération peut se comparer à l'opérateur de mutation des algorithmes génétiques, qui permet de diversifier la population.

Ce résultat n'est pas en contradiction avec ceux obtenus par le système de foumis PPV : les expériences mentionnées dans l'article [11] sont généralement effectuées avec $q_0 = 0,98$. Cela ne laisse qu'une infime place à l'aléatoire. Cependant, un q_0 égal à 1 n'est pas envisageable dans ce cas puisque l'heuristique du PPV n'a à sa disposition que N conditions initiales (une fourmi ne peut débuter son cycle qu'à partir d'un seul nœud). En comparaison, $N!$ séquences de nœuds différentes peuvent être insérées par GENI. Dans le PPV, on peut évidemment modifier le critère d'insertion et choisir un nœud au hasard plutôt que le plus proche voisin. Dans ce cas, nous aurions aussi $N!$ séquences possibles ; mais le PPV en serait dénaturé et donnerait probablement de moins bons résultats.

4.2.2 Fourmis coopératives

Nous avons vu comment les fourmis se comportent individuellement et avons déterminé de « bonnes » valeurs pour les paramètres, c'est-à-dire des valeurs qui donnent des résultats acceptables. Pour compléter notre étude des paramètres, il nous faut voir si la coopération a véritablement une influence sur les résultats.

La coopération est contrôlée par deux paramètres qui ne vont pas l'un sans l'autre : m, le nombre de fourmis, et ρ, le paramètre de diversification de la recherche. Comme précédemment, nous avons fait varier un paramètre à la fois. : lorsque nous étudiions les valeurs de m, ρ était égal à 0,5 (voir la figure 4.3[1]); lorsque ρ variait, nous avons fixé m à 10 (voir le tableau 4.6). Les autres paramètres ont conservé leur valeur par défaut.

1. Dans cette figure, la valeur pour $m = 1$ est celle que nous avons obtenue lorsque $\rho = 0$. En effet, la mise à jour locale n'a pas de sens lorsque le problème n'est exploré que par une seule fourmi.

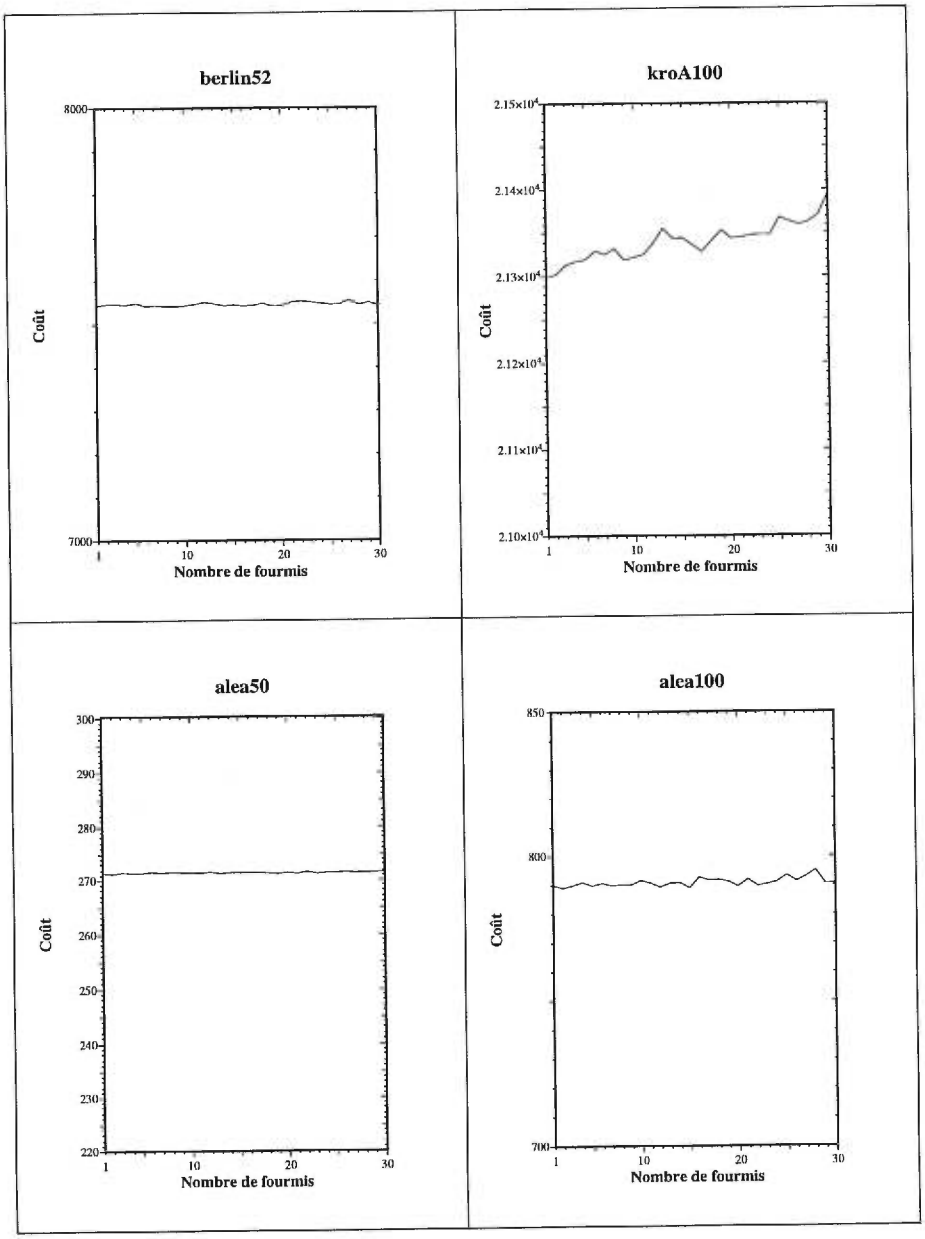

FIG. 4.3 – Évolution de la solution moyenne obtenue après $\frac{10 \cdot N}{m}$ itérations, selon le nombre de fourmis du GACS, pour les quatre problèmes-tests.

