Revue d'Intelligence Artificielle Vol : No. 1999

Méthaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes

Jin-Kao Hao[*], Philippe Galinier[**], Michel Habib[***]

[*] *LERIA, U.F.R. Sciences, Université d'Angers, 2 bd Lavoisier, 49045 Angers, Jin-Kao.Hao@univ-angers.fr*
[**] *LGI2P, Ecole des Mines d'Alès, Parc Scientifique Georges Besse, 30000 Nîmes, galinier@eerie.fr*
[***] *LIRMM, URA CNRS, 161 rue Ada, 34392 Montpellier, habib@lirmm.fr*

RÉSUMÉ. *Nous présentons une synthèse de principales métaheuristiques : les méthodes de voisinage, les algorithmes évolutifs et les algorithmes hybrides. Nous proposons une analyse de ces métaheuristiques en dégageant les idées fondamentales et avançons des pistes pour guider le choix d'une métaheuristique en pratique. Enfin, nous discutons les limitations de ces méthodes et présentons quelques voies de recherche. Les références de l'article donnent des pointeurs sur les métaheuristiques et leurs applications.*

ABSTRACT. *We present an overview of the main metaheuristics including neighbourhood search, evolutionary and hybrid methods. We analyse these metaheuristics by identifying some fundamental principles and propose guidelines for choosing metaheuristics in practice. We discuss the limits of these methods and present research perspectives. The references included in the paper give further pointers on metaheuristics and their applications.*

MOTS-CLÉS : *optimisation combinatoire, affectation sous contraintes, heuristiques, métaheuristiques, méthodes de voisinage, algorithmes évolutifs, méthodes hybrides.*

KEY WORDS: *combinatorial optimisation, constrained problems, heuristics, metaheuristics, neighbourhood search, evolutionary algorithms, hybrid methods.*

1. Introduction

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation [PAP 82] et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire [RIB 94]. Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données [GAR 79].

Etant donnée l'importance de ces problèmes, de nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO) et en intelligence artificielle (IA). Ces méthodes peuvent être classées sommairement en deux grandes catégories : les méthodes exactes (complètes) qui garantissent la complétude de la résolution et les méthodes approchées (incomplètes) qui perdent la complétude pour gagner en efficacité.

Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des solutions de l'espace de recherche. Pour améliorer l'énumération des solutions, une telle méthode dispose de techniques pour détecter le plus tôt possible les échecs (calculs de bornes) et d'heuristiques spécifiques pour orienter les différents choix. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles les techniques de séparation et évaluation progressive (SEP) ou les algorithmes avec retour arrière. Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrès réalisés (notamment en matière de la programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante.

Les méthodes approchées constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. En effet, ces méthodes sont utilisées depuis longtemps par de nombreux praticiens. On peut citer les méthodes gloutonnes et l'amélioration itérative : par exemple, la méthode de Lin et Kernighan qui resta longtemps le champion des algorithmes pour le problème du voyageur de commerce [LIN 73].

Depuis une dizaine d'années, des progrès importants ont été réalisés avec l’apparition d’une nouvelle génération de méthodes approchées puissantes et générales, souvent appelées *métaheuristiques* [REE 93a, AAR 97]. Une métaheuristique est constituée d’un ensemble de concepts fondamentaux (par exemple, la liste tabou et les mécanismes d’intensification et de diversification pour la métaheuristique tabou), qui permettent d'aider à la conception de méthodes heuristiques pour un problème d'optimisation[1]. Ainsi les métaheuristiques sont adaptables et applicables à une large classe de problèmes.

Les métaheuristiques sont représentées essentiellement par les *méthodes de voisinage* comme le recuit simulé et la recherche tabou, et les *algorithmes évolutifs* comme les algorithmes génétiques et les stratégies d'évolution. Grâce à ces métaheuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications qu'il était impossible de traiter auparavant [LAP 96, OSM 96]. On constate, depuis ces dernières années, que l'intérêt porté aux métaheuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle.

Cet article s'intéresse aux principales métaheuristiques : les méthodes de voisinage, les algorithmes évolutifs ainsi que les méthodes hybrides. Après l'introduction des problèmes d'optimisation combinatoire et d'affectation sous contraintes (section 2), nous donnons un panorama des méthodes de résolution représentatives en RO et en IA (section 3). Nous présentons ensuite les différentes métaheuristiques (section 4). Nous analysons ces méthodes pour dégager quelques principes fondamentaux, comparons les performances sur deux problèmes de référence et proposons des pistes pour guider le choix d'une métaheuristique en pratique (section 5). Nous discutons enfin les limitations des métaheuristiques et préconisons quelques voies de recherche.

2. Optimisation combinatoire et affectation sous contraintes

2.1. *Optimisation combinatoire*

Un problème d'optimisation combinatoire est défini par un ensemble d'instances. A chaque instance du problème est associé un ensemble discret de solutions S, un sous-ensemble X de S représentant les solutions admissibles (réalisables) et une fonction de coût f (ou fonction objectif) qui assigne à chaque solution $s \in X$ le nombre réel (ou entier) $f(s)$. Résoudre un tel problème (plus précisément une telle instance du problème) consiste à trouver une solution $s^* \in X$ optimisant la valeur de la fonction de coût f. Une telle solution s^* s'appelle une *solution optimale* ou un *optimum global*. Nous avons donc la définition suivante :

Définition [PAP 82]. Une *instance I* d'un problème de minimisation est un couple (X, f) où $X \subseteq S$ est un ensemble fini de solutions admissibles, et f une fonction

[1] Une *heuristique* est une méthode, conçue pour un problème d'optimisation donné, qui produit une solution non nécessairement optimale lorsqu'on lui fournit une instance de ce problème. Une métaheuristique est définie de manière similaire, mais à un niveau d'abstraction plus élevé (d'après E. Taillard). Le terme « métaheuristique » a été initialement utilisé par F. Glover pour distinguer la méthode tabou des heuristiques spécifiques [GLO 86]. Notons que ce terme est également utilisé par J-L. Laurière dans son système de résolution Alice [LAU 78].

de coût (ou objectif) à minimiser $f : X \rightarrow R$. Le problème est de trouver $s^* \in X$ tel que $f(s^*) \leq f(s)$ pour tout élément $s \in X$.

Notons que d'une manière similaire, on peut également définir les problèmes de maximisation en remplaçant simplement ≤ par ≥. L'optimisation combinatoire trouve des applications dans des domaines aussi variés que la gestion, l'ingénierie, la conception, la production, les télécommunications, les transports, l'énergie, les sciences sociales et l'informatique elle-même.

