Chapitre 2

Principes d'implémentation des métaheuristiques

Éric D. Taillard[1]

2.1 Introduction

Les métaheuristiques ont changé radicalement l'élaboration d'heuristiques : alors que l'on commençait par s'interroger sur les caractéristiques et les particularités du problème à résoudre avant de commencer à programmer une méthode spécifique, les métaheuristiques ont en quelque sorte inversé le processus, la trame de la méthode de résolution étant fournie par la métaphore qui a inspiré la métaheuristique. Ayant une première heuristique, on cherche ensuite à l'améliorer en observant les faiblesses qu'elle présente pour le problème à résoudre.

Il s'agit là d'une conception que l'on peut qualifier de partisane, liée à l'école d'une métaheuristique particulière. Jusque vers le milieu des années 90, une question à laquelle bien des personnes ont cherché à répondre était de savoir si telle ou telle métaheuristique était meilleure ou moins bonne que telle ou telle autre. La réponse n'a bien évidemment jamais été trouvée, pour la simple raison que l'efficacité des méthodes mises au point dépend avant tout des adaptations de la trame de base aux spécificités du problème (et même des exemples de problèmes que l'on cherche à

[1] Institut d'Informatique, École d'Ingénieurs du Canton de Vaud, Haute École Spécialisée de Suisse Occidentale, Route de Cheseaux 1, 1400 Yverdon-Les-Bains, Suisse, eric.taillard@eivd.ch, http://www.eivd.ch/ina/Collaborateurs/etd/default.htm.

résoudre pratiquement). Ainsi, plus la trame est riche et complexe, comme par exemple pour la recherche avec tabous, plus on a de possibilités d'amélioration d'une méthode de base, mais, par contre, plus il est difficile de sélectionner et d'assembler les bons principes de la métaheuristique.

Par la suite, la situation s'est notablement compliquée avec l'avènement de méthodes dites *hybrides*. Au lieu de faire preuve de sectarisme et de limiter les principes de construction d'une heuristique à une seule école — génétique, recuit simulé, etc. —, on s'est rendu compte que la mise en commun des principes de plusieurs métaheuristiques voire de méthodes exactes pouvait mener à l'élaboration d'heuristiques encore plus performantes.

Cependant, la mise au point d'une méthode basée sur diverses métaheuristiques devient difficile, car il faut tout d'abord sélectionner un sous-ensemble de principes qui devront être appropriés au problème à résoudre. On peut donc dire que bien des heuristiques actuellement développées sont un assemblage plus ou moins judicieux d'un nombre relativement restreint de principes de base, parfois déguisés pour pouvoir être exprimés sous la forme d'une métaheuristique « pure ». De plus, la situation se complique encore par l'apparition de « nouvelles » métaheuristiques, qui ne sont souvent que le résultat d'un assemblage — particulièrement bien adapté à un problème donné — de quelques principes de base.

La présentation « d'application des métaheuristiques » en général ne peut donc se faire qu'en décrivant comment ces quelques principes de base ont été appliqués dans certains problèmes d'optimisation. L'assemblage de ces principes pour obtenir une heuristique spécifique pour un problème donné, heuristique inspirée d'une ou de plusieurs métaheuristiques, fera l'objet de chapitres ultérieurs. Mentionnons encore que ce chapitre n'aura pas pour objet un passage en revue des problèmes pour lesquels il existe une méthode efficace basée sur une métaheuristique, cet ensemble de problème étant bien trop vaste.

Nous terminerons ce chapitre par quelques suggestions en ce qui concerne la comparaison des heuristiques itératives non déterministes. En effet, les métaheuristiques mènent souvent à ce type d'heuristiques, qui sont encore beaucoup trop rarement comparées de manière valable dans la littérature.

2.2 Voisinage

Sans conteste, le principe général le plus largement utilisé dans l'élaboration d'heuristiques est celui de voisinage. À chaque solution $\boldsymbol{s}$ du problème, on associe un sous-ensemble $V(\boldsymbol{s})$ de solutions. Ce sous-ensemble peut être statique, comme dans le cas du recuit simulé ou des méthodes de bruitage, mais aussi dynamique et dépendre

du temps *t* (ou plus précisément de l'itération à laquelle on se trouve). Notons que la dépendance du voisinage par rapport au temps est étroitement couplée à d'autres principes de base contenus dans la métaheuristique, comme par exemple la liste de tabous ou de candidats dans une recherche avec tabous.

Même dans le cas d'une structure de voisinage statique et très simple, où le seul but est de trouver un optimum local, on ne connaît que très peu de résultats théoriques utiles. L'algorithme du simplexe pour la programmation linéaire est un exemple typique. Pour ce problème, il est bien connu que le nombre d'itérations pour arriver à une solution optimale peut être exponentiel en la taille du problème. Pourtant, en pratique, on observe que le nombre d'itérations croît bien plus lentement, ce qui en fait un algorithme efficace et très largement utilisé. L'algorithme du simplexe est basé sur un voisinage statique : pour toute solution $\boldsymbol{s}$, $V(\boldsymbol{s})$ est constitué des solutions qui diffèrent de $\boldsymbol{s}$ par l'échange de deux variables, l'une sortant de la base et l'autre y entrant (à cela s'ajoute bien évidemment d'autres considérations, comme l'admissibilité de la solution voisine ou le fait que celle-ci ne doit pas être plus mauvaise que la solution de départ).

Pour un problème d'optimisation combinatoire non convexe, pour lequel il est possible de définir un ensemble de voisinages a priori intéressants, la situation se complexifie. Il devient alors difficile de se décider pour l'un ou l'autre des voisinages autrement que par des essais. Comme le voisinage n'est qu'une partie des principes, tous interdépendants, utilisés dans l'heuristique, le choix d'un (ou de plusieurs) voisinages devient réellement problématique car on ne dispose que de très peu de résultats théoriques sur la qualité d'un voisinage pour un problème particulier. Angel (1998) a cependant proposé et analysé une mesure de l'adéquation des voisinages appelée *rugosité* : l'idée consiste à évaluer la variance des coûts de deux solutions voisines. Cette variance est ensuite normalisée par la variance des coûts de l'ensemble des solutions et par la taille du problème, afin d'avoir une mesure indépendante de la valeur absolue de la fonction-objectif ou de la taille du problème. La rugosité donne une indication sur la difficulté à résoudre un problème à l'aide d'une recherche locale utilisant un voisinage donné. En effet, si cette mesure est élevée, on peut supposer que le voisinage est peu adapté car les valeurs des solutions voisines semblent peu corrélées, alors que si cette mesure est faible, on aura un « paysage » lisse, avec relativement peu d'optima locaux, donc un problème facile à optimiser à l'aide d'une recherche locale. Naturellement, si la rugosité est une notion mathématiquement bien définie, l'interprétation donnée ci-dessus n'est qu'une supposition plausible, étayée seulement par des considérations empiriques (Angel et Zissimopoulos, 1998).

2.2.1. *Voisinage sur une permutation*

Illustrons tout d'abord comment des voisinages simples peuvent être définis pour des problèmes où une solution est une permutation. De nombreux problèmes d'optimisation combinatoire peuvent s'exprimer sous la forme d'une recherche de permutations : celui du voyageur de commerce (ordre dans lequel on parcourt les villes, voir Lawler et al. 1985) celui de l'affectation quadratique (site de chaque unité à placer, voir chapitre 3, tome 2 du présent ouvrage) ou encore celui de la chaîne de montage (ordre dans lequel on fait passer les travaux sur la chaîne, voir B•azevicz et al. 1994).

A priori, tous ces problèmes pourraient utiliser le même voisinage puisque leurs solutions se représentent sous la même forme. L'inversion de deux éléments contigus est un des voisinage les plus simples pour une permutation. La taille de ce voisinage est en $O(n)$ pour un problème à n objets (villes, sites, tâches) et c'est certainement un des plus petits que l'on puisse imaginer fonctionner en pratique (il possède la propriété de connexité pour ces problèmes). On observe cependant que ce voisinage n'est pas approprié pour les trois problèmes mentionnés ci-dessus. En regardant d'un peu plus près l'action de ce voisinage, on s'aperçoit qu'il est trop restreint car il est nécessaire d'appliquer plusieurs mouvements pour modifier la solution de manière significative, par exemple pour supprimer un croisement dans une solution de voyageur de commerce symétrique.