TAB. 4.6 – Influence du facteur de diversification ρ sur la moyenne des coûts. En gras, les meilleures moyennes.

Prob.	It.	ρ					
		0	0,1	0,25	0,5	0,75	1
berlin52	N/m	**7592,8**	7598,8	7622,3	7621,6	7662,2	7654,9
	$10^{0,5} \cdot N/m$	7555,5	7556,9	**7552,4**	7559,1	7558,6	7589,5
	$10 \cdot N/m$	7547,0	7550,1	7547,1	**7543,5**	7545,1	7546,9
kroA100	N/m	21441	**21420**	21447	21502	21568	21542
	$10^{0,5} \cdot N/m$	**21345**	**21345**	21353	21385	21414	21407
	$10 \cdot N/m$	**21298**	21304	21308	21322	21359	21334
alea50	N/m	273,2	272,7	271,9	272,1	**271,7**	272,8
	$10^{0,5} \cdot N/m$	271,6	**271,4**	**271,4**	271,6	271,5	271,8
	$10 \cdot N/m$	271,4	271,2	**271,0**	271,2	**271,0**	271,4
alea100	N/m	802,9	802,7	802,6	**801,3**	**801,3**	802,9
	$10^{0,5} \cdot N/m$	796,0	796,6	798,1	796,1	**794,7**	797,8
	$10 \cdot N/m$	**787,8**	789,7	789,8	791,6	790,2	794,0

La coopération ne permet pas clairement d'atteindre de meilleurs résultats Le tableau 4.6 et la figure 4.3 nous donnent une information intéressante : les résultats obtenus par plusieurs fourmis partageant une mémoire globale sont équivalents, globalement, à ceux auxquels parvient une fourmi individuelle. Et, si l'on se fie au problème kroA100, on se rend compte que la coopération peut même *nuire* à la recherche de solutions lorsque l'on combine une mise à jour locale trop aggressive avec un nombre élevé de fourmis. Dans ce cas, en effet, les chemins les moins coûteux sont défavorisés puisque progressivement dénués de phéromone...

La mise à jour locale de la phéromone n'apparaît pas *a priori* comme une opération clé du GACS et le choix de ρ semble devoir être arbitraire. Cependant, rappelons-nous que l'opération de mise à jour globale est relativement coûteuse (parce qu'appliquée à *toutes* les arêtes) et, ainsi, lorsque $m = 1$, le temps de calcul est bien plus élevé que lorsque $m = 10$, pour un même nombre de tours construits. Pour cette raison, nous avons opté pour $m = 10$, accompagné d'une légère dispersion, $\rho = 0,1$.

4.3 Comparaison

Maintenant que nous savons comment réagit le GACS, nous pouvons comparer sa performance à celle des algorithmes dont il est issu : GENI et le ACS. Nous commencerons par examiner si une simple succession de GENI (appelé le *GENI itéré* dans ce qui suit) permet d'atteindre d'aussi bons résultats que le GACS.

4.3.1 GENI itéré

Voyons donc si l'adjonction d'une mémoire à GENI lui permet d'atteindre de meilleurs résultats. Pour cela, nous avons comparé les résultats obtenus par le GENI itéré et par le GACS sur 6 problèmes réels, tous tirés de la librairie TSPLIB : kroA100, d198, lin318, pcb442, rat783, u1060.

Nous avons choisi pour le GACS les paramètres suivants :

- $\alpha = 0,1$: nous voulons trouver rapidement de bonnes solutions, or nous avons vu que, moins le nombre d'itérations permis est élevé, plus α doit être petit (section 4.2.1) ;
- $\beta = 0,1$: nous invoquons la même raison que précédemment (section 4.2.1) ;
- $m = 10$: bien que l'influence de la coopération soit peu quantifiable, $m = 10$ nous assure que la (coûteuse) mise à jour globale ne sera effectuée qu'une fois sur dix[1] (section 4.2.2);
- $\rho = 0,1$: cela n'éloigne pas trop les fourmis des régions intéressantes de l'espace des solutions (section 4.2.2);
- $q_0 = 1$: nous avons vu que la stochastique est intéressante lorsque les fourmis travaillent longtemps ; or, nous souhaitons avoir de bons résultats rapidement (voir 4.2.1).

Pour le GACS comme pour le GENI itéré, nous avons choisi $p = 5$ pour tous les problèmes. Les résultats présentés sont la moyenne sur 10 essais. D'un essai à l'autre, l'ordre d'insertion est différent puisque les nœuds sont tirés d'une distribution pseudo-aléatoire uniforme. Cependant, pour que les comparaisons soient plus justes, les ordres d'insertion des essais 1, 2, 3, etc. respectivement du GENI itéré sont les mêmes que pour les essais 1, 2, 3, etc. respectivement du GACS (*i.e.* le germe (*seed*) du générateur de nombre pseudo-aléatoire y est identique).