2.2. Problème d'affectation sous contraintes

La définition générale de l'optimisation ne précise ni la forme des solutions de *S,* ni la façon de générer ces solutions (admissibles ou non admissibles). Nous nous intéressons en particulier à une classe importante de problèmes dont une solution peut être décrite explicitement par une *affectation* de valeurs à l'ensemble des variables du problème. Etant donné un ensemble fini $V = \{V_1,...,V_n\}$ de variables et un ensemble $D = \{D_1,...,D_n\}$ de domaines finis associés, une solution potentielle du problème (affectation) consiste à choisir pour chaque variable V_i $(1 \leq i \leq n)$ une valeur choisie dans son domaine D_i. L'ensemble S des solutions potentielles est donc représenté par le produit cartésien $D_1 \times ... \times D_n$ des domaines. On dispose en outre d'un ensemble $C = \{C_1,...,C_p\}$ de contraintes : chaque contrainte C_j $(1 \leq j \leq p)$ est une relation sur un sous-ensemble V'_j de V qui spécifie quelles combinaisons de valeurs sont compatibles pour les variables de V'_j.

Nous utilisons le terme informel de « *problèmes d'affectation sous contraintes* » (PASC) pour qualifier la classe des problèmes d'affectation utilisant la notion de contrainte (en étendant éventuellement la définition de contrainte présentée plus haut).

Cette classe de problèmes inclut notamment les problèmes de satisfaction de contraintes (CSP pour *Constraint Satisfaction Problems*) [MAC 77, 87], les problèmes de satisfaction partielles (maximales) (MCSP) [FRE 92] et les problèmes d'optimisation sous contraintes (CSOP) [TSA 93].

Etant donné un triplet $<V,D,C>$, le problème CSP consiste à trouver une assignation qui satisfait toutes les contraintes et le problème MCSP une assignation qui satisfait un nombre maximum de contraintes. Etant donné un quadruplé $<V,D,C,f>$, le problème CSOP consiste à trouver une assignation qui satisfait toutes les contraintes et qui minimise la fonction $f : S \rightarrow R$.

Il existe de nombreux autres modèles comme les CSP flous, possibilistes et probabilistes [FAR 95] ainsi que les CSP valués [SCH 97] qui peuvent être éventuellement inclus dans la classe PASC.

Les problèmes d'affectation sous contraintes possèdent de très nombreuses applications pratiques concernant l'affectation de ressources, le groupement, la classification, la planification, l'emploi du temps et l'ordonnancement, dans des domaines très variés. Les PASC permettent également de modéliser facilement des problèmes de référence comme par exemple la k-coloration et la satisfiabilité.

2.3. *Complexité*

Un problème est dit *polynomial* s'il existe un algorithme permettant de trouver une solution optimale pour toutes ses instances en un temps polynomial par rapport à la taille de l'instance. Un tel algorithme est dit *efficace* pour le problème en question. C'est notamment le cas de certains problèmes de plus court chemin dans un graphe valué, du recouvrement d'un graphe valué par un arbre de poids minimum, des problèmes classiques de flots, ainsi que pour les problèmes Horn-SAT et 2-SAT (notons cependant que MAX-2-SAT reste NP-difficile). Cependant, pour la majorité des problèmes d'optimisation combinatoire, aucun algorithme polynomial n'est connu actuellement.

La difficulté intrinsèque de ces problèmes est bien caractérisée par la théorie de la NP-complétude [GAR 79]. De nombreux problèmes d'optimisation combinatoire (la plupart de ceux qui sont vraiment intéressants dans les applications !) ont été prouvés NP-difficiles[2].

Cette difficulté n'est pas seulement théorique et se confirme hélas dans la pratique. Il arrive que des algorithmes exacts de complexité exponentielle se comportent efficacement face à de très grosses instances - pour certains problèmes et certaines classes d'instances. Mais c'est très souvent l'inverse qui se produit ; pour de nombreux problèmes, les meilleures méthodes exactes peuvent être mises en échec par des instances de taille modeste, parfois à partir de quelques dizaines de variables seulement. Par exemple, on ne connaît aucune méthode exacte qui soit capable de colorier de façon optimale un graphe aléatoire de densité 1/2 lorsque le nombre de sommets dépasse 90 [JOH 91]. Or pour le problème d'affectation de fréquences dans le domaine des réseaux radio-mobiles qui est une extension du problème de coloration, nous avons eu à traiter des instances comportant plus d'un millier de variables. Pour traiter les grosses instances de ce type de problèmes, on se contente de solutions approchées obtenues avec une méthode heuristique.

3. Introduction Méthodes de résolution

3.1. *Un panorama*

Un très grand nombre de méthodes de résolution existent en RO et en IA pour l'optimisation combinatoire et l'affectation sous contraintes. La figure 1 met en parallèle les méthodes représentatives développées en RO et en IA, avec à titre indicatif la date approximative d'apparition de chaque méthode.

[2] Cette classe contient des problèmes pour lesquels aucun algorithme polynomial n'est connu. De plus, on conjecture qu'il n'existe pas d'algorithme polynomial pour ces problèmes.

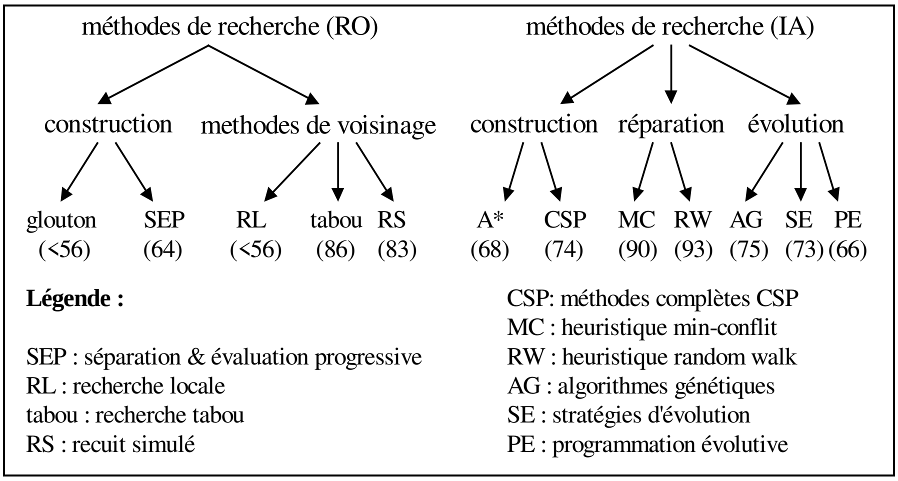

Figure 1. *Classement des méthodes de résolution*

D'une manière très générale, les méthodes de résolution suivent quatre approches différentes pour la recherche d'une solution : l'approche de construction, l'approche de relaxation, l'approche de voisinage et l'approche d'évolution.

