La programmation à mémoire adaptative
ou
l'évolution des algorithmes évolutifs

Éric D. Taillard[1] — Luca M. Gambardella[1]
Michel Gendreau[2] — Jean-Yves Potvin[2]

[1] *IDSIA, Corso Elvezia 36, CH-6900 Lugano.*
E-mail : {eric, luca}@idsia.ch

[2] *CRT et DIRO, Université de Montréal,*
C.P.6128, succ. Centre-Ville, Montréal H3C 3J7.
E-mail : michelg@crt.umontreal.ca, potvin@iro.umontreal.ca

RÉSUMÉ : *Cet article analyse les développements récents de plusieurs méta-heuristiques à mémoire et montre que les manières d'implanter l'une ou l'autre de ces méthodes générales d'optimisation ont tendance à se rapprocher. Une présentation unifiée de ces méthodes est proposée sous le nom de* programmation à mémoire adaptative*. Ce schéma de résolution peut se paralléliser de façon naturelle. Finalement, un certain nombre de méthodes récemment proposées pour la résolution de problèmes d'affectation quadratique et d'élaboration de tournées de véhicules sont passées en revue et présentées sous l'angle de la programmation à mémoire adaptative.*

ABSTRACT : *This article analyses recent developments of a number of meta-heuristics with memory and shows that these general solving methods are implemented in a more and more similar way. A unified presentation of these methods is proposed under the name of* adaptive memory programming*. This solving scheme can be conveniently parallelised. Finally, a number of recently proposed methods for quadratic assignment and vehicle routeing problems are reviewed and presented under an adaptive memory programming point of view.*

MOTS-CLÉS : *Méta-heuristique, algorithme évolutif, recherche avec tabous, algorithme génétique, recherche par dispersion, affectation quadratique, tournée de véhicule.*

KEYWORDS : *Meta-heuristics, evolutionary algorithms, taboo search, genetic algorithms, scatter search, quadratic assignment, vehicle routeing.*

1. Introduction

Les méthodes génériques et heuristiques d'optimisation, aussi appelées méta-heuristiques ou méthodes de recherche locale générales, se développent de manière explosive à l'heure actuelle : toutes les plus importantes conférences en recherche opérationnelle ont au moins une session, sinon un groupe de sessions entièrement consacrés aux méta-heuristiques ; la littérature leur fait également une place de plus en plus grande. Parmi les techniques qui connaissent le plus de succès, on peut citer par exemple les algorithmes génétiques, le recuit simulé, la recherche avec tabous, la recherche par dispersion et les systèmes de fourmis. Le succès de ces méthodes est dû à plusieurs facteurs, comme leur facilité d'implantation, leur capacité et leur flexibilité à prendre en considération des contraintes spécifiques apparaissant dans les applications pratiques ainsi que la qualité élevée des solutions qu'elles sont capables de trouver.

D'un point de vue théorique, l'usage de méta-heuristiques n'est pas vraiment justifié et les résultats obtenus sont plutôt faibles. Par exemple, il existe quelques théorèmes de convergence pour le recuit simulé [HAJ88] ou la recherche avec tabous [FAI92] mais qui sont inutilisables en pratique : ils établissent seulement que l'on a une probabilité très élevée de trouver une solution globalement optimale si un temps de calcul complètement disproportionné est alloué, temps supérieur à celui nécessaire pour énumérer complètement l'espace de toutes les solutions.

Cependant, en pratique, les performances de ces techniques sont excellentes et elles sont souvent extrêmement compétitives. Dans cette course à l'excellence, on observe que les méthodes les plus efficaces hybrident deux ou plusieurs méta-heuristiques. Il en résulte que des techniques ayant à peu près les mêmes principes de fonctionnement sont appelées par des noms très différents comme « hybride génétique », « recherche avec tabous probabiliste », « système de fourmis MIN-MAX ». Toutes ces méthodes partagent trois particularités : premièrement elles mémorisent des solutions ou des caractéristiques des solutions visitées durant le processus de recherche ; deuxièmement, elles font usage d'une procédure créant une nouvelle solution à partir des informations mémorisées et troisièmement, elles sont dotées d'une recherche locale, que ce soit une méthode gloutonne, une recherche avec tabous élémentaire ou un recuit simulé.

Considérant la grande similitude existant entre ces méthodes, une présentation unifiée se justifie, de même qu'un nom à la fois plus synthétique et plus général. Ceci permet également d'échapper aux idiotismes d'une école professant l'usage de termes spécifiques de moins en moins en relation avec le fonctionnement réel de la méthode. Nous proposons de regrouper sous un même toit ces méta-heuristiques et de les qualifier de programmation à mémoire adaptative. Ces termes ont déjà été proposés par Glover [GLO96] dans le contexte de la recherche avec tabous. Il est vrai que cette dernière a toujours été présentée comme étant dotée d'au moins une mémoire (la liste de tabous) et ouverte à toutes les composantes de l'intelligence artificielle. Cependant, les évolutions successives de l'implantation d'une recherche avec tabous pourraient facilement mener à une méthode sans liste de tabous (mais avec une autre forme de mémoire). Le nom de la technique devient par là difficile à justifier sinon pour des raisons historiques. Ainsi, le souci de Glover dans [GLO97] est de mieux qualifier une telle méthode sans tabous, basée sur d'autres ingrédients déjà proposés dans ses articles de base [GLO89, GLO90].

Un problème similaire peut se poser pour d'autres méta-heuristiques. Par exemple, des applications d'algorithmes génétiques ne codent plus une solution du problème sous la forme d'un vecteur binaire et les opérateurs de croisement et de mutation n'ont souvent plus rien à voir avec les opérateurs standard pour lesquels quelques résultats théoriques ont été établis. Nous allons donc commencer par faire une revue des principales méta-heuristiques à mémoire avant d'en faire une synthèse et de les présenter sous l'angle de la programmation à mémoire adaptative. Ensuite, nous montrerons comment paralléliser cette technique avant de terminer par la présentation de quelques-unes de ses applications pour les problèmes de l'affectation quadratique et de l'élaboration de tournées de véhicules.

2. Méta-heuristiques à mémoire

Dans cette section, nous allons passer en revue quelques méta-heuristiques dotées d'une mémoire. Précisons d'emblée que nous faisons une interprétation assez large des formes que peut prendre une mémoire : par exemple, nous considérons que la population d'un algorithme génétique ou d'une recherche par dispersion est une forme de mémoire, alors que d'autres auteurs [GLO97b] voient les algorithmes génétiques comme des méthodes sans mémoire.

2.1. Algorithmes génétiques

L'idée de base des algorithmes génétiques est de simuler le processus d'évolution des espèces sexuées. Une des théories de l'évolution est que deux mécanismes fondamentaux, lors d'une reproduction sexuée, permettent de créer un nouvel individu différent de ses parents. Le premier est le croisement de ces derniers qui consiste à combiner deux moitiés du patrimoine génétique de chacun des parents pour constituer le patrimoine génétique de l'enfant. Le second mécanisme est la mutation, ou la modification spontanée de quelques gènes de l'enfant. Le nouvel individu ainsi créé est différent de chacun de ses parents mais partage cependant certaines de leurs caractéristiques. Si le hasard fait que l'enfant hérite de « bonnes » caractéristiques, son espérance de vie est plus élevée et il aura par conséquent une plus grande chance de se reproduire qu'un individu taré. L'analogie entre cette théorie de l'évolution et une méta-heuristique pour l'optimisation combinatoire a été proposée par Holland en 1975. Elle peut être décrite comme suit :

Un individu est associé à une solution admissible du problème à résoudre. Initialement les solutions étaient codées sous la forme de vecteurs binaires. Dans les applications les plus récentes — hérésie notoire pour les puristes — les solutions sont représentées de manière naturelle, par exemple sous la forme d'une permutation pour le problème du voyageur de commerce ou de l'affectation quadratique. L'opérateur de croisement, qui consistait initialement à échanger des sous-chaînes des vecteurs correspondant aux codes des deux parents, est actuellement souvent remplacé par un opérateur mieux adapté au problème à résoudre. La mutation est un opérateur qui était invoqué occasionnellement et qui consistait à changer quelques éléments du vecteur-enfant. Actuellement, l'opérateur de mutation est parfois beaucoup plus complexe, allant jusqu'à effectuer une recherche avec tabous.