Le but du voisinage connu sous le nom de 2-opt (Croes, 1958) consiste précisément à essayer de supprimer des croisements ou des zigzags pour le problème du voyageur de commerce. Si l'on nomme π_i la $i^{\text{ième}}$ ville que l'on visite dans la solution courante ($i = 0, 1, \ldots, n-1$), le voisinage 2-opt commence par choisir deux villes u et v ($v \neq u,\ u+1$). La solution voisine θ est obtenue à partir de π en supprimant les deux arcs $(u, u+1)$ et $(v, v+1)$, en ajoutant les arcs (u, v) et $(u+1, v+1)$ et en inversant le sens de parcours du chemin entre $u+1$ et v, c'est-à-dire en posant, avec les indices calculés modulo n :

$$\theta_{u+1} = \pi_v,\ \theta_{u+2} = \pi_{v-1},\ \ldots,\ \theta_v = \pi_{u+1},\ \theta_{v+1} = \pi_{v+1},\ \ldots,\ \theta_u = \pi_u$$

Ce voisinage est de taille $O(n^2)$. Il est très largement utilisé en pratique pour le problème du voyageur de commerce. Si l'on essayait d'aborder l'affectation quadratique à l'aide de ce voisinage, on serait bien déçu car il n'est pas adapté à la structure de ce problème (à moins que l'on cherche à résoudre un exemple très particulier de problème d'affectation quadratique qui correspond précisément à un voyageur de commerce). En effet, l'inversion de la partie de la permutation située entre $u+1$ et v implique une modification énorme de la solution pour ce problème, alors qu'il s'agissait d'une opération mineure pour le voyageur de commerce

symétrique. Nous verrons plus loin que la transposition (échange de deux éléments π_u et π_v, voir chapitre 3, tome 2 du présent ouvrage) est presque le seul voisinage utilisé dans les recherches locales pour l'affectation quadratique, tout d'abord parce qu'il ne modifie pas trop la solution, ensuite parce qu'il est rapide à évaluer et enfin parce que sa taille est raisonnable.

Si l'on essayait d'utiliser les transpositions pour définir un voisinage pour le problème de la chaîne de montage, comme cela a été fait dans Widmer et Hertz (1989), on serait à nouveau déçu car il est également mal adapté à ce problème (Taillard 1990, Reeves 1999) pour deux raisons : premièrement son évaluation complète est plus complexe et secondement sa structure n'est pas adaptée au problème. Dans ce cas, on a intérêt à déplacer une tâche dans la séquence. Étant donnés deux indices u et v, $u \neq v$ on définit θ, la permutation voisine de π en déplaçant la tâche en $v^{\text{ième}}$ position avant (respectivement : après) la tâche en $u^{\text{ième}}$ position si v est plus grand (respectivement : plus petit) que u :

$$\text{si } u < v : \theta_k = \pi_k \ (1 \leq k < u \text{ et } v < k \leq n),\ \theta_u = \pi_v,\ \theta_k = \pi_{k-1} \ (u + 1 \leq k \leq v)$$

$$\text{si } u > v : \theta_k = \pi_k \ (1 \leq k < v \text{ et } u < k \leq n),\ \theta_k = \pi_{k+1} \ (v \leq k < u),\ \theta_u = \pi_v.$$

Il est montré dans Taillard (1990) que ce voisinage peut être évalué en entier en $O(mn^2)$, ce qui représente un gain appréciable par rapport au voisinage utilisé dans Widmer et Hertz (1989) qui ne peut être évalué complètement qu'en $O(mn^3)$. De nouveau, ce voisinage n'est adapté ni pour le problème de l'affectation quadratique, ni pour le problème du voyageur de commerce (il s'agit d'un très petit sous-ensemble d'un voisinage couramment utilisé pour ce problème proposé par Bock (1958) et connu sous le nom de 3-opt).

2.2.2. *Chaînes d'éjections*

À l'aide de voisinages simples, il est possible de définir des voisinages plus compliqués en composant plusieurs modifications. La technique des chaînes d'éjections est basée sur ce principe. Cette possibilité est en particulier intéressante pour des problèmes très contraints pour lesquels il n'est pas forcément évident de définir une structure de voisinage telle que toute solution ait au moins un voisin. Une première forme de chaîne d'éjections a été proposée par Berge (1958). Par la suite, la technique a été généralisée et a été utilisée avec succès par divers auteurs (voir en particulier Dorndorf et Pesch 1994, Pesch et Glover 1995, Rego et Roucairol 1996 ainsi que Glover et Laguna 1997). Prenons l'exemple du problème de l'élaboration de tournées de véhicules sous contrainte de capacité qui sera présenté plus en détail dans le chapitre 2 du tome 2 «Applications» du présent ouvrage. Une des formes les

plus simples de ce problème consiste en 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 être réapprovisionné à partir d'un dépôt unique. Connaissant d'une part la distance entre chaque paire de clients et, d'autre part la distance entre le dépôt et chaque client, on cherche des tournées de longueur totale minimale telles que la quantité à livrer dans chaque tournée soit au plus égale à Q.

Un des voisinages les plus simples pour ce problème consiste à transférer un client d'une tournée à une autre. Évidemment, si pour une solution donnée chaque tournée comporte une quantité de biens à livrer plus grande que $Q - q_{\min}$ (où $q_{\min}$ est la plus petite quantité demandée par un client), il n'existera pas de solution voisine admissible. La technique des chaînes d'éjection permet de contourner le problème comme ceci : la tournée recevant un nouveau client en éjecte un autre, de manière à ce qu'elle reste admissible. Le client qui a été éjecté doit naturellement être intégré dans une autre tournée, au besoin en éjectant un troisième client. Ainsi, à chaque étape, on se trouve dans l'une des deux situations suivantes : Soit le véhicule n'a pas une capacité suffisante pour intégrer le nouveau client. Dans ce cas, un autre client doit être à son tour éjecté de la tournée et le processus se poursuit (à moins que la chaîne des clients éjectés-intégrés soit plus longue qu'un paramètre h). Soit le véhicule a une capacité suffisante pour intégrer le client. Dans ce cas, on a une solution admissible que l'on peut garder en mémoire avant d'éjecter un autre client si la chaîne des clients éjectés-intégrés est plus courte que h.

Naturellement, la chaîne d'éjection peut être initialisée à partir de n'importe lequel des n clients. De plus, on pourra essayer plusieurs clients lors des éjections ultérieures. On pourra donc répéter le processus pour un certain nombre de clients initiaux (éventuellement tous) et après chaque insertion d'un client dans une tournée, on pourra également essayer l'éjection d'un certain nombre de clients hors de cette même tournée. Ceci permettra généralement de mémoriser de nombreuses solutions admissibles qui formeront une nouvelle structure de voisinage.

Si chaque insertion donne lieu à plusieurs éjections, il est clair que la taille du voisinage ainsi engendré croît exponentiellement avec la valeur du paramètre h. Selon le problème et les propriétés des opérations élémentaires d'éjection et d'insertion, il est parfois possible de trouver la meilleure solution voisine (ou une bonne solution voisine) avec un effort de calcul polynomial en h et n, typiquement en résolvant un problème de plus court chemin, d'affectation linéaire ou de flot à coût minimum (voir par exemple Xu et Kelly 1996). Le paragraphe suivant présentera une autre technique permettant de travailler avec des voisinages potentiellement très grands.