Les figures 4.4, 4.5 et 4.6 montrent l'évolution dans le temps du coût des solutions trouvées par le GENI itéré et le GACS. De manière générale, on constate que le GACS parvient à de meilleurs résultats qu'une simple succession de GENI.

Nous pouvons donc affirmer 1° que GENI est sensible (de façon plus ou moins significative) à l'adjonction d'une mémoire et 2° que la méta-heuristique des colonies de fourmis est suffisamment générique pour encadrer une autre heuristique de

1. Nous pourrions aussi choisir $m = 1$ et ne mettre à jour globalement la phéromone qu'une fois sur dix... Nous avons préféré ne pas modifier l'algorithme ACS.

TAB. 4.7 – Nombre de cycles construits avant d'atteindre la solution optimale lors de l'expérience ayant donné le meilleur résultat parmi 15 expériences, pour le GACS, GENI itéré et l'ACS. Entre crochet, le temps approximatif d'exécution en secondes.

Problème	GACS	GENI itéré	ACS
eil50	8 [0,1]	8 [0,1]	1830 [1,8]
eil75	1179 [137,9]	176 [18,3]	3480 [7,9]
kroA100	11 [1,7]	17 [2,6]	4820 [19,3]

construction que le PPV.

4.3.2 ACS

Nous avons comparé le GACS avec le GENI itéré, afin de voir si l'apprentissage des meilleures solutions permettait d'améliorer un algorithme de construction de solutions. Nous avons pu constater que c'était généralement le cas. Voyons maintenant si ce nouvel algorithme permet de surpasser l'ACS original, basé sur un PPV stochastique. Les résultats de l'ACS que nous fournissons ici sont issus de l'article [11].

Sans optimisation

Considérons le tableau 4.7. Dans ce tableau, on donne le nombre de cycles qu'ont dû construire le GACS, GENI et l'ACS, avant de parvenir à la solution optimale du problème considéré. Nous prenons le résultat de l'expérience ayant donné le meilleur résultat, parmi 15 expériences (les résultats publiés ont dicté ce choix). Les paramètres utilisés sont les suivants :

- GENI : $p = 5$;
- GACS : $p = 5$, $\alpha = \rho = \beta = 0,1$, $q_0 = 1$, $m = 10$;
- ACS : $m = 20$ et les autres paramètres ne sont pas indiqués dans l'article [11].