Une version élémentaire d'un algorithme génétique peut se formuler schématiquement comme suit :

Algorithme génétique

```
1) Choisir un code binaire pour représenter une
   solution et générer une population de vecteurs
   binaires.
2) Répéter les pas suivants, tant qu'un critère
   d'arrêt n'est pas vérifié.
   2a) Sélectionner deux vecteurs dans la population à
       l'aide d'un opérateur de sélection.
   2b) Combiner les deux vecteurs-parents à l'aide d'un
       opérateur de croisement pour obtenir un vecteur-
       enfant.
   2c) Éventuellement, appliquer un opérateur de muta-
       tion au vecteur-enfant.
   2d) Évaluer la qualité de la solution-enfant.
   2e) Insérer l'enfant dans la population.
   2f) Éventuellement, éliminer des vecteurs de la
       population avec un opérateur d'élimination.
```

La formulation de cet algorithme appelle quelques remarques : tout d'abord, il est souvent peu naturel de coder une solution sous la forme d'un vecteur binaire et suivant le schéma de codage choisi, l'algorithme peut produire des résultats très différents. Une telle propriété n'est pas vraiment souhaitable. Ensuite, un critère d'arrêt doit être choisi, comme c'est d'ailleurs le cas pour toutes les méta-heuristiques. Les algorithmes génétiques possèdent un critère d'arrêt naturel : la convergence de la population. Selon les opérateurs de sélection et d'élimination, on observe que les solutions de la population ont tendance à se ressembler, sinon à devenir toutes pareilles. Si l'on se trouve dans une telle situation, il convient d'arrêter la recherche. Cependant, la solution vers laquelle l'algorithme converge est souvent trouvée longtemps avant la convergence et l'on préfère généralement fixer a priori un nombre d'itérations que l'algorithme effectuera.

L'opérateur de sélection favorise typiquement l'élection des meilleures solutions de la population, ce qui permet une convergence plus rapide de l'algorithme, parfois au détriment de la qualité des solutions trouvées. Alors que dans les versions initiales le point-clé était de choisir un bon schéma de codage, les autres opérateurs faisant partie d'un ensemble standard, les implantations les plus récentes utilisent une représentation naturelle des solutions avec des opérateurs de croisement et de mutation fortement dépendants du problème à résoudre, et constituent les éléments-clé des nouvelles implantations.

Finalement, l'opérateur d'élimination consiste souvent à éliminer les solutions les plus mauvaises de la population, de sorte à ramener le nombre d'individus entre des valeurs préfixées.

Les implantations d'algorithmes génétiques sont innombrables et couvrent à peu près tous les domaines de l'optimisation.

2.2 Recherche par dispersion

La recherche par dispersion, ou « scatter search » en anglais, a été proposée par Glover [GLO77] mais cette technique n'a pas connu un succès comparable à celui des algorithmes génétiques, ce qui peut s'expliquer peut-être par le fait que la méthode n'a pas été présentée sous des traits autant aguichants. Pourtant, la recherche par dispersion est assez similaire aux algorithmes génétiques, du moins si l'on considère les implantations les plus récentes. Il n'est donc pas exagéré de prétendre que cette technique était très moderne pour son temps, même si elle n'a pas été appréciée à sa juste valeur.

La recherche par dispersion a été proposée dans le cadre de la résolution de programmes mathématiques en nombres entiers. Tout comme les algorithmes génétiques, elle est basée sur une population de solutions (vecteurs d'entiers) qui évolue dans le temps à l'aide à la fois d'un opérateur de sélection, de la combinaison linéaire de solutions de la population pour créer une nouvelle solution provisoire (non forcément entière ou admissible), d'un opérateur de projection permettant de rendre la solution provisoire admissible et d'opérateurs d'élimination. Ainsi, on peut voir la recherche par dispersion comme un algorithme génétique spécial présentant les particularités suivantes :

— Les vecteurs binaires sont remplacés par des vecteurs d'entiers.

— L'opérateur de sélection peut élire plus de deux solutions.

— L'opérateur de croisement est remplacé par une combinaison linéaire convexe ou non convexe de vecteurs.
— L'opérateur de mutation est remplacé par un opérateur de réparation ou de projection qui ramène la solution nouvellement créée dans l'espace des solutions admissibles.

Ces particularités peuvent également être vues comme des généralisations des algorithmes génétiques qui ont été proposées et exploitées ultérieurement par divers auteurs, notamment Mühlenbein, Georges-Schleuter & Krämer [MÜH88]. Citons en particulier :

— L'usage d'opérateurs de croisement différents de l'échange de sous-chaînes.
— L'application d'une recherche locale pour améliorer la qualité des solutions produites par l'opérateur de croisement.
— L'usage de plus de deux parents pour créer l'enfant.
— La subdivision de la population à l'aide de méthodes de regroupement en lieu et place d'un opérateur d'élimination élémentaire.

Du fait de sa redécouverte récente, les applications de la recherche par dispersion sont peu nombreuses ; on peut toutefois citer, dans l'ordre chronologique, outre la publication initiale de Glover [GLO77], une référence à cette dernière dans une application à des problèmes d'élaboration de tournées [ROC95], les travaux de Fleurent et al. [FLE96] dans le cadre de l'optimisation continue sans contraintes, ceux de Kelly, Rangaswamy & Xu [KEL96] pour la phase d'apprentissage des réseaux neuronaux ainsi que la méthode de Cung et al. [CUN97] qui permet d'obtenir des solutions de très bonne qualité pour les problèmes d'affectation quadratique.

2.3 Recherche avec tabous

Parmi les méta-heuristiques passées en revue dans cette section, la recherche avec tabous est la seule qui a été présentée expressément avec une mémoire, ou plus exactement un ensemble de mémoires. Le terme de recherche avec tabous a été proposé pour la première fois par Glover [GLO86]. L'idée à la base de cette technique est de modifier localement une solution de manière itérative, tout en gardant une trace de ces modifications pendant un certain laps de temps, afin d'éviter de réaliser les modifications inverses susceptibles de mener à des solutions déjà visitées. Les modifications, certaines de leurs caractéristiques ou encore des solutions entières sont mémorisées dans des listes qui interdisent pour les itérations futures l'usage de certaines modifications, d'où l'appellation de liste de tabous.

L'analogie avec le monde réel présidant à la recherche avec tabous est l'imitation de l'être humain cherchant la solution d'un problème d'optimisation combinatoire : tout d'abord, on construit une solution initiale, même de piètre qualité, puis on l'améliore petit à petit, par des modifications locales. Ces modifications n'améliorent pas forcément la solution à tous les coups mais on cherche à diriger la recherche dans une portion prometteuse de l'espace des solutions.

Plus tard, en 1989 et 1990, Glover a proposé un certain nombre de stratégies pour diriger la recherche et la rendre plus efficace, tout en précisant que la recherche avec tabous est ouverte à toutes les stratégies utiles pour le problème spécifique que l'on veut résoudre [GLO89, GLO90]. Il faut noter que ces nouvelles idées pour diriger la recherche n'ont été utilisées que beaucoup plus tard dans les implantations pratiques. On trouve encore à l'heure actuelle des articles décrivant des méthodes n'utilisant que les quelques ingrédients proposés dans le papier initial [GLO86] (une liste de tabous à court terme, « aspiration » d'une solution interdite si elle est la meilleure connue), ce qui ne signifie pas forcément qu'elles sont mauvaises, mais elles peuvent présenter certaines faiblesses dans la résolution de problèmes un peu particuliers.

L'usage d'une mémoire à long terme s'est répandu bien plus tard. On peut en voir un embryon dans notre implantation pour le problème de l'affectation quadratique [TAI91] où l'on force l'utilisation d'une modification jamais élue pendant un grand nombre d'itérations. Le pas suivant a été une pénalisation des modifications proportionnelle à leur fréquence d'élection comme dans [TAI93, TAI94], technique rapidement adoptée par d'autres auteurs comme [GEN94]. On peut voir là un premier exemple de mémoire adaptative, utilisée dans le but de diversifier le processus de recherche pour le forcer à explorer des portions non encore examinées de l'espace des solutions.