2.2.3. *Optimisations partielles*

Un autre type de modifications d'une solution consiste à en isoler une partie et à optimiser cette dernière plus ou moins indépendamment du reste de la solution. L'ensemble des solutions qu'il est possible d'obtenir en ne modifiant que la partie qui a été isolée définit une structure de voisinage un peu particulière qui fera l'objet de cette section. Récemment, Taillard et Voß (1999) ont donné aux méthodes faisant appel à ce type de structure le nom de *POPMUSIC*, abréviation de métaheuristique par optimisation partielle sous certaines conditions. Cette dénomination regroupe un ensemble de techniques présentant de fortes ressemblances mais qui ont reçu au cours du temps divers noms tels que *Shuffle* (Applegate et Cook 1991), *LOPT* (abréviation d'optimisations locales, Taillard 1996), *MIMAUSA* (Mautor et Michelon, 1997), *LNS* (abréviation de recherche dans un grand voisinage, Shaw 1998), méthode hybride de recherche avec tabous-branchement & saut (Büdenbender et al. 1999) ou encore VNDS (abréviation de recherche par décomposition et voisinage variable, Hansen et Mladenovic 1999).

Les techniques du type POPMUSIC s'appliquent essentiellement aux exemples de problèmes de grande taille dont les solutions peuvent être optimisées partiellement (d'où les « conditions spéciales » contenues dans la dénomination de la technique). L'idée consiste à optimiser des parties d'une solution une fois que cette dernière est disponible. Par exemple, dans la procédure Shuffle, appliquée à un problème d'ordonnancement, seules les opérations devant avoir lieu sur un sous-ensemble de machines peuvent être réordonnées. Une partie de la solution est donc dans ce cas un sous-ensemble d'opérations.

Supposons donc qu'une solution $\boldsymbol{s}$ d'un problème puisse être représentée comme un ensemble de p parties $s_1, \ldots, s_p$. De plus, on supposera qu'il est possible de définir et de quantifier une relation existant entre chaque paire de parties. L'idée centrale de POPMUSIC est de sélectionner une partie-racine s_i et les $r < p$ parties $s_{i_1}, \ldots, s_{i_r}$ qui sont le plus étroitement couplées à s_i pour former une solution d'un sous-problème R_i (r est un paramètre de la méthode). Ce sous-problème est ensuite optimisé à l'aide d'une méthode appropriée. Pour que la technique puisse s'appliquer à un problème donné, on fait encore l'hypothèse que l'on peut facilement trouver une solution meilleure que la solution de départ $\boldsymbol{s}$ si l'on a réussi à améliorer la qualité de R_i.

Illustrons cela sur le problème d'élaboration de tournées présenté en section 2.2.2. Une solution $\boldsymbol{s}$ de ce problème est composée de p tournées (en général, la valeur de p dépend de $\boldsymbol{s}$, à moins que p soit contenu dans les données du problème). On peut donc définir une partie de $\boldsymbol{s}$ comme l'ensemble des clients desservis par une tournée donnée de $\boldsymbol{s}$. Ensuite, il est possible de quantifier une

relation existant entre deux parties quelconques s_i et s_j, par exemple en prenant la distance minimale, moyenne ou maximale qui existe entre deux clients de s_i et s_j. En considérant les clients des r parties les plus étroitement couplées avec une partie donnée s_i, on peut définir un sous-problème R_i qui est également un problème d'élaboration de tournées, mais de taille plus petite que le problème initial. Il est clair que si l'on arrive à améliorer la qualité de la solution de R_i définie par les r tournées sélectionnées, on peut améliorer la qualité de $\boldsymbol{s}$ en remplaçant ces r tournées par la solution améliorée de R_i que l'on a trouvée. Ainsi, POPMUSIC peut être décrit brièvement comme suit :

1. Données : Une solution $\boldsymbol{s}$ composée de parties $s_1, \ldots, s_p$.
2. Poser $O = \varnothing$.
3. Tant que $O \neq \{s_1, \ldots, s_p\}$ répéter :
   a. Sélectionner $s_i \notin O$
   b. Créer un sous-problème R_i composé des r parties $s_{i_1}, \ldots, s_{i_r}$ les plus étroitement reliées à s_i.
   c. Optimiser R_i.
   d. Si la solution de R_i a été améliorée, mettre $\boldsymbol{s}$ à jour (ainsi que ses parties) et poser $O \leftarrow \varnothing$; sinon poser $O \leftarrow O \cup \{s_i\}$

À la base, POPMUSIC est donc une méthode de gradient qui part d'une solution initiale et qui s'arrête dans un optimum local relativement à la structure d'un grand voisinage. En effet, cette structure de voisinage contient toutes les solutions $\boldsymbol{s'}$ qui diffèrent de $\boldsymbol{s}$ par le sous-problème R_i, $i = 1, \ldots, p$. Ainsi, la taille du voisinage est égale au nombre de solutions des sous-problèmes. Naturellement ce nombre peut être très grand et croît exponentiellement avec le paramètre r (pour $r = p$, $R_i = \boldsymbol{s}$ et la technique ne fait qu'optimiser $\boldsymbol{s}$ directement). Cela signifie que l'optimisation à l'étape 3c se fait non pas par énumération complète mais par une technique d'énumération implicite (comme suggéré dans Shuffle, MIMAUSA ou LNS) ou de manière heuristique (par exemple avec une recherche avec tabous comme dans Taillard (1993) ou avec une méthode à voisinage variable comme dans VNDS).

Dans l'algorithme esquissé ci-dessus, le mécanisme proposé pour éviter d'essayer d'optimiser un sous-problème que l'on aurait déjà essayé sans succès d'améliorer, est de maintenir un ensemble O de parties-racines qui ont déjà été utilisées pour définir un sous-problème. Une fois que toutes les parties sont contenues dans O, la solution est localement optimale et le processus s'arrête. Au cas où un sous-problème R_i a été amélioré avec succès, un certain nombre de parties de $s_{i_1}, \ldots, s_{i_r}$ ont été modifiées et il convient de réinitialiser l'ensemble O avant de poursuivre le processus. Très schématiquement, les composantes-clé de POPMUSIC qui doivent être définies par l'implanteur sont :

1. La définition des *parties* d'une solution.

2. Une procédure de *sélection* d'une partie n'appartenant pas à *O*.
3. Une fonction donnant un *degré de relation* entre deux parties.
4. Une procédure d'*optimisation.*

Mentionnons encore que certains problèmes permettent la définition d'un ensemble de sous-problèmes complètement indépendants. Il est alors possible d'optimiser ces sous-problèmes en parallèle si l'on dispose d'une machine distribuée. Le problème de l'élaboration de tournées de véhicules se prête particulièrement bien à l'optimisation parallèle dès que le nombre de tournées d'une solution est suffisamment grand, ainsi que l'a montré Taillard (1993).

2.3 Mémoire

On peut dire que toutes les métaheuristiques font appel à une structure de voisinage ou à une mémoire, et généralement aux deux à la fois. En effet, pour simplifier et en ne considérant que les versions les plus primitives de ces métaheuristiques, il n'y a que les algorithmes génétiques et les colonies de fourmis qui n'utilisent pas de structure de voisinage. Dans leurs versions plus élaborées, ces techniques se dotent cependant d'une procédure de recherche locale, et donc d'une structure de voisinage. De manière analogue, seuls le recuit simulé, les algorithmes de bruitage et d'acceptation à seuils ne reposent que sur un générateur pseudo-aléatoire et ne font pas appel à une forme de mémoire pour s'échapper des optima locaux relativement à la structure de voisinage choisie.

Le fait que les métaheuristiques s'appuient actuellement sur deux pans, formés par les structures de voisinage et les mémoires, a suggéré à certains auteurs de présenter de manière unifiée un vaste ensemble de métaheuristiques sous la dénomination de *programmation à mémoire adaptative* (Taillard 1998). La première partie de ce chapitre présente diverses structures de voisinage. Nous allons maintenant passer en revue quelques techniques d'implantation de mémoires.

2.3.1. *Population*

Un ensemble de solutions, ou population de solutions pour reprendre le vocabulaire de certains algorithmes évolutifs (Bremermann et al. 1966, Rechenberg 1973, Holland 1975, Glover 1977) est une des plus anciennes formes de mémoire utilisée dans les métaheuristiques. En effet, on peut considérer qu'une population de solutions forme une mémoire traduisant l'évolution du processus au cours du temps. Les principales différences qu'il y a entre les algorithmes génétiques, la recherche par dispersion («scatter search», Glover 1977, Glover et Laguna 1997) ou les évolutions de ces algorithmes résident essentiellement dans la gestion de la

population et de son utilisation pour construire de nouvelles solutions. Nous reviendrons plus en détail sur ce dernier point ultérieurement. Examinons tout d'abord comment une population de solutions peut être gérée.