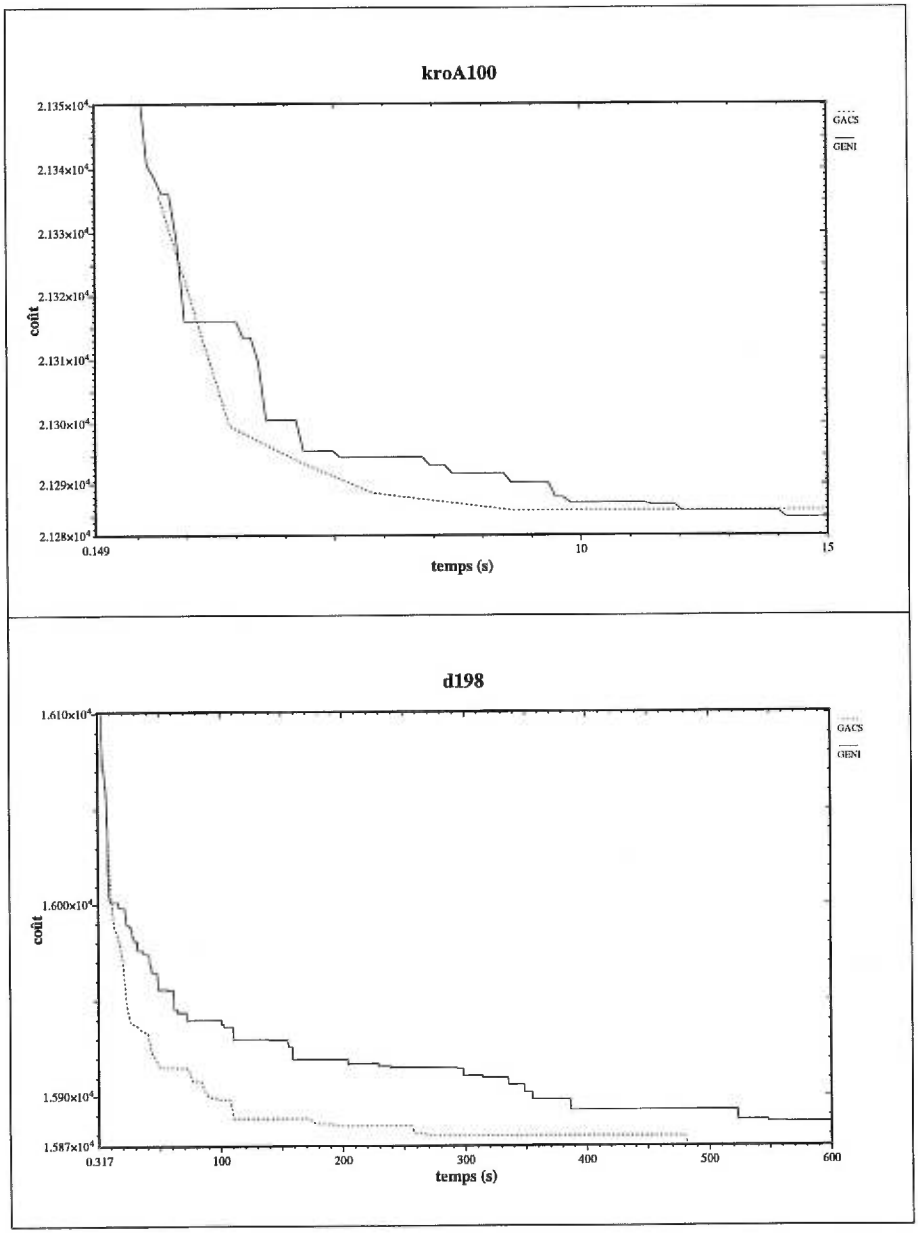

FIG. 4.4 – Évolution dans le temps du coût des solutions trouvées par le GENI itéré et le GACS (kroA100 et d198).

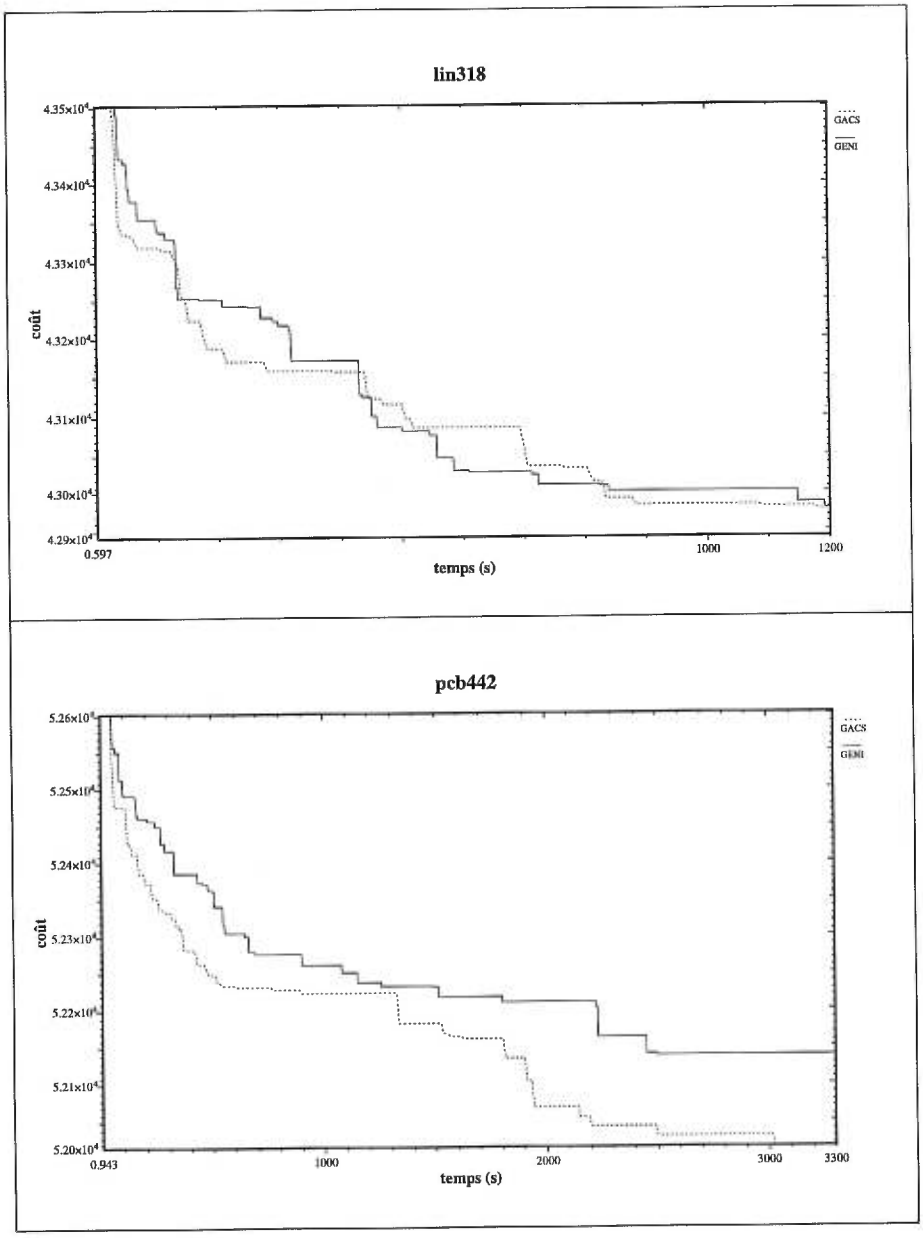

FIG. 4.5 – Évolution dans le temps du coût des solutions trouvées par le GENI itéré et le GACS (lin318 et pcb442).