Ces méthodes font partie de deux groupes de nature différente. Le premier groupe comprend les méthodes exactes d'arborescence qui garantissent la complétude de la résolution : c'est le cas de SEP, A* et CSP. Le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre (dans le pire des cas). Pour améliorer l'efficacité de la recherche, on utilise des techniques variées pour calculer des bornes permettant d'élaguer le plus tôt possible des branches conduisant à un échec. Parmi ces techniques, on peut citer les différentes relaxations[3] : la relaxation de base en programmation linéaire, la relaxation lagrangienne [HEL 70, 71, BEA 93], la relaxation agrégée (*surragote relaxation*) [GLO 65, 77] et la décomposition lagrangienne. De plus, on emploie des heuristiques pour guider les choix de variables et de valeurs durant l'exploration de l'arborescence.

Le second groupe comprend les méthodes approchées dont le but est de trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue. Les méthodes approchées sont fondées principalement sur diverses heuristiques, souvent spécifiques à un type de problème. Les techniques de relaxation permettent également de fournir des solutions approchées.

Les métaheuristiques constituent une autre partie importante des méthodes approchées et ouvrent des voies très intéressantes en matière de conception de méthodes heuristiques pour l'optimisation combinatoire.

La suite de cet article est essentiellement consacrée aux métaheuristiques : les méthodes de voisinage et les algorithmes évolutifs. Nous présentons également les possibilités de combiner différentes méthodes pour créer des méthodes hybrides.

---

[3] Ces techniques de relaxations ne concernent pas A* ni les méthodes CSP.

Nous commençons dans la section suivante par une brève introduction de l'approche de construction afin de mettre en contraste le principe d'augmentation successive de cette approche avec les principes de réparation et d'évolution.

3.2. *Approche de construction*

L'approche de construction est probablement la plus ancienne et occupe traditionnellement une place très importante en optimisation combinatoire et en intelligence artificielle. Une méthode de construction construit pas à pas une solution de la forme $s = (<V_1,v_1> <V_2,v_2>...<V_n,v_n>)$. Partant d'une solution partielle initialement vide $s = ()$, elle cherche à étendre à chaque étape la solution partielle $s = (<V_1,v_1>...<V_{i-1},v_{i-1}>)$ $(i \leq n)$ de l'étape précédente. Pour cela, elle détermine la prochaine variable V_i, choisit une valeur v_i dans D_i et ajoute $<V_i,v_i>$ dans s pour obtenir une nouvelle solution partielle $s = (<V_1,v_1>...<V_{i-1},v_{i-1}> <V_i,v_i>)$. Ce processus se répète jusqu'à ce que l'on obtienne une solution complète.

Durant la recherche d'une solution, une méthode de construction fait intervenir des heuristiques pour effectuer chacun des deux choix : le choix de la variable suivante et le choix de la valeur pour la variable. Les méthodes de cette classe diffèrent entre elles selon les heuristiques utilisées. En général, les heuristiques portent plus souvent sur le choix de variables que sur le choix de valeurs car les informations disponibles concernant le premier choix semblent souvent plus riches. La performance de ces méthodes dépend largement de la pertinence des heuristiques employées, *i.e.*, de leur capacité d'exploiter les connaissances du problème.

Un premier type de méthodes de construction est représenté par les *méthodes gloutonnes*. Une méthode gloutonne consiste à fixer à chaque étape la valeur d'une variable sans remettre en cause les choix effectués précédemment [BER 64, LAW 66, PAP 82]. Par exemple, une heuristique gloutonne bien connue pour la coloration introduite par Brélaz est la suivante : pour choisir le nœud suivant à colorier, prendre celui dont les nœuds adjacents sont déjà coloriés avec le plus grand nombre de couleurs différentes [BRE 79] et lui assigner la couleur autorisée de plus petit rang possible. Les méthodes gloutonnes sont généralement rapides, mais fournissent le plus souvent des solutions de qualité médiocre. Elles ne garantissent l'optimum que dans des cas particuliers, par exemple la présence d'une structure de matroïde [KOR 91, RAR 93].

Un deuxième type de méthode de construction est représenté par les méthodes avec *retour arrière*. Une méthode de retour arrière avec une stratégie de recherche en profondeur d'abord consiste à fixer à chaque étape la valeur d'une variable. Aussitôt qu'un échec est détecté, un retour arrière est effectué, *i.e.*, une ou plusieurs instanciations déjà effectuées sont annulées et de nouvelles valeurs recherchées [BIT 75]. Par exemple, un algorithme typique avec retour arrière pour la résolution d'un problème de satisfaction de contraintes cherche à prolonger à chaque étape l'assignation courante de manière consistante. En cas d'échec, un retour arrière est effectué sur la dernière variable instanciée possédant encore des valeurs non essayées [MAC 87]. Les méthodes avec retour arrière sont en général complètes et de complexité exponentielle. Pour réduire le nombre de retour arrière (et le temps de recherche), on utilise des techniques de filtrage afin d'anticiper le plus tôt possible les échecs. Citons les systèmes de programmation sous contraintes comme par exemple

ALICE [LAU 78], CHIP [DIN 88], Prolog III [COL 90] ou ILOG Solver [PUG 94] qui sont essentiellement fondés sur le principe de retour arrière.

Une troisième type de méthode de construction concerne de nombreux algorithmes basés sur le principe de séparation et évaluation progressive [ROY 65, PAP 82]. Un exemple typique est l'algorithme A* avec une stratégie « meilleur d'abord » pour la recherche d'un plus court chemin dans un graphe valué [PEA 84]. Cet algorithme débute avec un nœud initial et prolonge ensuite parmi l'ensemble de chemins partiels développés le chemin dont la longueur est la plus faible, en utilisant une estimation de la longueur restante pour calculer la borne inférieure. Lorsqu'un chemin complet est trouvé, les chemins partiels dont la longueur est supérieure à celle du chemin complet sont éliminés. De manière générale, on utilise des techniques de relaxations pour obtenir des bornes aussi serrées que possible (un exemple simple consiste à relâcher la contrainte d'intégrité pour appliquer un algorithme de simplexe).

Il existe un nombre important de publications dans la littérature concernant les méthodes de construction, voir par exemple les références [NIC 71, SIL 80, PAP 82, EGL 86, FIS 89, ZAN 89, PEA 84, SHA 87].

3.3. *Recherche locale*

La recherche locale[4], appelée aussi la descente ou l'amélioration itérative, représente une classe de méthodes heuristiques très anciennes [FLO 56, CRO 58]. Traditionnellement, la recherche locale constitue une arme redoutable pour attaquer des problèmes réputés très difficiles tels que le voyageur de commerce [LIN 65, LIN 73] et la partition de graphes [KER 70]. Contrairement à l'approche de construction, la recherche locale manipule des configurations complètes durant la recherche.