Par la suite, on s'est rendu compte de l'importance accrue qu'il fallait donner à la diversification de la recherche et actuellement on fait de plus en plus appel à une mémoire adaptative pour réaliser cette diversification. Une autre composante négligée dans les premières implantations de recherche avec tabous mais qui prend de plus en plus d'importance est l'intensification de la recherche dans une portion jugée prometteuse de l'espace des solutions. D'une manière extrêmement schématique, une recherche avec tabous peut être formulée comme suit :

Trame d'une recherche avec tabous

1) Générer une solution initiale $\boldsymbol{s}_0$, poser $\boldsymbol{s}^* = \boldsymbol{s}_0$, initialiser k à 0 ainsi que les mémoires.
2) Répéter, tant qu'un critère d'arrêt n'est pas satisfait :
   - 2a) Choisir $\boldsymbol{s}_{k+1}$ dans le voisinage de la solution $\boldsymbol{s}_k$, en prenant les mémoires en considération.
   - 2b) Si $\boldsymbol{s}_{k+1}$ est meilleur que $\boldsymbol{s}^*$, poser $\boldsymbol{s}^* = \boldsymbol{s}_{k+1}$ et incrémenter k.
   - 2c) Mettre à jour les mémoires.

Cette trame est exprimée en termes très généraux et chacune de ses étapes peut être d'une sophistication très variable. Par exemple, le choix de s_{k+1} à l'étape 2a peut consister simplement à examiner l'ensemble des solutions que l'on peut obtenir en appliquant une modification locale et non interdite à la solution s_k, comme dans les implantations les plus élémentaires ; mais ce choix peut également résulter d'un processus beaucoup plus complexe comme dans [ROC95] où l'on construit une nouvelle solution de toutes pièces en s'aidant des solutions précédemment produites par la recherche, solution qui est améliorée par une recherche avec tabous plus élémentaire. Ce mécanisme permet de réaliser à la fois une intensification et une diversification de la recherche.

2.4. Colonies de fourmis

L'idée d'imiter le comportement des fourmis pour trouver de bonnes solutions à des problèmes d'optimisation combinatoire a été proposée par Colorni, Dorigo & Maniezzo [COL91]. Le principe de cette méta-heuristique est basé sur la manière dont les fourmis cherchent leur nourriture et retrouvent leur chemin pour retourner dans la fourmilière. Initialement, les fourmis explorent les environs de leur nid de manière aléatoire. Sitôt qu'une source de nourriture est repérée par une fourmi, son intérêt est évalué (quantité, qualité) et la fourmi ramène un peu de nourriture au nid. Pour retrouver son chemin, elle a pris soin de laisser derrière elle une phéromone, trace chimique qu'elle arrive à détecter. Durant le chemin du retour, elle dépose également une quantité de phéromone dépendant de l'intérêt de la source de nourriture. Comme toutes les fourmis font de même, les traces laissées augmentent plus rapidement pour les sources de nourritures proches de la fourmilière, et lorsque plusieurs traces mènent à la même source, les traces correspondant aux chemins les plus courts sont renforcées à un rythme plus élevé. Il en résulte qu'après un certain temps, les chemins les plus rapides menant aux sources de nourriture les plus importantes sont marqués par de fortes traces.

Pour transposer ce comportement à un algorithme général d'optimisation combinatoire, on fait une analogie entre l'aire dans laquelle les fourmis cherchent de la nourriture et l'ensemble des solutions admissibles du problème, entre la quantité ou la qualité de la nourriture et la fonction-objectif à optimiser et enfin entre les traces et une mémoire adaptative. Une description détaillée de ces analogies pour le problème du voyageur de commerce peut être trouvée dans [DOR96]. Très schématiquement, un algorithme basé sur une colonie de fourmis peut être décrit comme suit :

Colonie de fourmis

```
1) Initialiser les traces.
2) Répéter en parallèle pour chacune des p fourmis et
   tant qu'un critère d'arrêt n'est pas satisfait :
   2a) Construire une nouvelle solution à l'aide des
       informations contenues dans les traces et une
       fonction d'évaluation partielle.
   2b) Évaluer la qualité de la solution.
   2c) Mettre à jour les traces.
```

Une particularité de cet algorithme est qu'il a été pensé pour une exécution concurrente, contrairement à la recherche avec tabous ou le recuit simulé. Le point crucial de cette méta-heuristique est la définition des traces et leur mise à jour selon le problème à résoudre. Pour un voyageur de commerce par exemple, on peut prévoir une trace pour chaque paire de villes. Initialement, deux mécanismes d'utilisation des traces ont été imaginés : l'exploration et l'exploitation. L'exploration consiste à choisir la prochaine composante utilisée pour construire la solution (par exemple une arête dans le cas du voyageur de commerce) avec une probabilité proportionnelle à la valeur de la trace associée à cette composante. L'exploitation consiste à choisir la composante qui optimise une fonction mariant la valeur de la trace et la fonction d'évaluation partielle (par exemple, pour le voyageur de commerce, une combinaison linéaire de la valeur de la trace et de la longueur de l'arête correspondante).

La mise à jour des traces a été proposée comme suit : tout d'abord, la valeur de toutes les traces est diminuée, pour simuler le phénomène d'évaporation des phéromones. Ensuite, les traces correspondant aux composantes choisies dans la construction de la solution sont renforcées d'une valeur croissant avec la qualité de la solution. Naturellement, les algorithmes basés sur une colonie de fourmis ont aussi évolué. Pour les faire converger plus rapidement, on peut par exemple ne faire la mise à jour des traces que pour la meilleure solution produite par les fourmis à l'itération courante ou même durant toute la recherche. Toutes les implantations récentes [DOR97, GAM97, STÜ97, TAI97b] améliorent la solution produite par chaque fourmi à l'aide d'une recherche locale gloutonne.

3. Programmation à mémoire adaptative

Bien que les méta-heuristiques présentées ci-dessus soient décrites de manière très différentes et qu'elles semblent basées sur des principes différents, elles sont en réalité très semblables, du moins si l'on considère les implantations les plus récentes et les plus efficaces. En effet, ces implantations fonctionnent toutes selon les mêmes principes. En d'autres termes, on observe une unification des diverses méta-heuristiques au point qu'il devient difficile de les distinguer les unes des autres. Par exemple, la population d'un algorithme génétique ou d'une recherche par dispersion ou encore les traces d'une colonie de fourmis peuvent être vues comme une mémoire spéciale d'une recherche avec tabous. Inversement, une recherche avec tabous qui construit périodiquement une nouvelle solution en utilisant une mémoire peut être assimilée à une colonie de fourmis ou à une recherche par dispersion. On observe donc que les meilleures méthodes générales de résolution de problèmes d'optimisation combinatoire présentent les mêmes caractéristiques :

1) Les solutions générées par la recherche (ou bien une structure de données spéciale agrégeant des particularités de ces solutions) sont mémorisées.
2) Une solution provisoire est construite en consultant la mémoire.
3) La solution provisoire est améliorée à l'aide d'une recherche locale gloutonne ou avec une autre méta-heuristique de base.
4) La nouvelle solution est utilisée pour mettre à jour la mémoire, pour adapter cette dernière aux nouvelles connaissances que la solution apporte.

Ainsi, il est très naturel de parler de programmation à mémoire adaptative pour qualifier ce type de méta-heuristique. Il est certainement plus logique et moins restrictif de parler de mémoire que de tabous pour une recherche avec tabous car la part des interdits est souvent petite par rapport aux autres composantes, pour des implantations relativement avancées. De même, il est aussi naturel de considérer les traces d'une colonie de fourmis ou les solutions de la population d'un algorithme génétique ou d'une recherche par dispersion comme une forme de mémoire qui s'adapte aux nouvelles connaissances acquises en cours de recherche.

Seule la recherche avec tabous, parmi les méthodes à mémoire considérées dans cette section, n'a pas explicitement été conçue initialement avec une construction de toutes pièces d'une nouvelle solution, alors que toutes les autres incluent cela comme principe de base. Cependant, bien des recherches avec tabous incluent un redémarrage du processus à partir d'une nouvelle solution, que ce soit dans un but de diversification, par exemple par la réalisation d'un certain nombre de modifications aléatoires comme dans la recherche avec tabous réactive de Battiti & Tecchiolli [BAT94], ou dans un but d'intensification, par exemple en repartant périodiquement des voisins des meilleures solutions trouvées par la recherche [NOW96].

Concernant l'incorporation d'une procédure de recherche locale dans le processus, il est clair que la recherche avec tabous est elle-même une recherche locale. La recherche par dispersion a été conçue avec une procédure spéciale de réparation de la solution provisoire, afin de ramener cette dernière dans l'ensemble des solutions admissibles. Dans sa mouture la plus récente [GLO97a], elle incorpore expressément une procédure d'amélioration ; elle est donc parfaitement calquée sur le modèle de programmation par mémoire adaptative proposé ici.