Il n'existe pas de version unique des algorithmes génétiques et par conséquent plusieurs manières ont été suggérées pour la mise à jour de la population. Une des idées possibles est de produire un certain nombre de nouvelles solutions à l'aide d'une population donnée. Ces nouvelles solutions formeront la population à la génération suivante, ce qui signifie que l'ancienne population est complètement remplacée. Plutôt que d'effacer intégralement l'ancienne population, il est également possible de ne remplacer qu'une partie de celle-ci. L'autre extrême consiste à remplacer une solution chaque fois qu'une autre a été produite, pour autant que la nouvelle solution soit de qualité suffisante. On rencontre plusieurs politiques de remplacement dans la littérature telles que :

- Élimination des solutions les plus anciennes.
- Élimination aléatoire de solutions.
- Élimination des solutions les plus mauvaises.

Ces politiques influent en particulier sur la rapidité de convergence des algorithmes évolutifs. On observe en effet que le remplacement des plus mauvaises solutions entraîne une convergence plus rapide que les autres politiques pour une population de taille donnée. Il faut en effet noter que ces algorithmes convergent vers une population d'individus identiques. On peut corriger une convergence trop rapide en augmentant la taille de la population. On pourra consulter Mühlenbein (1997) pour obtenir quelques informations théoriques sur la convergence d'algorithmes génétiques.

Dans la recherche par dispersion, Glover (1977) propose d'éviter la convergence de la population à l'aide d'une politique plus élaborée d'élimination de solutions. On conservera dans la population non seulement les meilleures solutions, mais également des solutions aux caractéristiques différentes. Pour arriver à ce but, on peut définir un facteur de similarité entre les solutions de la population étendue (ancienne population à laquelle on ajoute les nouvelles solutions produites). On applique ensuite une méthode de classification automatique basée sur le facteur de similarité, ce qui permet d'obtenir un nombre donné de groupes à l'intérieur desquels on choisira au moins une solution pour former la prochaine population.

La technique connue sous le nom de *construction de vocabulaire* (Glover et Laguna, 1997) est une forme de mémoire très proche d'une population de solutions. À la place de mémoriser des solutions entières, il est possible de stocker des parties de solutions. Si l'on prend l'exemple de l'élaboration de tournées de véhicules, appliquée avec un certain succès par Rochat et Taillard (1995), cette technique

consiste à mémoriser des tournées réalisables de façon indépendante les unes des autres. Ces tournées seront par la suite utilisées pour construire de nouvelles solutions.

2.3.2. *Statistique*

Une autre forme de mémoire très utilisée est l'enregistrement de statistiques sur les solutions visitées, sur des caractéristiques de ces dernières ou encore sur les modifications apportées aux solutions dans le contexte d'une recherche locale. La recherche avec tabous de même que les colonies de fourmis artificielles sont des métaheuristiques qui sont principalement basées sur des informations statistiques.

L'idée à la base de la recherche avec tabous est de décrire le voisinage sous la forme de modifications (ou mouvements) que l'on peut apporter à une solution. Le voisinage sera limité en mémorisant l'itération à laquelle on a utilisé une certaine modification. Si le numéro d'itération est trop élevé pour un mouvement donné, la solution voisine correspondante sera taboue, c'est-à-dire éliminée du voisinage. Dans ce cas, la statistique mémorisée est très simple, il s'agit d'une liste de numéros d'itération associés à chaque modification possible. Ce mécanisme d'interdiction est parfois appelé mémoire à court terme, car les mouvements sont en principe interdits pour un nombre restreint d'itérations.

Pour les problèmes sur des permutations, ce type de mémoire peut être implantée sous la forme d'une matrice $\boldsymbol{T} = (t_{ij})$ de dimension $n \times n$. Pour l'affectation quadratique, l'élément t_{ij} indique à quelle itération on a déplacé à l'emplacement j l'élément qui était à l'emplacement i. L'échange des éléments occupant les emplacements i et j est interdit si le maximum entre t_{ij} et t_{ji} est plus grand qu'une certaine valeur (voir par exemple Taillard, 1991).

Pour le problème de l'ordonnancement de tâches sur une chaîne de production, t_{ij} doit prendre une signification différente car la position absolue d'une tâche à l'intérieur de la chaîne a moins d'importance que la tâche qui la précède ou celle qui lui succède. Dans ce cas, t_{ij} indique l'itération à laquelle on a déplacé la tâche i avant (ou après) la tâche j.

À côté de ce mécanisme d'interdiction, très simple et efficace, on trouve une mémoire à long terme, généralement implantée à l'aide d'une autre statistique. Un exemple typique consiste à enregistrer la fréquence d'utilisation de chaque mouvement possible. Une des premières utilisations conjointes de ces deux statistiques a été proposée par Taillard (1994). Dans cette application, on utilise la seconde statistique pour bruiter la fonction-objectif de manière à déformer le relief du voisinage. On réalise ce bruitage en dégradant simplement la qualité d'une

solution voisine proportionnellement à la fréquence d'utilisation du mouvement que l'on doit utiliser pour l'atteindre. Ceci permet de diminuer le nombre de solutions voisines interdites par la mémoire à court terme (ce qui permet d'améliorer la qualité moyenne des solutions visitées) tout en évitant l'examen cyclique d'un sous-ensemble des solutions admissibles.

Les colonies de fourmis artificielles (cf. chapitre 5 du présent ouvrage) sont basées sur l'utilisation intensive de statistiques associées à chaque composante d'une solution. Le premier problème abordé par cette métaheuristique a été celui du voyageur de commerce (Colorni et al. 1992). Dans ce cas, une statistique est calculée pour chaque arc. Elle est mise à jour lors de la construction de nouvelles solutions sur la base de l'utilisation ou non de cet arc dans la solution. Nous passerons ici sur les alchimies proposées par divers auteurs pour le calcul de la statistique associée à chaque arc, nous aurons l'occasion d'y revenir plus loin. Disons simplement qu'elle dépend de la qualité de la solution nouvellement créée, de la meilleure solution générée par la recherche jusque là et de la qualité intrinsèque de la composante (pour le voyageur de commerce ou l'élaboration de tournées, il s'agit de la longueur de l'arc). Des implantations plus récentes incluent encore d'autres informations.

2.4 Construction de solutions

La reconstruction de nouvelles solutions à l'aide d'une mémoire fait partie de nombreuses métaheuristiques. Les algorithmes génétiques ou les colonies de fourmis artificielles sont basées essentiellement sur le principe de la construction périodique de solutions entièrement nouvelles, contrairement au recuit simulé ou des algorithmes de bruitage qui s'appuient seulement sur une structure de voisinage pour modifier légèrement une solution existante. Le cas de la recherche avec tabous est intermédiaire. Dans sa forme la plus élémentaire, elle aussi n'a qu'une structure de voisinage, modulée en fonction des solutions précédemment énumérées. Cependant, certains principes plus évolués de la recherche avec tabous font également appel à la construction de nouvelles solutions, en particulier la technique de construction de vocabulaire.

2.4.1. *Construction avec une population de solutions*

L'opérateur de croisement des algorithmes génétiques est une des techniques les plus anciennes de construction d'une nouvelle solution avec une mémoire du type population. Parmi les opérateurs les plus fréquemment utilisés, citons ceux du croisement à un ou plusieurs points et celui du croisement uniforme qui s'appliquent à des solutions pouvant s'exprimer comme des vecteurs binaires quelconques (mais dont le nombre de composantes, n, est fixé). Le croisement à un point permet

d'obtenir une nouvelle solution ***w*** (appelée aussi solution-enfant) à partir de deux autres, ***u*** et ***v*** (les parents). Il consiste à générer un nombre aléatoire j compris entre 2 et n et à poser $w_i = u_i$ ($i = 1, \ldots, j-1$) et $w_i = v_i$ ($i = j, \ldots, n$). En quelques mots, les premières composantes de l'enfant ***w*** sont identiques au premier parent, ***u***, et les dernières au second, ***v***. Parfois, on génère simultanément une solution ***w'***, obtenue en permutant les rôles des vecteurs ***u*** et ***v***.