Une méthode de recherche locale est un processus itératif fondé sur deux éléments essentiels : un voisinage $N : X \rightarrow 2^X$ (voir § 4.1.) et une procédure exploitant le voisinage. Plus précisément, elle consiste à 1) débuter avec une configuration quelconque s de X, et 2) choisir un voisin s' de s tel que $f(s') < f(s)$ et remplacer s par s' et à répéter 2) jusqu'à ce que pour tout voisin s' de s, $f(s') \geq f(s)$.

Cette procédure fait intervenir à chaque itération le choix d'un voisin qui améliore la configuration courante. Plusieurs possibilités peuvent être envisagées pour effectuer ce choix. Il est possible d'énumérer les voisins jusqu'à ce qu'on en découvre un qui améliore strictement (première amélioration). On peut également rechercher le meilleur voisin (meilleure amélioration). Cette dernière solution peut sembler plus coûteuse, mais le voisin découvert sera en général de meilleure qualité. De plus, l'utilisation d'une structure de données appropriée peut souvent permettre de trouver directement ce meilleur voisin.

Comme l'espace des solutions X est fini, cette procédure de descente s'arrête toujours, et la dernière configuration trouvée ne possède pas de voisin strictement meilleur qu'elle-même. Autrement dit, la recherche locale retourne toujours un optimum local.

---

[4] Notons que dans la littérature, le terme « la recherche locale » est de plus en plus employé pour désigner la classe des méthodes de voisinage (§ 4.1.) au lieu de la descente seule.

L'avantage principal de cette méthode réside dans sa grande simplicité et sa rapidité. Mais les solutions produites sont souvent de qualité médiocre et de coût très supérieur au coût optimal. Pour remédier à ce problème, la solution la plus simple est la *méthode de relance aléatoire* qui consiste à générer une nouvelle configuration de départ de façon aléatoire et à recommencer une descente. On remarque cependant que cette solution ne tire aucun profit des optima locaux déjà découverts. Une autre solution consiste à accepter des voisins de même performance que la configuration courante. Cette approche permet à la recherche de se déplacer sur les plateaux, mais n'est pas suffisante pour ressortir de tous les optima locaux. D'autres raffinements plus élaborés sont également possibles, par exemple, : l'introduction de voisinages variables [KER 70, LIN 73] et les techniques de réduction [LIN 65, LIN 73] ou d'élargissement [STE 68].

La recherche locale est à la base de métaheuristiques comme la méthode tabou et des méthodes hybrides. Notons enfin qu'on trouve également l'idée de recherche locale dans le célèbre algorithme du simplexe pour la programmation linéaire [PAP 82].

Maintenant, nous allons présenter dans la section suivante trois grandes classes de métaheuristiques, à savoir les méthodes de voisinage, les algorithmes évolutifs, et les méthodes hybrides.

4. Métaheuristiques

4.1. *Méthodes de voisinage*

Les méthodes de voisinage sont fondées sur la notion de voisinage. Nous allons donc introduire d'abord cette notion fondamentale ainsi que quelques notions associées.

Définition. Soit X l'ensemble des configurations admissibles d'un problème[5], on appelle *voisinage* toute application $N : X \rightarrow 2^X$. On appelle *mécanisme d'exploration* du voisinage toute procédure qui précise comment la recherche passe d'une configuration $s \in X$ à une configuration $s' \in N(s)$. Une configuration s est un *optimum (minimum) local* par rapport au voisinage N si $f(s) \leq f(s')$ pour toute configuration $s' \in N(s)$.

Une méthode typique de voisinage débute avec une configuration initiale, et réalise ensuite un processus itératif qui consiste à remplacer la configuration courante par l'un de ses voisins en tenant compte de la fonction de coût. Ce processus s'arrête et retourne la meilleure configuration trouvée quand la condition d'arrêt est réalisée. Cette condition d'arrêt concerne généralement une limite pour le nombre d'itérations ou un objectif à réaliser. Un des avantages des méthodes de voisinage réside précisément dans la possibilité de contrôler le temps de calcul : la qualité de la solution trouvée tend à s'améliorer progressivement au cours du temps et l'utilisateur est libre d'arrêter l'exécution au moment qu'il aura choisi. Les méthodes de voisinage diffèrent essentiellement entre elles par le voisinage utilisé et la stratégie de parcours de ce voisinage. La recherche locale est un exemple simple de cette classe de méthodes.

---

[5] Un voisinage peut être également défini sur *S*, l'ensemble des configurations du problème.

Un voisinage peut être représenté à l'aide d'un graphe orienté : les nœuds sont les configurations et il existe un arc entre deux nœuds *s* et *s'* si *s'* est voisin de *s*. Plus précisément, on a la définition suivante :

Définition. Soit *N* un voisinage et *X* l'ensemble des configurations admissibles d'une instance d'un problème. Le graphe (orienté) de l'espace de solutions *S* induit par *N* est défini par $G_N = (X,E)$ tel que pour tout $s, s' \in X$, $<s, s'> \in E$ si et seulement si $s' \in N(s)$.

A chaque arc $<s_i, s_j> \in E$ du graphe, on peut également associer une valeur $c_{ij} = f(s_j) - f(s_i)$ qui correspond à la variation de coût entre s_i et s_j.

Avec cette notion de graphe, une méthode de voisinage peut être vue comme un processus qui parcourt un chemin du graphe. A chaque nœud, le mécanisme de parcours choisit l'arc à parcourir essentiellement en fonction des valeurs des arcs partant du nœud courant. L'efficacité de la méthode dépend donc de deux choses : la structure du graphe déterminée par le voisinage et la façon de parcourir le graphe déterminée par le mécanisme de parcours du voisinage.

Pour les problèmes d'affectation sous contraintes PASC, nous pouvons préciser une représentation des configurations et proposer un voisinage simple et généralement efficace.

Définition. Etant donné un problème d'affectation sous contraintes PASC $<V,D,C,f>$, une configuration *s* est une affectation définie par $s = \{<V_i,v_i> \mid V_i \in V$ et $v_i \in D_i\}$. La notation $s(i)$ représente la valeur de V_i dans la configuration *s*, *i.e.* pour tout $<V_i,v_i> \in s$, $s(i) = v_i$.

Ainsi, l'ensemble *S* des configurations d'une instance du problème contient toutes les affectations possibles. Il est clair que le cardinal de *S* est égal au produit de la taille de tous les domaines, *i.e.* $\Pi|D_i|$ $(1 \leq i \leq n)$.

A partir de cette représentation des configurations, nous pouvons définir un voisinage à la fois très simple et général.