On pourrait objecter que les algorithmes génétiques et les méthodes basées sur les colonies de fourmis n'ont pas été proposés initialement avec une procédure de recherche locale. On observe toutefois un usage de plus en plus systématique d'une telle procédure dans les implantations récentes, car on s'est rendu compte que cela permettait d'améliorer notablement la qualité des solutions trouvées. En particulier, les algorithmes génétiques sont actuellement presque systématiquement hybridés avec une autre méta-heuristique, telle qu'un algorithme glouton d'optimisation, un recuit simulé ou une recherche avec tabous. De même, les dernières évolutions des colonies de fourmis sont aussi dotées d'une recherche locale. Ainsi, les trames des méthodes passées en revue dans la section précédente sont similaires et peuvent se formuler comme suit :

Programmation à mémoire adaptative

1) Initialiser la mémoire.
2) Répéter, tant qu'un critère d'arrêt n'est pas satisfait :
   - 2a) Générer une solution provisoire μ à partir des informations contenues dans la mémoire.
   - 2b) Améliorer μ à l'aide d'une recherche locale pour obtenir une solution π.
   - 2c) Mettre à jour la mémoire en incorporant les éléments d'information que contient π.

4. Programmation à mémoire adaptative parallèle

Bien que nous n'ayons pas formulé l'algorithme général de la programmation à mémoire adaptative sous une forme concurrente, il est cependant facile de le paralléliser. En effet, il suffit pour cela d'exécuter la boucle principale de manière indépendante par plusieurs processus qui ne communiquent entre eux que par l'intermédiaire de la mémoire. Cela signifie qu'une machine idéale pour l'exécution concurrente d'un programme à mémoire adaptative doit posséder une mémoire partagée. Le schéma d'une telle parallélisation est présenté en figure 1. Il est généralement très efficace car le processus de lecture de la mémoire et de sa mise à jour est beaucoup plus rapide que les autres, en particulier celui de la recherche locale qui consomme typiquement presque l'intégralité des ressources de calcul. On suppose cependant que chaque processus d'optimisation possède une copie locale des données du problème.

Nous pouvons illustrer une implantation typique d'un programme à mémoire adaptative parallèle sur le problème de l'affectation quadratique. La solution d'un tel problème, pour un exemple de taille n, peut être représentée par une permutation de n éléments. Les données du problème sont constituées de deux matrices carrées de dimension $n \times n$. Dans nos récentes implantations [GAM97, TAI97b], de même que dans l'implantation [FLE94], la mémoire adaptative est elle aussi constituée d'une matrice carrée de dimension $n \times n$ (ou éventuellement de $O(n)$ permutations de n

éléments, ce qui correspond à un encombrement similaire). Cela signifie que la quantité d'informations transférées pour consulter la mémoire ou la mettre à jour est au pire en $O(n^2)$, mais souvent en $O(n)$, le temps nécessaire pour calculer les nouvelles valeurs de la mémoire étant du même ordre de complexité.

Par contre, le processus de recherche locale est d'une complexité supérieure : $O(n^3)$. Ainsi, des problèmes de congestion dans les accès à la mémoire partagée pourront apparaître pour un nombre de processus d'optimisation de l'ordre de $O(n)$, voire $O(n^2)$.

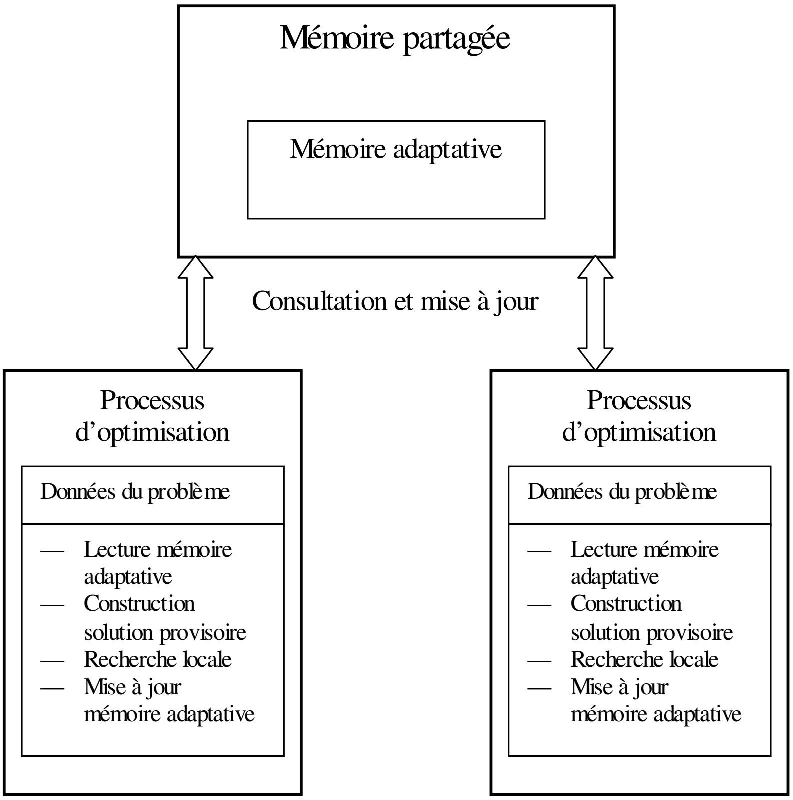

Figure 1. Parallélisation avec une mémoire commune.

Machines sans mémoire commune :

Les machines à mémoire commune ne sont cependant pas très courantes ; un exemple type de machine parallèle utilisée aujourd'hui est un réseau de stations de travail. Pour paralléliser un programme à mémoire adaptative sur un tel réseau, nous avons défini un processus chargé de simuler une mémoire partagée en plus des processus de recherche locale. Comme les opérations spécifiques à réaliser sur la mémoire ne sont pas gourmandes en temps de calcul, il est possible de les faire par le processus simulant la mémoire et on arrive au schéma de parallélisation présenté dans la figure 2.

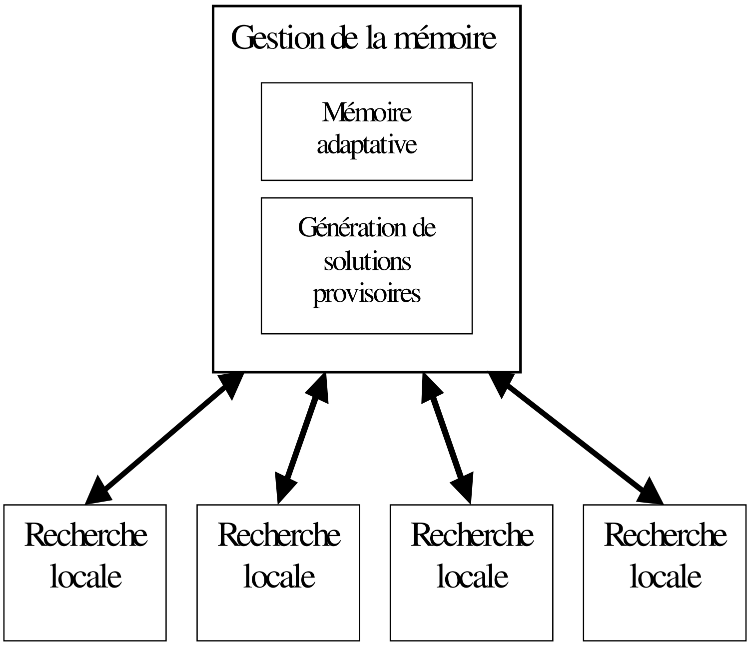

Figure 2. Parallélisation d'un programme à mémoire adaptative par processus communicants.