Le croisement à deux points consiste à générer deux nombres aléatoires $1 < j < k \leq n$ et à poser $w_i = u_i$ ($i = 1, \ldots, j-1, i = k, \ldots n$) et $w_i = v_i$ ($i = j, \ldots, k-1$). Le principe peut être généralisé avec plusieurs points en alternant la copie des éléments de ***u*** et ***v*** dans ***w***. Finalement le croisement uniforme copie dans ***w*** les éléments identiques dans ***u*** et ***v*** et génère aléatoirement les autres.

Naturellement, les opérateurs de croisement décrits ci-dessus ne s'appliquent que lorsque les solutions peuvent être représentées par des vecteurs binaires quelconques, ce qui représente un handicap majeur pour certains problèmes, comme celui de l'affectation quadratique où une solution est une permutation et où il est par conséquent difficile de définir de manière naturelle une bijection (un codage) entre un sous-ensemble de vecteurs binaires et les permutations. Si les solutions de la population ne sont pas mémorisées sous la forme de vecteurs binaires quelconques, il est nécessaire de définir des opérateurs de croisement spécialisés. Dans le cas des permutations de n éléments, Tate et Smith (1995) ont proposé un algorithme présentant des similitudes avec l'opérateur uniforme : les éléments identiques des deux permutations parentes sont copiés dans la solution-enfant. Ensuite, cette dernière est partiellement complétée en copiant aléatoirement les éléments de l'un ou l'autre des parents, pour autant que ceux-ci ne fassent pas déjà partie de la solution-enfant. Finalement elle est complétée de manière aléatoire avec les éléments manquants.

Outre une gestion plus intelligente de la population, la recherche par dispersion suggère l'utilisation d'opérateurs de croisement plus élaborés, pouvant s'appliquer à plus de deux solutions qui sont représentées de manière naturelle. Ces opérateurs produisent une solution-enfant non nécessairement réalisable, mais qui sera rendue admissible par une procédure de réparation avant d'être améliorée avec une recherche locale.

Une fois ces principes généraux émis, il reste à les appliquer au problème que l'on cherche à résoudre, ce qui n'est pas forcément évident. Prenons l'exemple de la programmation linéaire en nombres entiers. Une solution pourra être naturellement représentée par un vecteur de nombres entiers. Une solution-enfant pourra être générée en faisant une combinaison linéaire de plusieurs solutions-parents. La procédure de réparation reste cependant dépendante du type de programme linéaire que l'on cherche à résoudre et il n'est malheureusement pas possible de décrire ici une procédure générale et efficace.

2.4.2. *Construction avec informations statistiques*

Les colonies de fourmis artificielles utilisent les informations statistiques liées à des composantes de solutions de façon très directe au travers d'une procédure constructive. Cette dernière génère une nouvelle solution de manière probabiliste. Plus la valeur de la statistique associée à une composante est élevée, plus la probabilité de choisir cette composante sera grande dans le processus de construction. Le fonctionnement d'une heuristique basée sur une colonie de fourmis artificielles est donc fortement dépendant des statistiques liées aux composantes des solutions et du fait que les bonnes solutions partagent des composantes communes.

Dans le cas du voyageur de commerce, on calcule une statistique s_{ij} pour chaque arc (i, j) (Colorni et al. 1992). La procédure constructive s'exprime ainsi :

1) Choisir un sommet π_1 aléatoirement, uniformément.
2) Pour $i = 1, \ldots, n - 1$, répéter :
   a. Choisir $j \notin \{\pi_1, \ldots, \pi_i\}$ aléatoirement, avec une probabilité dépendant de $s_{\pi_i j}$.
   b. Poser $\pi_{i+1} = j$.

Dans le cas de l'ordonnancement, la procédure est identique, à l'exception du choix de π_1 qui se fait non uniformément car, pour ce problème, les tâches aux extrémités de la séquence ont plus d'influence sur la fonction-objectif que celles placées au milieu.

Pour l'affectation quadratique, la statistique s_{ij} dépend principalement de l'occupation du site j par l'objet i (Stützle et Hoos 1999, Taillard 2000). La construction n'a pas lieu d'être ordonnée (étape 2). La procédure aura donc plutôt la forme :

1) Répéter n fois :
   a. Choisir i aléatoirement, uniformément parmi les objets non encore placés.
   b. Choisir j aléatoirement, parmi les sites non encore choisis, avec une probabilité dépendant de s_{ij}.
   c. Poser $\pi_i = j$.

On utilise souvent des informations statistiques conjointement à une population de solutions pour construire de nouvelles solutions. Beaucoup d'algorithmes génétiques sélectionnent les deux solutions-parents avec une probabilité dépendant d'une statistique associée à chaque solution, comme par exemple le rang de cette dernière dans la population (une fois triée selon la qualité des solutions).

2.5 Stratégies

Tout l'art du concepteur d'une heuristique résidera dans l'assemblage judicieux des différents principes énumérés plus haut, le tout épicé d'une bonne intuition, sans pour autant négliger les quelques éléments de théorie connus sur les métaheuristiques, en algorithmique ou dans le domaine des méthodes dites exactes. Naturellement, une connaissance des caractéristiques du problème à résoudre reste un atout indispensable pour la conception d'une méthode efficace.

Un principe très largement utilisé dans les métaheuristiques (et pas seulement pour la recherche avec tabous) est connu sous le nom d'oscillations stratégiques (Glover et Laguna 1997). Ce principe stipule qu'il faut modifier périodiquement les paramètres d'une recherche itérative pour être mieux à même de l'intensifier, c'est-à-dire d'examiner en détail un sous-ensemble très restreint des solutions, et de la diversifier en changeant fondamentalement la structure des solutions. Dans une recherche évoluée, ces phases seront alternées. Un exemple typique de technique d'intensification a été donné plus haut avec POPMUSIC. Une autre possibilité consiste à retourner en arrière et à redémarrer la recherche à partir d'une bonne solution trouvée précédemment, ou à partir d'une de ses voisines. Une façon très simple d'implanter une diversification dans une recherche avec tabous est d'effectuer un certain nombre de mouvements aléatoires. On trouve par exemple ce mécanisme dans la recherche avec tabous réactive qui sera exposée dans le chapitre 3 consacré à l'affectation quadratique dans le tome 2 « Applications » du présent ouvrage.

Un nombre important d'heuristiques incorporent des oscillations stratégiques avec le schéma de programmation à mémoire adaptative suivant (voir par exemple Taillard 1998) :

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

Dans les algorithmes génétiques ou la recherche par dispersion, la mémoire principale est constituée par la population de solutions. L'initialisation de cette mémoire consiste souvent à générer des solution aléatoires, éventuellement améliorées à l'aide d'une recherche locale. En recherche avec tabous, cette mémoire peut prendre plusieurs formes (fréquence d'utilisation de mouvements, parties de solutions qui ont de « bonnes » propriétés, nombre de solutions énumérées avec contrainte active, etc.) et a pour but d'activer un mécanisme de diversification. Pour initialiser cette mémoire, il peut donc être nécessaire d'effectuer un certain nombre

d'itérations « normales » durant lesquelles des informations statistiques sont mémorisées.

Concernant l'étape 2a, nous avons vu plus haut comment générer de nouvelles solutions à l'aide d'une mémoire. Pour l'étape 2b, le terme de recherche locale est utilisé dans un sens très large. Il peut s'agir simplement de se déplacer vers un optimum local relativement à une structure de voisinage, ou d'une recherche avec tabous élémentaire (avec mécanismes d'intensification activés), ou d'un recuit simulé ou encore de POPMUSIC, etc. Notons que les algorithmes génétiques basés sur ce schéma de programmation à mémoire adaptative sont souvent qualifiés d'hybrides. Moscato (1999) propose un schéma très semblable sous le nom d'« algorithmes mémétiques ».