Définition. Soit *s* une configuration dans *S*, le voisinage $N : S \rightarrow 2^S$ est une application telle que pour tout $s, s' \in N(s)$ si et seulement si

- il existe un et un seul i $(1 \leq i \leq n)$ tel que $s(i) \neq s'(i)$ et
- pour tout $j \in \{1...n\}, j \neq i, s(j) = s'(j)$

Dans ce voisinage, un voisin de *s* peut être obtenu par le simple changement de la valeur courante d'une variable quelconque dans *s*. Nous appelons ce voisinage « 1-changement ». Avec ce voisinage, *s* possède exactement $\Sigma(|Di|-1)$ $(1 \leq i \leq n)$ voisins.

La plupart des méthodes de voisinage pour les PASC, souvent appelées en IA les *méthodes de réparation,* sont basées sur ce voisinage.

Les méthodes de voisinage, telles que nous venons de les présenter, constituent une approche définie de manière extrêmement générale. Dans la suite, nous présentons un ensemble de métaheuristiques fondées sur la notion de voisinage, notamment le recuit simulé, les méthodes d'acceptation avec seuil, le bruitage, la recherche tabou et GRASP. Nous terminons cette partie en évoquant l'application de méthodes de voisinage aux problèmes PASC et en évoquant deux méthodes de résolution spécifiques comme la technique de min-conflits et la pondération.

4.1.1. *Le recuit simulé (Simulated Annealing)*

La méthode de recuit simulé [KIR 83, CER 85] s'inspire du processus de recuit physique. Ce processus utilisé en métallurgie pour améliorer la qualité d'un solide cherche un état d'énergie minimale qui correspond à une structure stable du solide. En partant d'une haute température à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température. Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermodynamique. Quand la température tend vers zéro, seules les transitions d'un état à un état d'énergie plus faible sont possibles.

Les origines du recuit simulé remontent aux expériences réalisées par Metropolis et al. dans les années 50 pour simuler l'évolution d'un tel processus de recuit physique [MET 53]. Metropolis et al. utilisent une méthode stochastique pour générer une suite d'états successifs du système en partant d'un état initial donné. Tout nouvel état est obtenu en faisant subir un déplacement (une perturbation) aléatoire à un atome quelconque. Soit ΔE la différence d'énergie occasionnée par une telle perturbation. Le nouvel état est accepté si l'énergie du système diminue ($\Delta E \leq 0$). Sinon, il est accepté avec une probabilité définie par : $p(\Delta E,T) = \exp(-\Delta E/(C_b \times T))$ où T est la température du système et C_b une constante physique connue sous le nom de *constante de Boltzmann*.

A chaque étape, l'acceptation ou non d'un nouvel état dont l'énergie est supérieure à celle de l'état courant est déterminée de manière probabiliste : un réel $0 \leq \theta < 1$ est tiré aléatoirement et ensuite comparé avec $p(\Delta E,T)$. Si $\theta \leq p(\Delta E,T)$, alors le nouvel état est accepté pour remplacer l'état courant, sinon, l'état courant est maintenu. Après un grand nombre de perturbations, un tel processus fait évoluer le système vers un état d'équilibre thermodynamique selon la *distribution de Boltzmann* qui est définie par la probabilité de se trouver dans un état d'énergie E : $Pr(E) = c(T) \times \exp(-E/(C_b \times T))$ où c(T) est un facteur de normalisation.

L'utilisation d'un tel processus du recuit simulé pour résoudre des problèmes d'optimisation combinatoire a été reportée dans [KIR 83, CER 85]. Le recuit simulé peut être vu comme une version étendue de la méthode de descente. Le processus du recuit simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui dégradent la fonction de coût. A chaque nouvelle itération, un voisin $s' \in N(s)$ de la configuration courante *s* est généré de manière aléatoire. Selon les cas, ce voisin sera soit retenu pour remplacer celle-ci, soit rejeté. Si ce voisin est de performance supérieure ou égale à celle de la configuration courante, *i.e.*, $f(s') \leq f(s)$, il est systématiquement retenu. Dans le cas contraire, *s'* est accepté avec une probabilité $p(\Delta f,T)$ qui dépend de deux facteurs : d'une part l'importance de la dégradation $\Delta f = f(s') - f(s)$ (les dégradations plus faibles sont plus facilement acceptées), d'autre part un paramètre de contrôle T, la température (une température élevée correspond à une probabilité plus grande d'accepter des dégradations). La température est contrôlée par une fonction décroissante qui définit un schéma de refroidissement. Les deux paramètres de la méthode définissent la longueur des paliers et la fonction permettant de calculer la suite décroissante des températures. En pratique, l'algorithme s'arrête et retourne la meilleure configuration trouvée lorsque aucune configuration voisine n'a été acceptée pendant un certain nombre d'itérations à une température ou lorsque la température atteint la valeur zéro.

La performance du recuit simulé dépend largement du schéma de refroidissement utilisé. De nombreux schémas théoriques et pratiques ont été proposés. De manière générale, les schémas de refroidissement connus peuvent être classés en trois catégories :

- réduction par paliers : chaque température est maintenue égale pendant un certain nombre d'itérations, et décroît ainsi par paliers.
- réduction continue [LUN 86] : la température est modifiée à chaque itération.
- réduction non-monotone [CON 90] : la température décroît à chaque itération avec des augmentations occasionnelles.

Il existe des schémas qui garantissent la convergence asymptotique du recuit simulé [HAJ 88]. En pratique, on utilise des schémas relativement simples même s'ils ne garantissent pas la convergence de l'algorithme vers une solution optimale.

Le recuit simulé constitue, parmi les méthodes de voisinage, l'une des plus anciennes et des plus populaires. Il a acquis son succès essentiellement grâce à des résultats pratiques obtenus sur de nombreux problèmes NP-difficiles. Des exemples de ces applications sont présentés dans [BON 84, VAN 87, COL 88, AAR 89, VID 93, KOU 94]. La preuve de convergence a également contribué à cette popularité, bien que cette preuve n'ait pas de portée en pratique.