Il faut également noter que la recherche locale elle-même peut être concurrente. Dans ce cas, le processus de gestion de la mémoire peut aussi englober un processus d'équilibrage des charges, comme nous l'avons implanté dans Badeau et al. [BAD97]. De cette manière, nous avons pu observer des efficacités de parallélisation allant de 60 à 75% pour un nombre de processeurs compris entre 4 et 17 et pour une application à un problème de distribution de biens. Cette première mesure de l'efficacité est le rapport du temps de calcul sur une machine séquentielle et du temps sur la machine parallèle multiplié par le nombre de ses processeurs. On pourrait définir une seconde mesure, prenant en considération le temps nécessaire pour obtenir des solutions de qualité donnée sur les deux types de machine. En effet, les deux algorithmes, séquentiels ou parallèles ne fonctionnent pas de la même manière : l'algorithme séquentiel construit la solution provisoire sur la base de l'ensemble de toutes les solutions trouvées par le programme alors que le processus de génération de solutions provisoires de l'algorithme parallèle ignore les solutions en cours de traitement par les autres processus. L'algorithme séquentiel met donc à jour la mémoire progressivement, alors que l'algorithme parallèle fait une mise à jour par paquet. Nous avons cependant remarqué que pour un faible nombre de processeurs (jusqu'à 16), la qualité des solutions obtenues dépendait essentiellement de la quantité totale de travail réalisée, indépendamment du nombre de processeurs utilisés pour réaliser ce travail [BAD97]. D'un point de vue pratique, les deux mesures d'efficacité sont donc plus ou moins équivalentes.

5. Application de programmes à mémoire adaptative

Nous donnerons dans cette section des applications de la programmation à mémoire adaptative sur deux types de problèmes : l'affectation quadratique et l'élaboration de tournées de véhicules. Bien que la trame de l'algorithme général que nous avons donnée pour la programmation à mémoire adaptative soit très simple, on peut en réaliser des implantations très diverses, suivant les choix faits à chacune des étapes de l'algorithme.

5.1. Problème de l'affectation quadratique

Le problème de l'affectation quadratique (QAP) peut être formulé comme suit : étant donnés n unités et n sites où elles peuvent être placées, connaissant le flot f_{ij} circulant entre les unités i et j ($i, j = 1, \ldots, n$) et la distance d_{rs} entre les sites r et s ($r, s = 1, \ldots, n$), on cherche un placement des unités sur les sites minimisant la somme totale des produits flots $\times$ distances. Mathématiquement, on cherche une permutation $\boldsymbol{\pi}^*$ de $\Pi(n)$, l'ensemble des permutations de n éléments, minimisant :

$$\min_{\pi \in \Pi(n)} f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\pi_i \pi_j}$$

Ce problème a été posé pour la première fois par Koopmans & Beckman [KOO57] ; il est NP-difficile, même si l'on ne désire trouver qu'une solution approchée à ε près [SHA76]. Beaucoup de problèmes peuvent être modélisés comme un QAP, à commencer par le problème du voyageur de commerce, la conception de bâtiments [ELS77], l'équilibrage de turbines [LAP88], la génération de trames de gris [TAI95], etc. Il s'agit sans doute d'un des problèmes d'optimisation combinatoire les plus difficiles puisque les méthodes exactes actuelles résolvent avec peine des problèmes de taille $n = 20$. Cette difficulté n'est pas étrangère à notre ignorance de bonnes bornes inférieures.

Un grand nombre de méthodes ont donc été mises au point pour obtenir des solutions approchées au QAP. Parmi les plus efficaces, nous pouvons citer : la recherche avec tabous réactive (RTS) de Battiti & Tecchiolli [BAT94], la recherche par dispersion (SS) de Cung & al. [CUN97], les algorithmes génétiques hybrides de

Fleurent & Ferland (GTSH) [FLE95] et de Taillard & Gambardella (GDH) [TAI97b], plusieurs méthodes basées sur les colonies de fourmis dues à Stützle & Hoos (MMAS) [STÜ97], Gambardella, Taillard & Dorigo (HAS-QAP) [GAM97], Taillard & Gambardella (FANT) [TAI97b], ainsi que notre recherche avec tabous (TS) [TAI91]. Toutes ces méthodes utilisent une mémoire, sous une forme une autre. Les deux recherches avec tabous TS et RTS utilisent cette mémoire pour interdire de visiter certaines solutions, justifiant pleinement l'appellation de recherche avec tabous. Ces deux méthodes sont particulièrement efficaces pour les problèmes générés aléatoirement, uniformément. Par contre leurs performances sur des problèmes irréguliers sont médiocres [TAI94, GAM97, TAI97b]. Les autres méthodes citées ci-dessus utilisent une mémoire pour construire une solution provisoire, solution qui sera ensuite améliorée avec une recherche locale. Il s'agit donc de procédures dont le fonctionnement est calqué sur la trame de la programmation à mémoire adaptative de la section 3. Nous allons donc présenter plus en détail les choix qui ont été faits pour l'implantation de ces méthodes, c'est à dire en spécifiant : 1) le type de mémoire utilisée avec son processus d'adaptation, 2) la procédure de construction d'une solution provisoire et 3) la procédure de recherche locale.

1) Mémoires

Une solution d'un QAP pouvant être représentée par une permutation π de n éléments, où π_i indique l'emplacement de l'unité i, la plupart des méthodes à mémoire adaptative utilisent une matrice de taille $n \times n$ pour implanter la mémoire, l'entrée (i, j) de cette matrice étant une statistique mesurant l'intérêt de poser $\pi_i = j$. Deux méthodes, (GTSH et GDH) utilisent une population formée des meilleures solutions trouvées par la recherche, dans l'esprit des algorithmes génétiques. Finalement, deux méthodes (SS, HAS-QAP) utilisent à la fois une (petite) population de solutions et une matrice statistique sur les emplacements des unités. À chaque itération d'une procédure à mémoire adaptative, une solution π est produite (éventuellement p solutions $\pi^1, \ldots \pi^p$ sont produites simultanément). On considère de plus π^*, la meilleure solution trouvée par la procédure jusqu'à l'itération courante. La mise à jour de la mémoire se fait de la manière suivante :

SS, GTSH et GDH

π est insérée dans la population et la plus mauvaise solution de la population en est retirée. SS maintient de plus une matrice qui enregistre le nombre de fois que chaque unité a occupé un site donné.

FANT, HAS-QAP, MMAS

Ces trois méthodes ont été élaborées dans le cadre des colonies de fourmis et ont donc pour mémoire une matrice T de traces dont l'entrée τ_{ij} indique l'intérêt de poser

$\pi_i = j$. Cet intérêt se mesure par une statistique sur les solutions trouvées par la recherche, statistique qui varie selon la méthode.

Par exemple, HAS-QAP initialise toutes les entrées à la même valeur $\tau_{ij} = \tau_0$, $(1 \le i, j \le n)$, où τ_0 est un paramètre de la méthode. Après chaque itération, toutes les entrées de la matrice sont affaiblies d'un facteur α $(0 < \alpha < 1)$ en posant $\tau_{ij} = \alpha\tau_{ij}$, où α est un second paramètre représentant l'évaporation des phéromones. Finalement, les entrées $\tau_{i\pi_i^*}$, $(1 \le i \le n)$, associées à la meilleure des solutions trouvées par la recherche, π^*, sont augmentées d'une valeur $\tau_0(1-\alpha)/f(\pi^*)$. De plus, deux mécanismes utilisés exceptionnellement sont prévus : le premier pour éviter la stagnation de la recherche à cause de la convergence de la mémoire adaptative, le second pour accélérer cette convergence lorsque cela se justifie. Cette méthode utilise également une petite population de solutions comme mémoire adaptative. À chaque itération, certaines solutions de cette population sont changées. Par exemple, lorsqu'on suspecte une convergence de l'algorithme, toutes les solutions sauf la meilleure sont éliminées et remplacées par des solutions aléatoires. Inversement, lorsque l'on vient d'améliorer la meilleure des solutions (ce qui indique que le processus n'a pas convergé), on n'élimine de la population que les solutions qui sont plus mauvaises que celles nouvellement générées à l'itération courante.

L'idée à la base de MMAS est de limiter les valeurs possibles des entrées de la matrice $\boldsymbol{T}$ entre deux valeurs, t_{min} et t_{max}, afin de prévenir une convergence prématurée de la recherche. Initialement, toutes les entrées de la matrice sont mises à t_{max} ; elles sont affaiblies d'un facteur α à chaque itération et les entrées $\tau_{i\pi_i}$ sont renforcées d'une quantité $Q/\tau_{i\pi_i}$, où Q est un paramètre et $\boldsymbol{\pi}$ la meilleure solution produite à l'itération courante.