2.6 Comparaison d'heuristiques itératives non déterministes

Les métaheuristiques posent un problème nouveau et pas encore résolu de manière satisfaisante : celui de la comparaison des heuristiques qu'elles engendrent. En effet, ces heuristiques sont itératives, ce qui veut dire que plus longtemps on les laisse travailler, meilleures sont les solutions qu'elles produisent. Elles sont également presque toujours non déterministes car elles utilisent un générateur pseudo-aléatoire. Cela signifie que si on les exécute plusieurs fois, on observera des solutions différentes. Quelques rares recherches avec tabous sont déterministes (toutes les autres métaheuristiques sont basées sur des choix probabilistes), mais il serait cependant facile de les rendre non déterministes car elles font des choix arbitraires, tels que l'ordre dans lequel les solutions voisines sont évaluées, la solution voisine choisie en cas d'égalité d'évaluation, etc.

Il est clair que la comparaison d'heuristiques itératives doit se faire sur la qualité des solutions produites en fonction de l'effort de calcul consenti. Ce dernier est traditionnellement un temps d'exécution sur une machine donnée. Malheureusement, cette mesure est à la fois imprécise et volatile. En effet, le temps de calcul est fortement dépendant du style de programmation, du compilateur et de ses options, de la charge de calcul au moment où les tests sont réalisés, etc. De plus, la durée de vie des équipements informatiques est fortement limitée. Ils sont parfois déjà obsolètes lorsque les articles décrivant de nouvelles heuristiques sont publiés !

Dans le cas où le problème combinatoire consiste à optimiser une fonction unique, il est facile de comparer la qualité de deux solutions (ce qui n'est plus forcément vrai en optimisation multicritères : voir le chapitre 7 du tome 2 du présent ouvrage). Cependant, la comparaison d'heuristiques non déterministes n'est plus si immédiate, puisque la qualité des solutions produites ne dépend pas seulement de l'effort de calcul mais également des paramètres d'initialisation d'une fonction

pseudo-aléatoire. Comment comparer deux méthodes heuristiques dans ces conditions ? On peut répondre à cette question par des tests statistiques.

Supposons que l'on ait exécuté l'heuristique ***A*** (respectivement : l'heuristique ***B*** avec laquelle on cherche à la comparer) n_A fois (respectivement : n_B fois), chaque exécution se faisant en consentant un effort de calcul important. Durant ces exécutions, on enregistre à chaque amélioration de la solution la qualité de cette dernière ainsi que le temps de calcul. De cette manière, pour un temps t donné, il sera possible de retrouver la qualité des $n_A + n_B$ solutions que les deux heuristiques ont été capables de fournir pour cet effort de calcul. La question intermédiaire, et plus simple, à laquelle il faut tenter de répondre est de savoir si l'heuristique ***A*** est meilleure que l'heuristique ***B*** après un temps de calcul t. Une première remarque s'impose : on peut considérer que la qualité des solutions obtenues par l'heuristique ***A*** (resp. ***B***) à l'instant t obéit à une variable aléatoire $X_A(t)$ (resp. $X_B(t)$) de fonction de densité $f_A(t)$ (resp. $f_B(t)$). On voudrait comparer les espérances mathématiques $\boldsymbol{E}(X_A(t))$ et $\boldsymbol{E}(X_B(t))$ pour savoir laquelle est la plus grande et donc décider laquelle des deux heuristiques est la meilleure pour le temps t. Malheureusement, les fonctions de répartition sont inconnues et, puisqu'on a généralement des nombres d'exécutions n_A et n_B restreints (typiquement d'une à quelques dizaines), il est illusoire d'essayer de déterminer f_A et f_B avec les données à disposition. Faire l'hypothèse que ce sont des courbes de Gauss est abusif (on sait que ces fonctions sont bornées par la valeur de l'optimum). La comparaison des méthodes sur la base des moyennes et écarts-types des qualités de solutions observées est donc pour le moins douteuse.

Comme il n'existe aucun test statistique sur les moyennes de deux populations arbitraires, il est nécessaire de se tourner vers d'autres hypothèses. Le test de Mann-Whitney permet une comparaison intéressante : en effet, traduite au cas de la comparaison d'heuristiques non déterministes, l'hypothèse nulle de ce test stipule que $f_A = f_B$. Si la probabilité que cette hypothèse soit vraie est plus petite qu'un seuil α que l'on se donne, en regard des qualités de solutions observées sur les n_A et n_B exécutions, on la rejettera au profit d'une hypothèse alternative : la probabilité que ***A*** produise une solution meilleure que ***B*** est plus grande (ou plus petite) que la probabilité que ***B*** produise une solution meilleure que ***A***. Mentionnons encore que si l'on sait que les distributions f_A et f_B sont identiques à une translation près, le test sera également valable pour l'hypothèse nulle plus forte : $\boldsymbol{E}(X_A) = \boldsymbol{E}(X_B)$.

Très brièvement, le test de Mann-Whitney s'effectue comme suit : tout d'abord, on classe les $n_A + n_B$ solutions obtenues par qualité décroissante et on leur donne un rang de 1 à $n_A + n_B$. Si plusieurs solutions ont la même qualité, on leur donne à chacune la moyenne des rangs qu'elles auraient obtenus s'il n'y avait pas eu égalité. Ensuite, on calcule une statistique S, correspondant à la somme des rangs des solutions issues de l'heuristique ***A***. Si $S > T_\alpha(n_A, n_B)$, on rejette l'hypothèse nulle avec

un seuil de confiance α et on accepte donc que ***A*** est moins bonne que ***B***. Les valeurs de $T_\alpha(n_A, n_B)$ peuvent être trouvées dans des tables numériques ; certaines tables donnent α en fonction de S et $n_A + n_B$. Pour plus de détails sur ce test, on pourra consulter l'ouvrage de Conover (1999), par exemple.

Naturellement, il faudra répéter ce test pour différentes valeurs de t. Ceci nous permettra de comparer commodément deux heuristiques par représentation graphique de la probabilité que ***A*** soit meilleure que ***B*** en fonction de l'effort de calcul. Évidemment, une telle comparaison suppose que l'on dispose des exécutables des deux heuristiques afin d'obtenir des temps de calcul fiables. Néanmoins, il se pourrait qu'une des heuristiques puisse être programmée de manière à ce qu'elle s'exécute plus rapidement au détriment de l'autre. Il convient donc d'être particulièrement prudent lorsqu'on compare des temps de calcul, par exemple en considérant un facteur multiplicatif de sécurité dans la mesure de ces temps.

Finalement, comme mentionné plus haut, le temps de calcul est certes un critère essentiel pour la comparaison d'heuristiques, mais ce critère est volatile. Afin de s'affranchir des caractéristiques sur lesquelles on a choisi d'exécuter les méthodes itératives et pour mesurer l'effort de calcul de manière absolue, on pourra mesurer ce dernier en nombre d'itérations. Si de plus on prend soin d'indiquer pour chaque méthode la complexité des calculs pour une itération, on aura une mesure de l'effort de calcul beaucoup plus fiable.

La figure 1 illustre ces principes de comparaison de deux heuristiques itératives. L'axe horizontal a une double gradation : le temps de calcul et le nombre d'itérations effectuées par l'une des heuristiques itératives. Sur le diagramme supérieur, l'axe vertical indique la valeur moyenne de la fonction-objectif que l'on désire minimiser. Sur le diagramme inférieur, l'axe vertical donne la probabilité qu'une heuristique soit meilleure que l'autre sur la base d'un test de Mann-Whitney. Les seuils de signification sont indiqués par deux lignes horizontales. Ce second diagramme permet de savoir immédiatement si une heuristique (celle représentée en gras dans le premier diagramme) est significativement moins bonne (on se trouve au-dessous du seuil inférieur) ou meilleure (on est au-dessus du seuil supérieur) que l'autre. De plus, on a indiqué en gras le fait que même si l'heuristique était programmée de manière à s'exécuter deux fois plus rapidement (respectivement : plus lentement), elle resterait significativement moins bonne (respectivement : meilleure) que l'autre.