4.1.2. *Les algorithmes d'acceptation avec seuil (Threshold algorithms)*

Ces algorithmes sont des variantes du recuit simulé [DUC 90] : la différence se situe au niveau de l'acceptation de dégradation à chaque étape. Dans le RS, cette décision est prise selon le critère de Metropolis. Dans un algorithme d'acceptation avec seuil, une telle décision est prise de manière déterministe. A chaque itération *k*, l'acceptation d'un voisin $s' \in N(s)$ se base uniquement sur une fonction auxiliaire r(s',s) et un seuil T_k : s' est accepté si $r(s',s) < T_k$. La fonction r(s',s) et le seuil T_k peuvent être définis de nombreuses manières. Dans le cas le plus simple, la fonction r(s',s) est définie par $\Delta f = f(s') - f(s)$. Le paramètre de seuil est défini de manière analogue à la température T du recuit simulé : il est initié à une valeur élevée puis décroît progressivement à chaque fois qu'un nombre prédéterminé d'itérations est effectué. Les seuils ainsi générés correspondent à une suite de valeurs positives décroissantes $T_1 \geq T_2 \geq ... \geq T_{k-1} \geq T_k \geq 0$ et $T_k \rightarrow 0$. L'idée est de diminuer petit à petit la chance d'accepter des configurations qui dégradent la fonction de coût. Quand T_k tend vers 0, l'algorithme réalise une recherche de descente aléatoire.

Ces algorithmes ont été utilisés pour résoudre des problèmes d'optimisation et ont obtenu des résultats intéressants [DUC 90, DUC 93, SIN 93]. La difficulté essentielle de cette approche se situe au niveau de la détermination des seuils pour une application donnée.

4.1.3. *Méthode du bruitage*

La méthode de bruitage a été introduite sur des problèmes dont la donnée comporte un ensemble de nombres réels [CHA 93], par exemple, le voyageur de commerce (arêtes valuées par des réels) et le problème d'affectation quadratique (matrices de réels). Cette méthode fait appel à une notion de bruitage de la donnée qui est définie de la façon suivante : la donnée bruitée est produite à partir de la donnée initiale en ajoutant à chacun des réels une composante calculée comme le produit de trois éléments : a) une fonction aléatoire à valeurs sur l'intervalle [0,1], b) un paramètre permettant de contrôler le niveau du bruit, c) le plus grand des réels concernés, afin de normaliser le niveau du bruit par rapport à la donnée.

L'exécution de l'heuristique comporte plusieurs étapes. Chaque étape consiste à calculer un bruitage de la donnée, puis à effectuer une descente prenant en compte la fonction de coût calculée à partir de la donnée bruitée. Le niveau du bruit est décrémenté au début de chaque nouvelle étape, et la descente s'effectue à partir de la configuration résultant de l'étape précédente. Il existe deux variantes pour le bruitage : 1) chaque descente sur la donnée bruitée est suivie par une descente effectuée sur la donnée non bruitée. L'objectif consiste à mieux tenir compte de la donnée réelle puisqu'un véritable optimum local est alors atteint ; 2) la configuration courante est régulièrement remplacée par la meilleure configuration obtenue depuis le début.

Cette méthode a été appliquée avec succès au problème du voyageur de commerce et a obtenu de très bons résultats selon les auteurs [CHA 93, 95].

4.1.4. *GRASP*

La méthode GRASP (pour *Greedy Randomized Adaptive Search Procedure*) est une procédure itérative introduite par Feo et Resende [FEO 89, FEO 95]. A proprement parler, GRASP est une méthode hybride car elle cherche à combiner les avantages des heuristiques gloutonnes, de la recherche aléatoire et des méthodes de voisinage. Un algorithme GRASP répète un processus composé de deux étapes : la construction d'une solution suivie par une descente pour améliorer la solution construite.

Durant l'étape de construction, une solution est itérativement construite : chaque itération ajoute un élément dans la solution partielle courante (cf. § 3.2.). Pour déterminer l'élément qui sera ajouté, on utilise une liste des meilleurs candidats obtenus avec une fonction gloutonne et on prend *au hasard* un élément dans cette liste. La liste des meilleurs candidats est dynamiquement mise à jour après chaque itération de construction. Cette étape de construction continue jusqu'à ce qu'une solution complète soit obtenue.

A partir de cette solution, une descente est appliquée pour améliorer la solution. Une procédure GRASP répète ces deux étapes et retourne à la fin la meilleure solution trouvée. Les deux paramètres de cette méthodes sont donc la longueur de la liste de candidats et le nombre d'itérations autorisées.

Cette méthode a été appliquée avec succès à plusieurs problèmes d'optimisation [FEO 94, LAG 94, KON 95].

4.1.5. *La recherche tabou*

La méthode tabou a été développée par Glover [GLO 86] et indépendamment par Hansen [HAN 86]. Cette méthode fait appel à un ensemble de règles et de mécanismes généraux pour guider la recherche de manière intelligente au travers de l'espace des solutions.

A l'inverse du recuit simulé qui génère de manière aléatoire une seule solution voisine $s' \in N(s)$ à chaque itération, tabou examine un échantillonnage de solutions de $N(s)$ et retient la meilleure s' même si s' est plus mauvaise que s. La recherche tabou ne s'arrête donc pas au premier optimum trouvé. Cependant, cette stratégie peut entraîner des cycles, par exemple un cycle de longueur 2 : $s \rightarrow s' \rightarrow s \rightarrow s'$... Pour empêcher ce type de cycle, on mémorise les k dernières configurations visitées dans une mémoire à court terme et on interdit tout mouvement qui conduit à une de ces configurations. Cette mémoire est appelée la *liste tabou*, une des composantes essentielles de cette méthode. Elle permet d'éviter tous les cycles de longueur inférieure ou égale à k. La valeur de k dépend du problème à résoudre et peut éventuellement évoluer au cours de la recherche.

La mémorisation de configurations entières serait trop coûteuse en temps de calcul et en place mémoire et ne serait sans doute pas la plus efficace. En fait, la liste tabou mémorise des *caractéristiques* des configurations au lieu de configurations complètes. Plus précisément, lorsqu'un mouvement vient d'être effectué, c'est généralement la caractéristique perdue par la configuration courante qui devient tabou. Par exemple, avec la structure vectorielle de configuration définie précédemment pour les problèmes d'affectation sous contraintes, un type de caractéristique très simple est le couple $<V,v>$, V et v étant respectivement une variable et une valeur possible pour la variable. Supposons qu'un mouvement qui affecte la valeur v à la variable V en remplacement de la valeur $v0$ soit effectué, la caractéristique $<V,v0>$ est alors mémorisée et interdite pour les k itérations suivantes. Autrement dit, la variable V qui vient d'abandonner la valeur $v0$ ne pourra pas reprendre cette valeur pendant cette durée. Le résultat est que non seulement la configuration courante ne pourra pas réapparaître lors des k prochaines itérations, mais que de nombreuses autres configurations seront également interdites.

Lorsque les listes tabou font intervenir des caractéristiques de modifications, les interdictions qu'elles engendrent peuvent s'avérer trop fortes et restreindre l'ensemble des solutions admises à chaque itération d'une manière jugée trop brutale. Un mécanisme particulier, appelé *l'aspiration*, est mis en place afin de pallier cet inconvénient. Ce mécanisme permet de lever le statut tabou d'une configuration, sans pour autant introduire un risque de cycles dans le processus de recherche. La fonction d'aspiration peut être définie de plusieurs manières. La fonction la plus simple consiste à révoquer le statut tabou d'un mouvement si ce dernier permet d'atteindre une solution de qualité supérieure à celle de la meilleure solution trouvée jusqu'alors.