FANT initialise toutes les entrées de la matrice avec une valeur r. À chaque itération, les entrées $\tau_{i\pi_i}$ sont incrémentées de r et les entrées $\tau_{i\pi_i^*}$ sont incrémentées de R, où r et R sont deux paramètres. Nous avons également prévu un mécanisme pour éviter la convergence prématurée de la méthode : lorsque les entrées $\tau_{i\pi_i^*}$ sont si grandes que l'information contenue dans $\boldsymbol{T}$ ne diffère pas significativement de $\boldsymbol{\pi}^*$, la valeur de r est augmentée et $\boldsymbol{T}$ est réinitialisé ; lorsque $\boldsymbol{\pi}^*$ est améliorée, r reprend sa valeur initiale.

2) Solution provisoire

La procédure de construction d'une solution à partir de la mémoire est fortement dépendante du type d'informations qui y sont stockées. Les méthodes travaillant avec une population de solutions (GDH, GTSH et HAS-QAP) auront donc une procédure de génération très différente de celles construisant une solution à l'aide d'une matrice-statistique (SS, MMAS, FANT).

GDH et GTSH construisent une nouvelle solution en sélectionnant deux solutions de la population et en appliquant un opérateur de croisement spécialisé pour les permutations dû à Tate & Smith [TAT95].

FANT et MMAS construisent une solution provisoire μ en choisissant séquentiellement les entrées de μ aléatoirement avec une probabilité dépendant des entrées de la matrice T. MMAS répète cette construction p fois (où p est un paramètre de la méthode, représentant le nombre de fourmis) ; ces p solutions provisoires sont évaluées et seule la meilleure est retenue pour amélioration par la recherche locale.

HAS-QAP maintient une population de p solutions. Cette méthode se distingue des autres car elle ne construit pas une solution de toutes pièces à partir de la mémoire, mais elle modifie toutes les solutions de la population en faisant subir σ modifications à chacune de ces solutions, où σ est un paramètre de la méthode. Chacune de ces modifications consiste à échanger deux unités de site, le premier étant choisi aléatoirement, uniformément, et le second avec une probabilité qui dépend des entrées de la matrice-mémoire.

3) Procédure d'amélioration

FANT, HAS-QAP et GDH utilisent toutes la même procédure d'amélioration qui consiste à évaluer chaque paire d'échange de deux unités et à effectuer immédiatement les échanges qui améliorent la solution. Cette procédure est répétée si une amélioration a été trouvée ; elle est très rapide, quoiqu'en $O(n^3)$.

GTSH utilise notre méthode TS comme procédure d'amélioration, chaque solution provisoire étant améliorée par $4n$ itérations de TS. La complexité de cette procédure est également $O(n^3)$, mais elle est approximativement 8 fois plus lente que la procédure utilisée dans FANT, HAS-QAP et GHD.

Il existe deux versions de MMAS, la première utilisant $4n$ itérations de TS comme procédure d'amélioration et la seconde effectuant une descente vers le premier optimum local relativement à l'échange de deux unités. SS utilise une recherche avec tabous comme procédure d'amélioration.

5.2. Efficacité des méthodes à mémoire adaptative pour le QAP

Dans [GAM97] on trouvera une comparaison de HAS-QAP avec d'autres méthodes : TS, RTS, un recuit simulé, GTSH. Dans [TAI97b], on confronte FANT et GDH à HAS-QAP, TS et GTSH. De plus, [STÜ97] compare deux versions de MMAS à HAS-QAP et GTSH. Finalement, nous comparons, entre autres, TS, RTS et GTSH dans [TAI95]. Il est donc possible d'avoir une bonne idée des performances relatives de toutes ces méthodes et tous les auteurs s'accordent sur un certain nombre de points :

— Lorsque les données présentent une structure marquée ou sont très irrégulières, par exemple lorsque les unités forment des groupes ou lorsque la matrice de flots a des coefficients très variables, avec un grand nombre d'entrées nulles, la programmation à mémoire adaptative réussit à identifier la structure des bonnes solutions alors que les recherches avec tabous, ou le recuit simulé, ont beaucoup plus de peine à se diriger vers de bonnes solutions, restant souvent piégés dans des optima locaux de piètre qualité.

— Inversement, lorsque les problèmes sont peu structurés, par exemple ceux générés aléatoirement, uniformément, les recherches avec tabous sont parmi les méthodes les plus performantes alors que les autres programmes à mémoire adaptative ont de la peine à identifier la structure d'une solution optimale, ce qui n'est pas étonnant si l'on sait que les optima locaux de tels problèmes sont répartis assez uniformément dans l'espace des solutions, comme nous avons pu l'établir en étudiant leur entropie [TAI95].

Les comparaisons directes des diverses méthodes à mémoire adaptative FANT, HAS-QAP, GDH et MMAS (variante « rapide » avec méthode de descente) révèlent des performances très voisines, bien que leurs implantations soient assez différentes. Il semble donc que la conception d'une méthode à mémoire adaptative soit relativement aisée à réaliser pour le QAP et que le type de mémoire utilisé et la procédure de construction d'une solution provisoire ne soient pas très sensibles aux options choisies, pour autant que l'on prenne soin de prévoir, d'une part, un mécanisme de diversification au cas où la méthode convergerait prématurément ; d'autre part, il faut également s'assurer que les informations contenues dans la mémoire convergent et permettent de construire des solutions avec une bonne structure générale.

Pour les problèmes irréguliers, il convient de mentionner que les programmes à mémoire adaptative dont nous venons de discuter sont les meilleures méthodes à l'heure actuelle pour le problème d'affectation quadratique [GAM97, TAI97b, STÜ97, CUN97].

5.3 Problème d'élaboration de tournées

Il existe un grand nombre de variantes de problèmes d'élaboration de tournées de véhicules. Dans une de ses formes les plus simples, on a un ensemble de n clients qui demandent un bien en quantité q_i ($i = 1, \ldots, n$). Pour satisfaire ces demandes, on dispose d'un véhicule de capacité Q qui doit se réapprovisionner à partir d'un dépôt unique. Connaissant les distances entre chaque paire de clients et entre le dépôt et les clients, on cherche des tournées de longueur totale minimale telles que la quantité à livrer dans chaque tournée soit au plus égale à Q. À partir de ce modèle de base, que l'on nommera A, il est possible d'ajouter des contraintes ou des ressources pour arriver à des modèles plus compliqués mais plus proche de la réalité :

- B la durée de chaque tournée est limitée, et on a un temps de service pour chaque client.
- C les livraisons s'organisent en journées de travail d'une durée limitée L et toutes les commandes doivent avoir été livrées dans un laps de temps de K jours ; il est donc possible de faire plusieurs tournées par jour ; afin d'assurer l'existence d'une solution admissible, il est permis de dépasser la limite de temps L moyennant une pénalité proportionnelle à la durée du dépassement.

D on dispose d'une flotte hétérogène de véhicules ; chacun étant spécifié par sa capacité, son coût fixe d'engagement et un coût variable d'utilisation dépendant de la longueur du trajet effectué.

E le nombre de tournées est limité et on désire minimiser la longueur de la plus longue tournée.

F les livraisons s'organisent en K journées de travail et on désire minimiser la durée de la plus longue journée de travail.

G les clients spécifient une fenêtre de temps durant laquelle ils peuvent recevoir leur livraison. On désire engager un nombre minimum de véhicules pour réaliser l'ensemble des livraisons.

Pour tous ces problèmes, nous avons mis au point des procédures à mémoire adaptative qui utilisent toutes la même forme de mémoire et le même mode de construction des solutions provisoires, aussi qualifié de construction de vocabulaire dans [GLO97b].

La mémoire est constituée de l'ensemble de toutes les tournées individuelles contenues dans les solutions trouvées par la recherche. La procédure de construction d'une solution provisoire consiste à choisir une tournée aléatoirement dans la mémoire telle qu'aucun de ses clients ne soient compris dans les tournées précédemment sélectionnées. La probabilité de choisir une tournée donnée dépend de la qualité de la solution dans laquelle on retrouve cette tournée. Ainsi, la mémoire peut contenir plusieurs fois la même tournée avec plusieurs évaluations différentes. La procédure de construction d'une solution provisoire ne produit pas une solution admissible à chaque appel : il est en effet très probable que des clients ne soient pas touchés par les tournées choisies mais que la mémoire ne contienne pas de tournée constituée uniquement d'un sous-ensemble des clients non touchés. Il est par conséquent très important de concevoir une procédure de recherche locale qui puisse rendre admissibles de telles solutions. Pour les variantes de problèmes A à F, nous avons utilisé comme méthode une recherche avec tabous de base décrite dans [TAI93].