Une telle figure permet de tirer des conclusions beaucoup plus nuancées et mieux étayées que celles que l'on aurait obtenues en donnant les résultats sous la forme d'un tableau. En effet, on aurait eu tendance à affirmer que l'heuristique FANT (Taillard 2000) est meilleure que MMAS (Stützle et Hoos 1999) — ou vice-versa ! — sur la base d'une comparaison de la qualité moyenne des solutions obtenues après un temps de calcul fixé et arbitraire. Si les résultats avaient été

donnés sous la forme du premier diagramme, on aurait eu tendance à dire que MMAS est meilleure que FANT pour des temps de calcul inférieurs à 1,5 secondes alors que cette conclusion (qui n'est pas forcément fausse) ne peut être valablement déduite des expériences numériques réalisées.

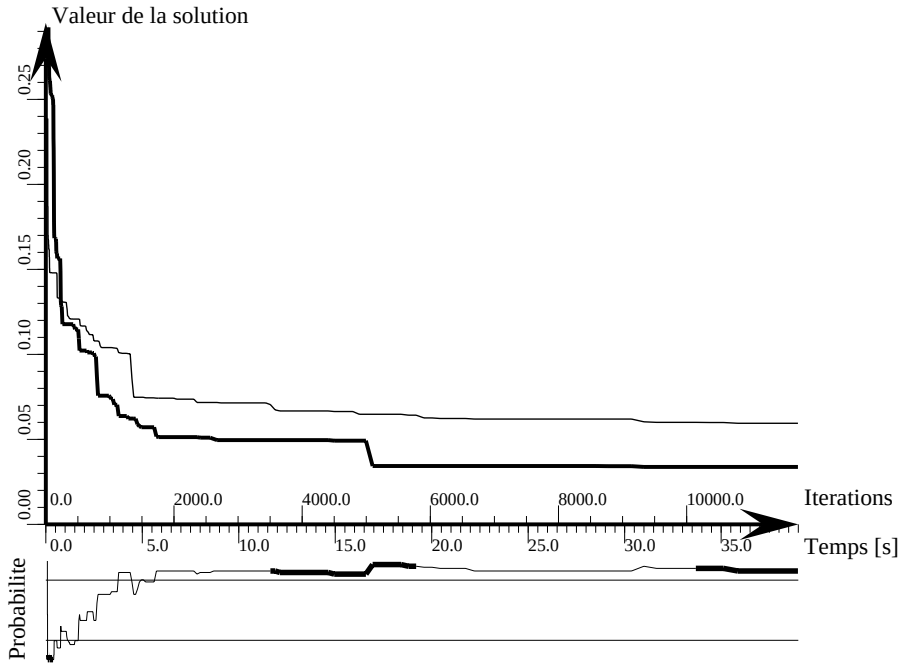

Figure 1. *Comparaison des heuristiques FANT et MMAS pour l'exemple du problème d'affectation quadratique bur26a. Le diagramme supérieur donne la valeur moyenne des solutions produites par chaque méthode sur 10 exécutions indépendantes en fonction du temps de calcul ou du nombre d'itérations d'une recherche locale incluse dans ces méthodes. Le diagramme inférieur donne la probabilité que FANT soit meilleure que MMAS. Une probabilité plus basse que le seuil de confiance (de 20%, donc peu fiable) indique que MMAS est significativement meilleure que FANT ; une probabilité indiquée en gras indique que même si l'une ou l'autre des méthodes s'exécutait plus rapidement d'un facteur 2, la comparaison resterait significative.*

Une comparaison détaillée telle que présentée en figure 1 ne peut se faire que pour un nombre très restreint de méthodes et/ou exemples de problèmes. Lorsque le nombre de méthodes à comparer est important, il est possible de le faire dans une seule figure en superposant pour chaque méthode un diagramme donnant la probabilité qu'elle soit meilleure qu'une méthode de référence.

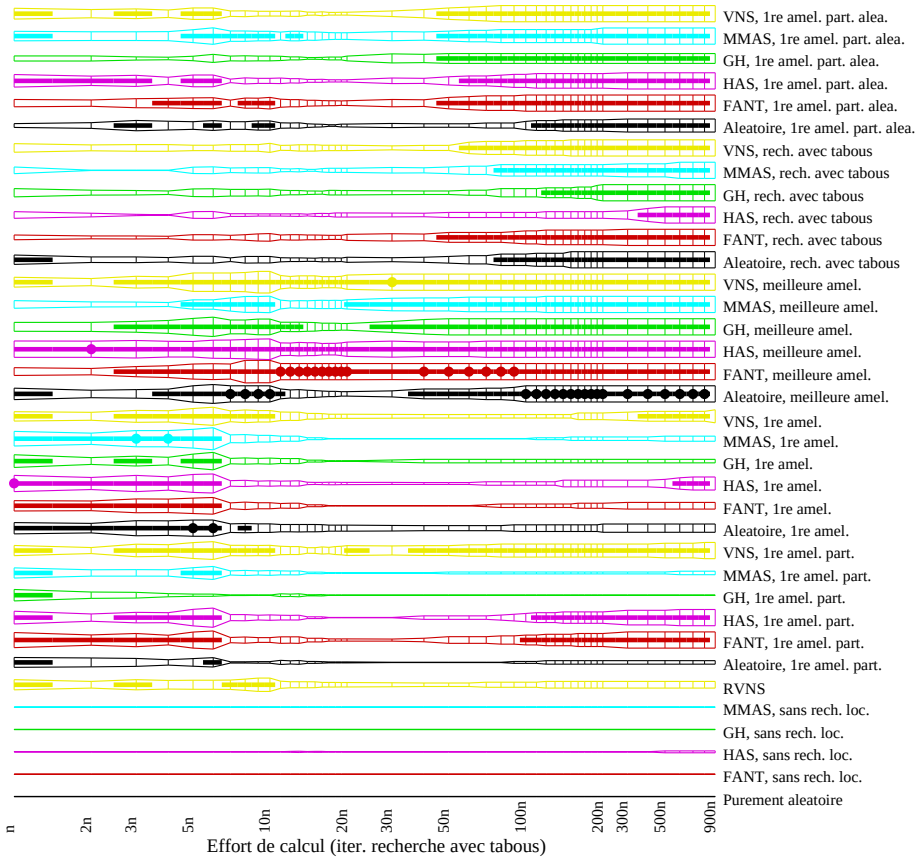

Figure 2. *Comparaison de 36 heuristiques basées sur des programmes à mémoire adaptative (problème d'affectation quadratique els19). Les heuristiques considérées ont été construites en combinant 6 manières de construire des solutions (saut aléatoire (VNS), diverses mémoires statistiques d'inspiration «colonie de fourmis» (MMAS, HAS, FANT), population de solutions (GH), solution aléatoire) et 6 implantations de recherches locales. Un gris différent est attribué pour chaque méthode de construction.*

En figure 2, une représentation un peu différente a été adoptée : à chaque méthode on fait correspondre une bande dont la largeur est modulée en fonction de la statistique S utilisée pour conduire un test de Mann-Whitney, c'est-à-dire proportionnelle à la somme des rangs d'une méthode de référence. Une bande étroite indique que la méthode de référence est meilleure car elle produit des solutions

systématiquement de qualité plus élevée pour l'exemple de problème considéré. Un trait épais à l'intérieur d'une bande indique que la méthode de référence n'est pas significativement meilleure que la méthode à laquelle on la compare.

Dans la figure 2, la méthode de référence est celle dont la qualité moyenne des solutions est la plus élevée. Elle peut donc changer en fonction de l'effort de calcul. Un petit disque à l'intérieur d'une bande identifie la méthode de référence pour un effort donné. On remarquera dans cette figure l'utilisation d'une échelle logarithmique pour mesurer l'effort de calcul, exprimé en équivalent d'itérations de recherches avec tabous. Comme on observe qu'il faut, très grossièrement, n itérations avant d'atteindre un optimum local pour un problème de taille n (il n'y a formellement aucune relation connue entre la taille du problème et le nombre d'itérations nécessaires pour atteindre un optimum local), la gradation est exprimée en multiples de n itérations. Ainsi, tout à gauche de l'axe, on a un temps de calcul équivalent à une heuristique très rapide s'arrêtant au premier optimum local, alors que tout à droite on a des temps de calcul très importants.