Il existe d'autres techniques intéressantes pour améliorer la puissance de la méthode tabou, en particulier, *l'intensification* et *la diversification*. Toutes les deux se basent sur l'utilisation d'une mémoire à long terme et se différencient selon la façon d'exploiter les informations de cette mémoire.

L'intensification se fonde sur l'idée d'apprentissage de propriétés favorables : les propriétés communes souvent rencontrées dans les meilleures configurations visitées sont mémorisées au cours de la recherche, puis favorisées pendant la période d'intensification. Une autre manière d'appliquer l'intensification consiste à mémoriser une liste de solutions de bonne qualité et à retourner vers une des ces solutions.

La diversification a un objectif inverse de l'intensification : elle cherche à diriger la recherche vers des zones inexplorées. Sa mise en oeuvre consiste souvent à modifier temporairement la fonction de coût pour favoriser des mouvements n'ayant pas été effectués ou à pénaliser les mouvements ayant été souvent répétés. L'intensification et la diversification jouent donc un rôle complémentaire.

Comme pour la plupart des méthodes heuristiques, il n'existe pas de résultats théoriques garantissant la convergence d'une procédure tabou vers un optimum global. La raison principale de cet état de fait tient à la nature même de la méthode. Celle-ci étant hautement adaptative et modulable, son analyse par les outils mathématiques est rendue plus difficile. Le seul résultat théorique connu à ce jour concerne une version probabiliste de la méthode tabou qui s'apparente à la méthode du RS [FAI 92]. Cependant, la méthode tabou suscite un intérêt toujours croissant depuis sa découverte et de nombreux raffinements ont été introduits dans la méthode [GLO 89, 93a, 93b, 95, 97]. Des résultats pratiques très intéressants ont été obtenus pour de nombreux problèmes d'optimisation combinatoire, et en particulier, pour des problèmes d'affectation sous contraintes. Nous pensons que la recherche tabou fait partie des meilleures métaheuristiques pour ces problèmes.

Cette méthode se caractérise par sa stratégie agressive de recherche (choix d'un des meilleurs mouvements à chaque itération) combinée avec différentes possibilités permettant de s'adapter au problème et d'intégrer des connaissances spécifiques. Cela demande naturellement un effort particulier d'adaptation de la méthode.

4.1.6. *Résolution des problèmes PASC par les méthodes de voisinage*

Les problèmes PASC peuvent être traités par les méthodes de voisinage en utilisant le voisinage présenté dans l'introduction de § 4.1. Dans le cas de MCSP, la fonction de coût est définie comme le nombre de contraintes violées par une configuration. Le problème CSP peut être traité comme MCSP mais en recherchant une configuration de coût nul[6].

De nombreux travaux ont été consacrés en IA à la résolution du problème CSP par des méthodes de voisinage. Des études parallèles ont également été consacrées au problème SAT (un CSP particulier). Dans certaines de ces études, les métaheuristiques comme le recuit simulé ou la méthode tabou ont été utilisées [HAN 90, SPE 96, BAT 97, GAL 97]. Des méthodes de résolution spécifiques, dites méthodes de réparation, ont également été introduites. Ces dernières ne sont pas des métaheuristiques à proprement parler. Néanmoins, ces méthodes introduisent des mécanismes généraux applicables à de nombreux problèmes PASC. C'est pour cette raison que nous les présentons ci-dessous.

La méthode de min-conflits a été introduite pour des problèmes de satisfaction de contraintes [MIN 90, 92]. A chaque itération, on commence par choisir aléatoirement une variable en conflit, c'est-à-dire une variable impliquée dans une contrainte non satisfaite. Pour cette variable, on choisit alors la valeur qui minimise le nombre de conflits, avec choix aléatoire en cas d'égalité. Comme aucune

---

[6] Ceci est vrai pour les problèmes ne comprenant que des contraintes simples comme dans les CSP binaires ou dans SAT. Une généralisation de ce type d'approche en présence de contraintes plus complexes est proposée dans [GAL 99b].

dégradation du coût n'est acceptée, il s'agit donc d'une descente avec un voisinage restreint. Minton et al., ont appliqué cette technique à un problème d'ordonnancement sous contraintes [MIN 92].

GSAT (pour *Greedy SATisfiability*) est une méthode d'amélioration itérative présentée comme une application du principe de min-conflits au problème SAT [SEL 92]. La procédure GSAT est basée sur la minimisation du nombre de clauses non satisfaites. Plus précisément, une itération GSAT consiste à 1) examiner toutes les variables en conflit, *i.e.,* impliquées dans une clause non satisfaite, 2) choisir celle dont l'inversion de la valeur minimise le nombre de clauses non satisfaites et 3) inverser la valeur de cette variable.

Plusieurs variantes ont été proposées pour améliorer min-conflits et GSAT dans le but de permettre de ressortir des optima locaux. Le principe de la variante « *random-walk* » est basé sur l'introduction de mouvements aléatoires : à chaque nouvelle itération, on effectue un mouvement aléatoire avec une probabilité fixée p et un mouvement de descente avec une probabilité 1-p. Dans GSAT, un mouvement aléatoire consiste simplement à choisir aléatoirement une variable en conflit et à inverser sa valeur [SEL 93]. Pour le problème CSP, un mouvement aléatoire consiste à choisir aléatoirement une variable en conflit, puis à choisir aléatoirement une nouvelle valeur pour cette variable [WAL 96, HAO 96]. On peut noter que cette stratégie de mouvements aléatoires est généralisable à d'autres types de problèmes.

Une autre variante importante est la méthode de pondération [MOR 93], appelée option « *weighted clauses* » dans GSAT [SEL 93]. L'idée consiste à introduire un mécanisme permettant de ressortir des optima locaux en modifiant dynamiquement la fonction de coût. La fonction de coût correspond initialement au nombre de contraintes violées. Lorsqu'un optimum local est atteint, cette fonction est modifiée en incrémentant la pondération des contraintes violées par l'optimum local.

Il existe également d'autres méthodes pour les CSP. Citons simplement les suivantes : EFLOP (pour *Escaping From Local Optima by Propagation*) qui est un type de mouvement spécifique permettant de ressortir des optima locaux dans le cadre des problèmes sous contraintes [YUG 94] et GLS (pour *Guided Local Search*) qui combine la descente avec un mécanisme spécifique pour modifier la fonction de coût de manière dynamique [VOU 95].