Pour la variante G, nous avons recouru à d'autres procédures d'optimisation, aussi basées sur une recherche avec tabous, mais capables de prendre en considération les fenêtres de temps. La première de ces procédures est la simplification d'une méthode mise au point pour une application industrielle tenant compte d'autres contraintes comme des fenêtres de temps multiples, les pauses des chauffeurs et l'inaccessibilité de certains clients par certains véhicules [ROC94]. La deuxième procédure prend en considération des fenêtres de temps souples, c'est à dire qu'il est possible d'arriver en retard chez un client moyennant une pénalité. Cette procédure est basée sur une recherche avec tabous utilisant un voisinage original : l'échange de sous-tournées. Ce voisinage en contient plusieurs autres communément utilisés pour les problèmes d'élaboration de tournées, comme l'insertion d'un client dans une tournée, l'échange de deux clients appartenant à des tournées différentes, la concaténation de deux tournées, etc.

L'algorithme général de résolution de ces problèmes de distribution peut être esquissé comme suit :

Programme à mémoire adaptative pour problème d'élaboration de tournées

1) $\boldsymbol{M} = \emptyset$.
2) Répéter, tant qu'un critère d'arrêt n'est pas satisfait :
   - 2a) $\boldsymbol{M'} = \boldsymbol{M}$, $\boldsymbol{s} = \emptyset$.
   - 2b) Répéter, tant que $\boldsymbol{M'} \neq \emptyset$:
     - 2b1) Choisir une tournée $T \in \boldsymbol{M'}$, aléatoirement.
     - 2b2) Poser $\boldsymbol{s'} = \boldsymbol{s'} \cup \{T\}$.
     - 2b3) Pour toute tournée $T' \in \boldsymbol{M'}$ telle que $T \cap T' \neq \emptyset$, poser $\boldsymbol{M'} = \boldsymbol{M'} \setminus \{T'\}$.
   - 2c) Compléter la solution partielle $\boldsymbol{s'}$ et l'améliorer à l'aide d'une recherche locale pour obtenir une solution $\boldsymbol{s}$.
   - 2d) Pour toute tournée T de $\boldsymbol{s}$, poser : $\boldsymbol{M} = \boldsymbol{M} \cup \{T\}$.
3) Trouver une solution $\boldsymbol{s^*}$ aussi bonne que possible en considérant les tournées contenues dans $\boldsymbol{M}$.

Ce programme appelle deux remarques : premièrement, dans le cas de problèmes avec flotte hétérogène, on doit considérer une mémoire différente pour chaque type de véhicules. En effet, nous n'avons implanté que des recherches locales pour des problèmes homogènes, donc nous répétons l'étape 2 pour chaque type de véhicule, comme si l'on avait une flotte homogène. Secondement, l'étape 3 consiste à résoudre un problème d'optimisation NP-difficile : pour les problèmes de type A, B, D et G, il s'agit de problèmes de partitionnement d'ensemble ; pour les problèmes de type C, E et F, le problème de partitionnement d'ensemble est couplé à un problème de mise en boîte. Il existe actuellement des méthodes exactes assez efficaces pour résoudre le problème de partitionnement d'ensemble ; même un logiciel général de programmation linéaire en nombres entiers permet d'en résoudre des exemples de taille acceptable (quelques centaines de clients et plusieurs centaines de tournées contenues dans la mémoire). Pour les types de problèmes requérant une mise en boîte, nous avons recouru à la méthode suivante [TAI96b, GOL97] : on commence par résoudre le problème de partitionnement en énumérant l'ensemble de toutes les solutions d'un problème de type A que l'on peut construire à l'aide des tournées contenues dans la mémoire ; ensuite, dans le cas de la variante E, on met une tournée par boîte pour chaque solution énumérée et on retient la meilleure solution admissible ainsi trouvée ; dans le cas des variantes C et F, on cherche pour chacune de ces solutions une mise en boîte des tournées (i.e. un regroupement par journée de travail) à l'aide d'une heuristique simple (insertion gloutonne suivie éventuellement d'une descente dans le premier optimum local relativement à l'échange de jours de deux tournées).

À l'aide de ce type de programme à mémoire adaptative, nous avons pu mettre au point des méthodes très efficaces pour tous les types de problèmes d'élaboration de tournées mentionnés ci-dessus. Tout d'abord, nous montrons dans [ROC95] que l'encapsulation d'une recherche avec tabous à l'intérieur de ce schéma de programme à mémoire adaptative permet d'obtenir une méthode sensiblement plus robuste que la recherche avec tabous initiale. Les meilleures solutions connues d'un nombre conséquent d'exemples de problèmes de type A, B et F de la littérature ont ainsi pu être améliorées ou obtenues.

Dans [TAI96a], nous proposons une méthode pour les problèmes avec flotte hétérogène. Nous avons réussi à obtenir ou à améliorer toutes les meilleures solutions connues de problèmes de la littérature à l'aide de cette technique. Dans [GOL97], nous décrivons plus en détail les méthodes conçues pour les variantes de type E et F. Dans [TAI96b], nous montrons que les solutions trouvées par la programmation à mémoire adaptative sont souvent très proches d'une borne inférieure, ou éventuellement que les dépassements de la durée limite L des problèmes de type C sont faibles.

Dans [TAI97a], nous évaluons les performances d'un nouveau type de voisinage pour les problèmes de distribution avec application à une variante du type G où les fenêtre de temps sont souples. Ce nouveau voisinage est utilisé dans une recherche avec tabous servant de procédure d'amélioration d'un programme à mémoire adaptative. Nous avons ensuite parallélisé cette méthode dans [BAD97], (voir ci-dessus pour les techniques de parallélisation) avant de l'adapter pour la résolution de problèmes dynamiques dans [GEN96a, GEN96b].

Un des avantages de la programmation à mémoire adaptative dont nous n'avons pas encore parlé est son adéquation au traitement de problèmes dynamiques. En effet, les modifications des données du problème peuvent être intégrées rapidement et les informations contenues dans la mémoire restent pertinentes si le problème n'a pas été trop modifié et que la structure des bonnes solutions ne change pas significativement. Bien que la recherche locale soit la partie la plus gourmande en temps de calcul, un appel ne prend pas beaucoup de temps en valeur absolue. Par conséquent, il est possible d'intégrer en temps réel les nouvelles données du problème, par exemple un nouveau client pour l'élaboration de tournées de véhicules. Dans [GEN96a, GEN96b], nous avons montré que la programmation à mémoire adaptative permettait d'obtenir de meilleures solutions que des heuristiques simples comme la meilleure insertion possible dans la solution courante ou la reconstruction complète d'une solution avec le nouveau client, suivies d'une procédure de recherche locale : la qualité des tournées obtenues par programmation adaptative, que ce soit en utilisant une méthode de descente ou une recherche avec tabous comme procédure de recherche locale, est nettement meilleure, tant du point de vue de la longueur des tournées, du nombre de clients qui ont pu être insérés avec succès que des éventuels retards dans les visites. De plus, le fait que la programmation à mémoire adaptative soit facilement parallélisable renforce encore ses performances sur les problèmes dynamiques, ainsi que nous l'avons également montré dans les articles cités.

6. Conclusions

Nous avons montré dans ce travail que les méta-heuristiques à mémoire se sont rapprochées les unes des autres ces dernières années, au point de se rencontrer parfois. La dénomination de programmation à mémoire adaptative nous semble à la fois plus précise et plus générale car elle est mieux représentative du fonctionnement des dernières évolutions des méta-heuristiques et laisse cependant une grande liberté dans le choix de l'implantation. Nous pouvons dire que la programmation à mémoire adaptative semble une des techniques les plus prometteuses à l'heure actuelle pour la résolution de problèmes d'optimisation combinatoire, et cela pour plusieurs raisons.

- Elle permet d'obtenir des solutions de qualité élevée.
- Elle reste conceptuellement simple et donc relativement facilement implantable.
- Elle est facilement parallélisable.
- Elle est efficace et particulièrement bien adaptée aux besoins des applications pratiques car on peut l'implanter de telle sorte qu'il soit possible d'obtenir n'importe quand durant la recherche non pas une seule, mais plusieurs bonnes solutions simultanément, ce qui est un atout non négligeable pour les applications réelles.
- Elle est bien adaptée à la résolution de problèmes dynamiques.