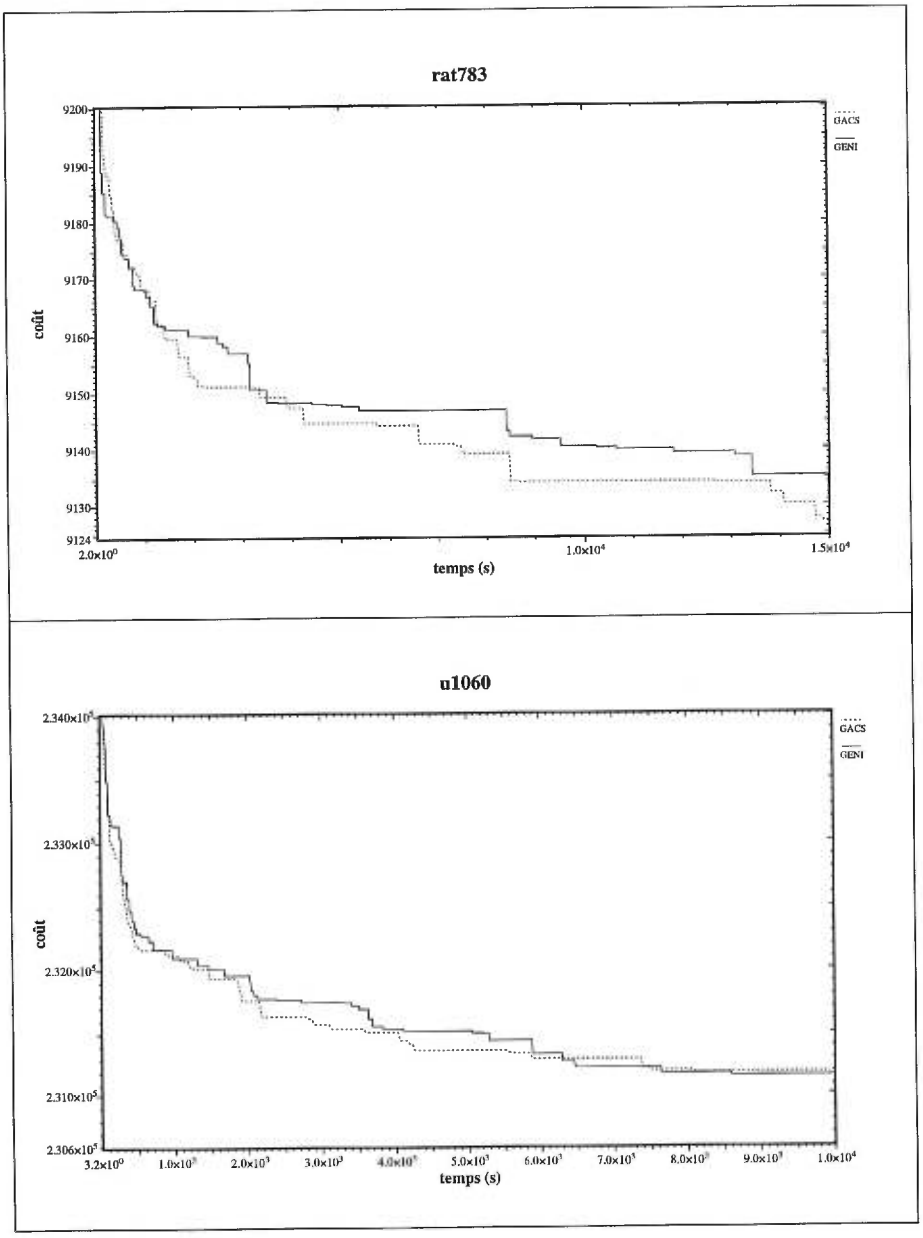

FIG. 4.6 – Évolution dans le temps du coût des solutions trouvées par le GENI itéré et le GACS (rat783 et u1060).

GENI et le GACS donnent de meilleurs résultats pour les problèmes de 50 et 100 nœuds mais sont dépassés par l'ACS pour le problème de 75 nœuds (qui semble présenter une topologie particulière). Bien sûr, ces résultats n'ont qu'une faible valeur : d'une part, ce ne sont que les meilleurs sur 15 expériences et, d'autre part, ils ne présentent que la faculté de l'heuristique à trouver la solution optimale. Une comparaison valable consisterait à comparer les qualités de solution moyennes en début de processus.

Le tableau 4.8 présente les résultats obtenus par les deux systèmes de fourmis sur de plus grands problèmes. Pour accélérer le traitement de ces problèmes, Dorigo et Gambardella ont implanté une *liste de candidats* pour leur algorithme PPV. Il s'agit de répertorier pour chaque nœud, à l'initialisation, les cl nœuds les plus proches. L'heuristique du plus proche voisin cherche en priorité dans ce voisinage.

Les paramètres utilisés dans ces expériences ne sont pas spécifiés pas les auteurs, sauf $cl = 15$. Pour notre part, nous avons choisi $m = 10$, $\alpha = \rho = \beta = 0,1$, $q_0 = 1$ et $p = 5$.

Les temps de calcul indiqués sont approximatifs puisque les deux algorithmes n'ont pas été testés sur les mêmes machines : un Sparc Server à 50 MHz pour l'ACS et un Sparc Ultra 1 à 140 MHz pour le GACS. D'après les spécifications que nous avons pu obtenir, il semblerait que le premier soit environ trois fois moins rapide que le deuxième. Nous avons donc divisé les temps de calcul indiqués dans l'article [11] par trois.

Nous constatons deux choses. D'abord, les fourmis PPV sont plus efficaces qu'on ne pourrait le penser et trouvent des solutions de haute qualité. Il est vrai que l'ACS est itéré un très grand nombre de fois. Cependant, pour pcb442 et rat783, nous avons lancé le GACS pendant une durée équivalente et n'avons pas obtenu de meilleurs résultats. Peut-être devons-nous en conclure que, sur le long terme, les fourmis PPV arrivent parfois à de meilleures solutions que les fourmis GENI. Nous n'avons pas accès aux résultats obtenus lors des premières itérations mais il est permis de penser, au vu des résultats de la section 4.3.1, qu'ils seraient à l'avantage de GENI.