De manière similaire, on aurait pu comparer deux heuristiques itératives pour tout un jeu d'exemples de problèmes, en créant une bande pour chacun de ces derniers. À l'aide de ces techniques, il est possible de procéder à des analyses d'heuristiques robustes et nuancées. Vu la compacité des diagrammes proposés, on pourra également donner des résultats négatifs. En effet, on a tendance à occulter les résultats obtenus avec une heuristique qui fonctionne mal, alors même que cela aurait pu nous aider à comprendre pourquoi telle autre heuristique fonctionne bien. Ces techniques de comparaisons sont également utiles pour procéder à des ajustements de paramètres de méthodes itératives.

2.7 Bibliographie

[ANG 98a] ANGEL É., *La rugosité des paysages : une théorie pour la difficulté des problèmes d'optimisation combinatoire relativement aux méta-heuristiques*, thèse de doctorat, Université de Paris-Sud, U. F. R. scientifique d'Orsay, Orsay, 1998.

[ANG 98b] ANGEL É. et ZISSIMOPOULOS V., « On the quality of local search for the quadratic assignment problem », *Discrete Applied Mathematics 82*, 1998, 15–25.

[APP 91] APPLEGATE D. et COOK W., (1991), « A computational study of the job-shop scheduling problem », *ORSA Journal on Computing 3*, 1991, 149–156.

[BER 58] BERGE C., *Théorie des graphes et ses applications*, Dunod, Paris, 1958.

[B•A 94] BLAZEVICZ J., ECKER K. H., SCHMIDT G. et WEGLARZ J. *Scheduling in Computer and Manufacturing Systems*, Springer, Berlin, 1994

[BOCK 58] BOCK F., « An algorithm for solving " traveling-salesman " and related network optimization problems », manuscrit associé à une présentation au *Fourteenth National Meeting of the Operations Research Society of America*, St. Louis, MO, 1958.

[BRE 66] BREMERMANN H. J., ROGHSON J., SALAFF S., « Global properties of evolution processes », dans : PATTEE H. H., EDELSAK E. A., FEIN L., CHALLAHAN A. B. (éditeurs), *Natural Automata and Useful Simulations*, Macmillan, Londres, 1966, 3–42.

[BUD 99] BÜDENBENDER K., GRÜNERT T. et SEBASTIAN H.-J., « A hybrid tabu search/branch and bound algorithm for the direct flight network design problem », rapport technique, Institut für Wirtschaftswissenschaften, Aachen, Allemagne, 1999. À paraître dans *Transportation Science*.

[COL 92] COLORNI, A., DORIGO M. et MANIEZZO V., « Distributed optimization by ant colonies », dans : *Toward a practice of autonomous systems : proceedings of the First Europena Conference on Artificial Life* (ECAL91), VARELA F. J. et BOURGINE P. (éditeurs), MIT Press, Cambridge, 1992, 134–142.

[CON 99] CONOVER W. J., *Practical Nonparametric Statistics*, 3e édition, Wiley, 1999.

[CROE 58] CROES G. A., « A method for solving traveling salesman problems », *Operations Research 6*, 1958, 791–812.

[DOR 94] DORNDORF U. et PESCH E., « Fast Clustering Algorithms », *ORSA Journal on Computing 6*, 1994, 141–153.

[GLO 77] GLOVER F., « Heuristics for integer programming using surrogate constraints », *Decision Sciences 8*, 1977, 156–166.

[GLO 97] GLOVER F. et LAGUNA M., *Tabu Search*, Kluwer, 1997.

[HAN 99] HANSEN P. et MLADENOVIC N., « An introduction to variable neighborhood search », dans : VOß S., MARTELLO S., OSMAN I. H. et ROUCAIROL C. (éditeurs), *Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization*, Kluwer, 1999, 422–458.

[HOL 75] HOLLAND J. H., *Adaptation in Natural and Artificial Systems*, presses de l'université du Michigan, Ann Arbor, 1975.

[LAW 85] LAWLER E. L., LENSTRA J. K., RINNOOY KAN A. G. H. et SCHMOYS D. B. *The Traveling Salesman Problem, A Guided Tour of Combinatorial Optimization*, Wiley, 1985.

[KLE 72] KLEE V. et MINTY G., « How good is the simplex algorithm ? », dans : SHISHA O. (éditeur), *Inequalities III*, Academic Press, New-York, 1972, 159–175.

[MAU 97] MAUTOR T. et MICHELON P., « MIMAUSA : a new hybrid method combining exact solution and local search », résumé étendu de la *2e conférence internationale sur les métaheuristiques*, Sophia-Antipolis, France, 1997.

[MUL 97] MÜHLENBEIN H., « Genetic algorithms », dans : AARTS E. et LENSTRA J. K. (éditeurs), *Local Search in Combinatorial Optimization*, Wiley, 1997, 137–171.

[PES 95] PESCH E. et GLOVER F., « TSP Ejection Chains », *Discrete Applied Mathematics 76*, 1997, 165–181.

[REC 73] RECHENBERG I., *Evolutionsstrategie : Optimierung technischer System nach Prinzipien der biologischen Information*, Fromman, Freiburg, 1973.

[REE 99] REEVES C. R. « Landscapes, operators and heuristic search », dans : SHARAIHA Y. M. et BEASLEY J. E. (éditeurs), *Advances in Combinatorial Optimization* (*Annals of Operations Research 86*), 1999, 473–490.

[REG 96] REGO C. et ROUCAIROL C., « A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem », dans : OSMAN I. H. et KELLY J. P. (éditeurs), *Meta-Heuristics : Theory and Applications*, Kluwer, 1996, 661–675.

[ROC 95] ROCHAT Y. et TAILLARD É. D., « Probabilistic diversification and intensification in local search for vehicle routing », *Journal of Heuristics 1*, 1995, 147–167.

[SHA 98] SHAW P., *Using constraint programming and local search methods to solve vehicle routing problems*, rapport technique, ILOG S.A., Gentilly, France, 1998.

[STÜ 99] STÜTZLE T. et HOOS H. H., « The Max-Min Ant System and Local Search for Combinatoiral Optimization Problems », dans VOß S., MARTELLO S., OSMAN I. H. et ROUCAIROL C. *Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization*, Kluwer, 1999, 313–329.

[TAI 90] TAILLARD É. D., « Some efficient heuristic methods for the flow shop sequencing problem », *European Journal of Operational Research 47*, 1990, 65–74.

[TAI 91] TAILLARD É. D., « Robust taboo search for the quadratic assignment problem », *Parallel Computing 17*, 1991, 443–455.

[TAI 93] TAILLARD É. D., « Parallel iterative search methods for vehicle routing problems », *Networks 23*, 661–673.

[TAI 94] TAILLARD É. D., « Parallel taboo search techniques for the job shop scheduling problem », *ORSA journal on Computing 6*, 1994, 108–117.

[TAI 98] TAILLARD É. D., *Programmation à mémoire adaptative et algorithmes pseudo-gloutons : nouvelles perspectives pour les méta-heuristiques*, thèse d'habilitation, Université de Versailles, France, 1998.

[TAI 00] TAILLARD É. D., « An Introduction to Ant Systems », dans LAGUNA M. et GONZALEZ-VELARDE J. L. (éditeurs), *Computing Tools for Modeling, Optimization and Simulation*, Kluwer, 2000, 131–144

[TAI 99] TAILLARD É. D. et VOß S., *POPMUSIC : Partial Optimization Metaheuristic Under Special Intensification Conditions*, rapport technique, Institut d'informatique, EIVD, Yverdon, 1999.

[TAT 95] TATE D. E. et SMITH A. E., « A genetic approach to the quadratic assignment problem» , *Computers & Operations Research 22*, 1995, 73–83.

[WID 89] WIDMER M., HERTZ A., « A new heuristic method for the flow shop sequencing problem », *European Journal of Operational Research 41*, 1989, 186–193.

[XU 96] XU J., KELLY J., « A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem », *Transportation Science 30*, 1996, 379–393