Ces deux derniers points mériteront une attention particulière dans un futur relativement proche. En effet, si les méta-heuristiques n'ont encore fait que de timides apparitions dans les logiciels commerciaux, il n'en reste pas moins qu'elles y sont de plus en plus présentes, l'augmentation de la puissance des micro-ordinateurs n'y étant certainement pas étrangère. Avec la perspective de pouvoir les utiliser pour des applications dynamiques, sur un réseau de calculateurs, on peut penser que la programmation à mémoire adaptative a de beaux jours devant elle.

Bibliographie

[BAD97] P. BADEAU, M. GENDREAU, F. GUERTIN, J.-Y. POTVIN & É. D. TAILLARD, « A parallel tabu search heuristic for the vehicle routing problem with time windows », *Transportation Research-C 5*, 1997, pp. 109-122.

[BAT94] R. BATTITI & G. TECCHIOLLI, « The reactive tabu search », *ORSA Journal on Computing 6*, 1994, pp. 126-140.

[COL91] A. COLORNI, M. DORIGO & V. MANIEZZO, « Distributed Optimization by Ant Colonies », actes de *ECAL91 — European Conference on Artificial Life*, 1991, pp. 134-142.

[CUN97] V.-D. CUNG, T. MAUTOR, PH. MICHELON & A. TAVARES, « A Scatter Search Based Approach for the Quadratic Assingment Problem », Rapport technique, Université de Versailles-Saint Quentin en Yvelines, 1997.

[DOR97] M. DORIGO & L. M. GAMBARDELLA, « Ant colony system : A cooperative Learning Approach to the Traveling Salesman Problem », *IEEE Transactions on Evolutionary Computing 1*, 1997, pp. 53-66.

[DOR96] M. DORIGO, V. MANIEZZO & A. COLORNI, « The Ant System : Optimization by a colony of cooperating agents », *IEEE Transactions on Systems, Man, an Cybernetics-Part B 26*, 1996, pp. 29-41.

[ELS77] A. E. ELSHAFEI, « Hospital Layout as a Quadratic Assignment Problem », *Operations Research Quaterly 28*, 1977, pp. 167-179.

[FAI92] U. FAIGLE & W. KERN, « Some convergence results for probabilistic tabu search », *ORSA Journal on Computing 4*, 1992, pp. 32-37.

[FLE94] C. FLEURENT & J. FERLAND, « Genetic hybrids for the quadratic assignment problem », *DIMACS Series in Mathematics and Theoretical Computer Science 16*, 1994, pp. 190-206.

[FLE96] C. FLEURENT, F. GLOVER, P. MICHELON & Z. VALLI, « A Scatter Search Approach for Unconstrained Continuous Optmization », actes de *IEEE International Conference on Evolutionary Computation*, 1996, pp. 643-648.

[GAM97] L. M. GAMBARDELLA, É. D. TAILLARD & M. DORIGO, « Ant colonies for the quadratic assignment problem », rapport technique *IDSIA-4-97*, IDSIA, Lugano, 1997.

[GEN96a] M. GENDREAU, P. BADEAU, F. GUERTIN, J.-Y. POTVIN & É. D. TAILLARD, « A solution procedure for real-time routing and dispatching of commercial vehicles », actes du *3. World Congress on Intelligent Transport Systems*, Orlando (Floride), 1996.

[GEN96b] M. GENDREAU, F. GUERTIN, J.-Y. POTVIN & É. D. TAILLARD, « Tabu search for real-time vehicle routing and dispatching », rapport technique *CRT-96-47*, Centre de recherche sur les transports, Université de Montréal, 1996.

[GLO77] F. GLOVER, « Heuristics for Integer Programming Using Surrogate Constraints », *Decision Sciences 8*, 1977, pp. 156-166.

[GLO86] F. GLOVER, « Future Paths for Integer Programming and Links to Artificial Intelligence », *Computers and Operations Research 13*, 1986, pp. 156-166.

[GLO89] F. GLOVER, « Tabu Search — Part I », *ORSA Journal on Computing 1*, 1989, pp. 190-206.

[GLO90] F. GLOVER, « Tabu Search — Part II », *ORSA Journal on Computing 2*, 1990, pp. 4-32.

[GLO96] F. GLOVER « Tabu Search and adaptive memory programming — advances, applications and challenges », à paraître dans : *Interfaces in Computer Science and Operations Research*, Barr, Helgason and Kennington éditeurs, Kluwer Academic Publishers, 1996.

[GLO97a] F. GLOVER « A template for Scatter Search and Path Relinking » rapport technique, Université du Colorado à Boulder, 1997.

[GLO97b] F. GLOVER & M. LAGUNA, *Tabu Search*, Kluwer Academic Publishers, 1997.

[GOL97] B. L. GOLDEN, G. LAPORTE, É. D. TAILLARD, « An adaptive memory heuristic for a class of vehicle routing problems with minmax objective » , *Computers and OR 24*, 1997, pp. 445-452.

[HAJ88] B. HAJEK, « Cooling schedules for optimal annealing » *Mathematics of Operations Research 13*, 1988, pp. 311-329.

[HOL75] J. H. HOLLAND, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press : Ann Arbor, 1975.

[KEL96] J. KELLY, B. RANGASWAMY & J. XU, « A Scatter Search-Based Learning Algorithm for Neural Network Training », *Journal of Heuristics 2*, 1996, pp. 129-146.

[KOO57] T. C. KOOPMANS & M. J. BECKMANN, « Assignment problems and the location of economics activities », *Econometrica 25*, 1957, pp. 53-76.

[LAP88] G. LAPORTE & H. MERCURE, « Balancing Hydraulic Turbine Runners : A Quadratic Assignment Problem », *European Journal of Operational Research 35*, 1988, pp. 378-381.

[MÜL88] H. MÜHLENBEIN, M. GORGES-SCHLEUTER & O. KRÄMER, « Evolution Algorithms in Combinatorial Optimization », *Parallel Computing 7*, 1988, pp. 65-88.

[NOW96] E. NOWICKI & C. SMUTNICKI, « A fast taboo search algorithm for the job shop problem » *Management Science 42*, 1996, pp.797-813.

[ROC94] Y. ROCHAT & F. SEMET, « A Tabu Search Approach for Delivering Pet Food an Flour in Switzerland », *Journal of the Operations Research Society 45*, 1994, pp. 1233-1246.

[ROC95] Y. ROCHAT & É. D. TAILLARD, « Probabilistic diversification and intensification in local search for vehicle routing ». *Journal of Heuristics 1*, 1995, pp. 147-167.

[SHA76] S. SHANI & T. GONZALEZ, « P-complete approximation problems », *Journal of the ACM 23*, 1976, pp. 555-565.

[STÜ97] T. STÜTZLE & H. HOOS, « MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems — Towards Adaptive Tools for Combinatorial Global Optimization ». conférence *MIC 97*, Sophia-Antipolis, juillet 1997.

[TAI91] É. D. TAILLARD, « Robust taboo search for the quadratic assignment problem », *Parallel computing 17*, 1991, pp. 443-455.

[TAI93] É. D. TAILLARD, « Parallel iterative search methods for vehicle routing problems », *Networks 23*, 1993, pp. 661-673.

[TAI94] É. D. TAILLARD, « Parallel taboo search techniques for the job shop scheduling problem », *ORSA journal on Computing 6*, 1994, pp. 108-117.

[TAI95] É. D. TAILLARD, « Comparison of iterative searches for the quadratic assignment problem », *Location Science 3*, 1995, pp. 87-105.

[TAI96a] É. D. TAILLARD, « A heuristic column generation method for the heterogeneous VRP », rapport technique *CRT-96-03*, Centre de recherche sur les transports, Université de Montréal, 1996, à paraître dans *RAIRO-OR*

[TAI97a] É. D. TAILLARD, P. BADEAU, M. GENDREAU, F. GUERTIN & J.Y. POTVIN, « A tabu search heuristic for the vehicle routing problem with soft time windows », *Transportation Science 31*, 1997, pp. 170-186.

[TAI97b] É. D. TAILLARD & L. M. GAMBARDELLA, « Adaptive Memories for the Quadratic Assignment Problem », rapport technique *IDSIA-87-97*, IDSIA, Lugano, 1997.

[TAI96b] É. D. TAILLARD, G. LAPORTE & M. GENDREAU, « Vehicle routeing with multiple use of vehicles », *Journal of the Operations Research Society 47*, 1996, pp. 1065-1070.

[TAT95] D. E. TATE & A. E. SMITH, « A genetic approach to the quadratic assignment problem », *Computers and Operations Research 1*, 1995, pp. 855-865.