TAB. 4.8 – Comparaison entre le GACS et l'ACS sur des problèmes de plus grande dimension. c^* indique le coût de la meilleure solution parmi 15 solutions et t^* le temps requis (en secondes) pour atteindre cette solution ; $\bar{c}$ indique la moyenne sur 15 expériences du coût des solutions et $\bar{t}$ le temps moyen (en secondes). La colonne « Err. » indique l'erreur de la solution moyenne par rapport à la solution optimale (pour le problème fl1577, à défaut de connaître la solution optimale, nous en indiquons les bornes). « n.d. » indique que le chiffre n'est pas disponible.

Prob.	ACS					GACS				
	c^*	t^* (s)	$\bar{c}$	$\bar{t}$ (s)	Err.	c^*	t^* (s)	$\bar{c}$	$\bar{t}$ (s)	Err.
d198	15888	3900	16054	n.d.	1,7 %	**15837**	111	**15874**	664	0,6 %
pcb442	**51268**	9916	**51690**	n.d.	1,8 %	51782	1317	51978	8374	2,4 %
att532	28147	19382	28523	n.d.	3,0 %	**28145**	6262	**28223**	6569	1,9 %
rat783	**9015**	42955	**9066**	n.d.	3,0 %	9071	12268	9116	42478	3,6 %
fl1577	22977	150720	23163	n.d.	4,1–4,3 %	**22787**	6548	**22970**	9733	3,2–3,4 %

Notre deuxième constatation porte sur le temps de calcul : de manière générale, les fourmis GENI trouvent de bonnes solutions beaucoup plus rapidement que les fourmis PPV.

Avec optimisation 3-opt

Dorigo *et al.* ont proposé dans l'article [11] d'optimiser les solutions trouvées par les fourmis PPV selon une procédure 3-opt[1] (voir section 2.1.3 pour une description de 3-opt). Les résultats qu'ils obtiennent sont excellents. Il semble naturel, dans ces conditions, de reprendre l'idée pour le GACS en effectuant une post-optimisation de type US (GUACS).

Par curiosité, nous avons aussi testé une post-optimisation 3-opt du même type que celle utilisée par l'ACS. Seulement, sachant qu'une insertion GENI de type I est équivalente à un mouvement 3-opt (voir 2.2.2), une post-optimisation 3-opt n'aurait, seule, qu'un effet très réduit. Nous avons donc forcé GENI à donner des tours de moins bonne qualité en posant $q_0 = 0,95$. Il est à noter que, pour le GUACS, $q_0 = 1$

1. Les auteurs utilisent en fait une procédure 3-opt restreinte : seuls sont considérés les mouvements 3-opt n'affectant pas le sens de parcours du tour. De plus, la procédure intègre une optimisation 2-opt. Enfin, pour des raisons d'efficacité, la procédure ne tente de mouvement 3-opt ou 2-opt qu'entre des nœuds voisins. Nous avons choisi un voisinage de 30.

TAB. 4.9 – Comparaison du système de fourmis PPV post-optimisé (ACS-3opt), du système de fourmis GENI post-optimisé par US (GUACS) et par 3-opt (GACS-3-opt). La colonne $\bar{c}$ indique la moyenne sur 10 expériences du coût des solutions ; la colonne t indique le temps en secondes pris, en moyenne, pour atteindre cette solution ; et la colonne ϵ indique l'erreur relative par rapport à la solution optimale.

Prob.	ACS-3-opt			GUACS			GACS-3-opt		
	$\bar{c}$	t	ϵ	$\bar{c}$	t	ϵ	$\bar{c}$	t	ϵ
d198	**15781,7**	357	0,01 %	15820,3	1388	0,3 %	15862,0	693	0,52 %
lin318	**42029**	806	0,00 %	42767,6	3757	1,76 %	43075,7	1961	2,49 %
att532	**27718,2**	1215	0,11 %	28103,9	15445	1,51 %	28322,7	6692	2,30 %
rat783	**8837,9**	1920	0,36 %	9067,8	29778	2,97 %	9176,4	16435	4,21 %

donne de meilleurs résultats.

Le tableau 4.9 compare les résultats obtenus par l'ACS-3-opt, le GUACS et le GACS-3-opt. Ici encore, les temps de calcul sont approximatifs : l'ACS-3-opt a été exécuté sur un serveur SGI Challenge L à 200 MHz alors que les expériences du GUACS et du GACS-3-opt se sont déroulées sur une station Sun Ultra 1 à 140 MHz. La première machine est environ une fois et demie plus rapide que la deuxième. Nous avons donc multiplié les temps de résolution de l'ACS-3-opt par 1,5.

Les résultats indiquent que l'ACS-3-opt obtient de bien meilleurs résultats que le GUACS, aussi bien en terme de temps que de qualité de solution. La moins bonne stratégie consiste à post-optimiser les solutions trouvées par GENI par une procédure 3-opt, GENI procédant déjà à une optimisation 3-opt partielle.

Dans ce chapitre, nous avons présenté et analysé les résultats pratiques obtenus par le système de fourmis GENI. Notre démarche a consisté dans un premier temps à nous familiariser avec les divers mécanismes de l'algorithme. Pour cela, nous en avons manipulé un à un les paramètres. Grâce aux résultats obtenus, nous avons pu appliquer le système de fourmis GENI à la résolution de problèmes pratiques, de plus ou moins grande dimension.

Pour déterminer la véritable valeur de l'algorithme, nous l'avons comparé à l'algorithme GENI itéré sans mémoire et au système de fourmis PPV. Il nous est d'abord apparu que la mémoire et la coopération contribuaient effectivement à l'amélioration du comportement de GENI. Ensuite, les résultats montrent que le GACS parvient rapidement à de bonnes solutions, équivalentes semble-t-il à celles obtenues par l'ACS classique.

Ceci achève la présentation de l'heuristique que nous proposons. Le prochain chapitre consistera en la conclusion de ce document : la synthèse de notre démarche et de nos résultats. Nous y verrons aussi quels aspects nous n'avons pas approfondis et qu'il serait instructif d'aborder.

Chapitre 5

Conclusion

Dans ce qui précède, nous avons présenté une nouvelle heuristique pour le PVC basée sur la métaphore des colonies de fourmis : le système de colonie de fourmis GENI. Après avoir présenté le contexte de cette recherche, nous avons analysé en détail l'heuristique originale du système de colonie de fourmis et en avons déduit le nouvel algorithme. Afin de déterminer si notre analyse était bonne et si le nouvel algorithme est performant, nous avons soumis ce dernier à une série de tests.

Ces tests ont fait ressortir que l'algorithme était une alternative viable au système de colonie de fourmis PPV et, surtout, nous ont permis de répondre à la question que nous nous posions en introduction :

> Est-ce que d'autres algorithmes de construction peuvent utiliser efficacement la pondération des arêtes pour construire de bonnes solutions ?

Nous pouvons maintenant répondre par l'affirmative, au moins en ce qui a trait à l'heuristique GENI. Non seulement les solutions sont-elles bonnes mais elles sont généralement meilleures que celles qu'aurait pu atteindre l'algorithme s'il n'avait pas bénéficié de sa propre expérience.

D'autres heuristiques pourraient sans doute bénéficier du même traitement, quoique

nous n'ayons pas de résultats à fournir pour étayer cette hypothèse. Mais, au delà d'une nouvelle heuristique, nous suggérons dans ce document une marche à suivre qui ouvre la voie à ce type de manipulation.

Mieux encore, l'introduction des fourmis GENI permet d'envisager la collaboration de différentes espèces : une fourmi PPV serait-elle à même de bénéficier des traces que laisserait une fourmi GENI? Les traces reflètant la même information dans les deux cas, la communication ne paraît pas impossible. Un algorithme pourrait lancer à intervalle régulier une fourmi GENI parmi les fourmis PPV et accélérer leur convergence vers une région intéressante de l'espace des solutions. La complexité serait réduite par rapport au système de colonie de fourmis GENI et la performance sans doute meilleure qu'avec les seules fourmis PPV.

Une autre avenue, opposée à la précédente, pourrait être explorée. Nous avons vu au chapitre 4 que le nombre de fourmis influençait peu la qualité des solutions obtenues par l'heuristique GACS. D'un autre côté, un système mono-fourmi n'est pas très efficace par rapport à un système à dix fourmis, ceci étant dû à la complexité de la règle de mise à jour globale, $O(n^2)$, par rapport à celle de mise à jour locale, $O(n)$. La coopération (et donc la mise à jour locale) aurait donc une justification ne relevant pas de l'amélioration de la qualité des solutions mais bien de l'efficacité. Il est tentant dans ces conditions d'éliminer la mise à jour locale et de remplacer l'équation de mise à jour globale actuelle par une autre, plus efficace. Nous pensons en particulier à une mise à jour globale de l'ordre de $O(n)$ qui ne ferait que renforcer la meilleure solution sans diminuer la pondération des arêtes non choisies (*i.e.* une mise à jour sans évaporation). Cela permettrait en outre d'éliminer le paramètre ρ.

À propos des paramètres, bien que nous les ayons explorés soigneusement, il serait intéressant de mesurer avec plus de précision l'influence des uns et des autres sur la performance de l'algorithme.

L'heuristique que nous avons proposée permet de générer des solutions de bonne qualité en peu de temps. Celles-ci sont cependant de qualité bien inférieure à celles obtenues par les meilleurs heuristiques actuellement disponibles pour le PVC. S'il

s'agit de se mesurer à ces heuristiques performantes, le GACS pourrait trouver son utilité dans la création de solutions destinées à être post-optimisées. Nous avons vu que la post-optimisation US n'est pas la procédure de choix mais plusieurs autres existent, que l'on pourrait explorer.

Ce document se veut une analyse détaillée de la méta-heuristique du système de colonie de fourmis. Nous avons voulu déterminer si elle permettait d'encadrer efficacement un autre type d'heuristique ; ce qui est le cas. Cette étude nous a en outre permis d'étudier scientifiquement le phénomène de la collaboration au sein d'une population d'entités autonomes. Et, enfin, ce projet a constitué notre initiation à la recherche scientifique.

Bibliographie

[1] E. Aarts et J. K. Lenstra, éditeurs. *Local Search in Combinatorial Optimization.* John Wiley and Sons Ltd., 1997.

[2] A.V. Aho, J.E. Hopcroft, et J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[3] V. Bachelet et E.-G. Talbi Z. Hafidi, P. Preux. Vers la coopération des méta-heuristiques. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, 10:211–223, 1998.

[4] J.L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing*, 4:387–411, 1992.

[5] R.G. Bland et D.F. Shallcross. Large traveling salesman problems arising experiments in x-ray crystallography : A preleminary report on computation. *Operations Research Letters*, 8:125–128, 1989.

[6] B. Bullnheimer, R.F. Hartl, et C. Strauss. Applying the ant system to the vehicle routing problem. Dans *MIC97 - 2nd International Conference on Metaheuristics*, pages 1–12, Sophia-Antipolis, France, 1997.

[7] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Rapport technique 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[8] G. Clarke et J.W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[9] A. Colorni, M. Dorigo, et V. Maniezzo. Distributed optimization by ant colonies. Dans *ECAL91, European Conference on Artificial Life*, pages 134–142, 1991.

[10] M. Dorigo et L.M. Gambardella. A study of some properties of ant-q. Dans

PPSM IV–4th Int. Conf. Parallel Problem Solving from Nature, pages 656–665, Berlin, Germany, 1996. Springer-Verlag.

[11] M. Dorigo et L.M. Gambardella. Ant colony system : A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1:53–65, 1997.

[12] M. Dorigo, V. Maniezzo, et A. Colorni. The ant system : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics*, 26:29–41, 1996.

[13] H.A. Eiselt et G. Laporte. A combinatorial optimization problem arising in dartboard design. *Journal of the Operational Research Society*, 42:113–118, 1991.

[14] M.M. Flood. The traveling salesman problem. *Operations Research*, 4:61–75, 1956.

[15] L.M. Gambardella et M. Dorigo. Ant-q : a reinforcement learning approach to the traveling salesman problem. Dans *Proceedings ML-95, 12th Int. Conf. Machine Learning*, pages 252–260, Palo Alto, CA, 1995. Morgan Kaufmann.

[16] L.M. Gambardella, É.D. Taillard, et M. Dorigo. Ant colonies for the quadratic assignment problem. Rapport technique IDSIA-4-97, IDSIA, Lugano, 1997.

[17] M. Gendreau, A. Hertz, et G. Laporte. New insertion and post-optimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1094, 1992.

[18] M. Gendreau, A. Hertz, et G. Laporte. Vehicle routing : Modern heuristics. Dans Aarts et Lenstra [1], chapitre 9, pages 311–336.

[19] F. Glover. Tabu search, part I. *ORSA Journal on Computing*, 1:190–209, 1989.

[20] F. Glover. Tabu search, part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[21] B.L. Golden et W.R. Stewart Jr. Empirical analysis of heuristics. Dans Lawler et al. [32], chapitre 7, pages 207–249.

[22] M. Held et R.M. Karp. The traveling salesman problem and minimum-spanning trees. *Operations Research*, 18:1138–1162, 1970.

[23] M. Held et R.M. Karp. The traveling salesman problem and minimum-spanning trees : part II. *Mathematical Programming*, 6:62–88, 1971.

[24] B. Holldobler et E.O. Wilson. *The Ants*. Springer-Verlag, Berlin, 1990.

[25] J.J. Hopfield et D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[26] D. S. Johnson et L. A. McGeoch. The traveling salesman problem : a case study. Dans Aarts et Lenstra [1], chapitre 8, pages 215–310.

[27] D.S. Johnson. Local optimization and the traveling salesman problem. Dans M.S. Paterson, éditeur, *Automata, Languages and Programming*, pages 446–461. Springer, Berlin, 1990.

[28] D.S. Johnson et C.H. Papadimitriou. Computational complexity. Dans Lawler et al. [32], chapitre 3, pages 37–85.

[29] L.P. Kaelbling, L.M. Littman, et A.W. Moore. Reinforcement learning : A survey. *Journal of Artificial Intelligence Res.*, 4:237–285, 1996.

[30] B.H. Korte. Applications of combinatorial optimization. Dans M. Iri et K. Tanabe, éditeurs, *Mathematical programming : Recent developments and applications*, pages 1–55. Kluwer, Dordrecht, 1989.

[31] G. Laporte. The traveling salesman problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:231–247, 1992.

[32] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, et D.B. Shmoys, éditeurs. *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization.* John Wiley and Sons Ltd., Chichester, U.K., 1985.

[33] S. Lin et B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973.

[34] O. Martin, S.W. Otto, et E.W. Felten. Large-step markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11:219–224, 1992.

[35] M. Minsky. *The Society of Mind.* Simon and Schuster, New York, NY, 1986.

[36] I. Or. *Traveling Salesman-type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking.* Thès de Ph.D., Northwestern University, Evanston, IL, 1976.

[37] J. Y. Potvin. The traveling salesman problem: a neural network perspective. *ORSA Journal on Computing*, 5:328–48, Fall 1993.

[38] É.D. Taillard et L.M. Gambardella. Adaptative memories for the quadratic assignment problem. Rapport technique IDSIA-87-97, IDSIA, Lugano, 1997.

[39] É.D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[40] É.D. Taillard, L.M. Gambardella, M. Gendreau, et J.-Y. Potvin. La programmation à mémoire adaptative ou l'évolution des algorithmes évolutifs. *Calculateurs parallèles, réseaux et systèmes répartis*, 10, 1998.

[41] C.J.C.H. Watkins. *Learning with delayed rewards.* Thèse de Ph.D., Psychology department, University of Cambridge, U.K., 1989.