Ces différentes options permettent d'améliorer très sensiblement les performances de la méthode de min-conflits ou de GSAT de base.

GSAT et ses variantes ont connu des succès remarquables pour la recherche de modèles d'instances difficiles de SAT. A titre indicatif, ces procédures sont capables de traiter des instances satisfiables de 3000 variables au seuil alors que les meilleures méthodes classiques ne dépassent pas 500 variables. Dans [WAL 96], plusieurs méthodes de réparation sont comparées sur des instances aléatoires de MCSP binaires. La méthode de min-conflits avec l'option *random walk* obtient les meilleurs résultats. Cette méthode est cependant nettement dominée sur ce type d'instances par un algorithme tabou de base [GAL 97].

4.2. *Algorithmes évolutifs*

Le terme « algorithmes évolutifs » englobe une autre classe assez large de métaheuristiques. Ces algorithmes sont basés sur le principe du processus d'évolution naturelle [DEJ 93, BÄC 93a, SCH 97b]. Les algorithmes évolutifs doivent leur nom à l'analogie avec les mécanismes d'évolution des espèces vivantes.

Un algorithme évolutif typique est composé de trois éléments essentiels : 1) une *population* constituée de plusieurs individus représentant des solutions potentielles (configurations) du problème donné; 2) un *mécanisme d'évaluation* de l'adaptation de chaque individu de la population à l'égard de son environnement extérieur; 3) un *mécanisme d'évolution* composé d'opérateurs permettant d'éliminer certains individus et de produire de nouveaux individus à partir des individus sélectionnés.

Du point de vue opérationnel, un algorithme évolutif typique débute avec une population initiale souvent générée aléatoirement et répète ensuite un cycle d'évolution composé de 3 étapes séquentielles : 1) mesurer l'adaptation (la qualité) de chaque individu de la population par le mécanisme d'évaluation, 2) sélectionner une partie des individus, et 3) produire de nouveaux individus par des recombinaisons d'individus sélectionnés. Ce processus se termine quand la condition d'arrêt est vérifiée, par exemple, quand un nombre maximum de cycles (générations) ou un nombre maximum d'évaluations est atteint. Selon l'analogie de l'évolution naturelle, la qualité des individus de la population devrait tendre à s'améliorer au fur et à mesure du processus.

Parmi les composantes d'un algorithme évolutif, l'individu et la fonction d'adaptation correspondent respectivement à la notion de configuration et à la fonction d'évaluation dans les méthodes de voisinage. La notion de mécanisme d'évolution est proche de celle du mécanisme de parcours du voisinage ; en particulier, on peut interpréter une itération d'une méthode de voisinage comme un mécanisme d'évolution portant sur une population réduite à un seul individu. Cependant, les opérateurs sont sensiblement différents. En effet, un algorithme évolutif comporte un ensemble d'opérateurs tels que la sélection, la mutation et éventuellement le croisement.

La *sélection* a pour objectif de choisir les individus qui vont pouvoir survivre et/ou se reproduire pour transmettre leurs caractéristiques à la génération suivante. La sélection se base généralement sur le principe de conservation des individus les mieux adaptés et d'élimination des moins adaptés. Le *croisement* ou recombinaison cherche à combiner les caractéristiques des individus parents pour créer des individus enfants avec de nouvelles potentialités dans la génération future. La *mutation* effectue de légères modifications de certains individus.

Les algorithmes évolutifs ont été appliqués à de très nombreux problèmes complexes et difficiles, y compris des problèmes d'optimisation (continue ou combinatoire). [BÄC 93b] collecte un nombre important d'exemples d'applications.

D'une manière générale, on peut distinguer trois grandes familles d'algorithmes évolutifs : les algorithmes génétiques, la programmation évolutive et les stratégies d'évolution. Ces méthodes se différencient par leur manière de représenter les données et par leur façon de faire évoluer la population d'une génération à l'autre.

4.2.1. *Algorithmes génétiques*

Les algorithmes génétiques classiques introduits par Holland s'appuient fortement sur un *codage universel* sous forme de chaînes 0/1 de longueur fixe et un ensemble d'opérateurs *génétiques* : la mutation, l'inversion et le croisement [HOL 75, 92, GOL 89, DEJ 94]. Un individu sous ce codage, appelé un chromosome, représente une configuration du problème. Les opérateurs « génétiques » sont définis de manière à opérer aléatoirement sur un ou deux individus sans aucune connaissance sur le problème.

Le *croisement* permet de produire deux nouvelles individus (enfants) à partir de deux individus (parents). Par exemple, le croisement bi-points consiste à choisir aléatoirement deux points de croisement et à échanger les segments des deux parents déterminés par ces deux points. Le croisement réalise donc uniquement des recombinaisons de valeurs (gènes) existantes entre deux parents et ne permet pas d'introduire de nouvelles valeurs dans les individus enfants. Pour cela, on applique la mutation. L'opérateur de *mutation* consiste à changer aléatoirement la valeur de certaines variables dans une individu. Dans les algorithmes génétiques la mutation est considérée comme un opérateur secondaire par rapport au croisement. Un cycle d'évolution complet d'un algorithme génétique est formé par l'application des opérateurs de sélection, croisement et mutation sur une population d'individus.

Bien que les algorithmes génétiques soient considérés aujourd'hui comme une méthode d'optimisation, l'objectif initial consistait à concevoir des systèmes d'apprentissage généraux, robustes et adaptatifs, applicables à une large classe de problèmes. En particulier, Holland s'est intéressé à l'élaboration d'une technique de programmation permettant l'évolution des programmes informatiques par reproduction, croisement et mutation. Cette motivation est à l'origine de l'universalité des algorithmes génétiques : ni le codage ni les opérateurs génétiques ne demandent des connaissances spécifiques du problème. Selon ce principe, pour un problème donné, il suffit de trouver une transformation des paramètres du problème en chaînes 0/1 et d'appliquer les opérateurs génétiques sur une population de solutions potentielles ainsi codées. C'est grâce à cette universalité que Holland a pu réaliser des analyses théoriques sur les algorithmes génétiques conduisant à la théorie des schémas et à la caractérisation du rôle et de l'importance du croisement. De plus, des résultats théoriques ont été obtenus sur la convergence des algorithmes génétiques standard [CER 94].

L'universalité d'un tel algorithme pose évidemment des problèmes d'efficacité en pratique. En effet, en tant que méthode d'optimisation, un algorithme génétique classique se base uniquement sur des opérateurs « aveugles » et est donc rarement en mesure de produire des résultats comparables à ceux d'une méthode de voisinage. Une technique pour remédier à ce problème consiste à *spécialiser* l'algorithme génétique au problème donnée. Plus précisément, à la place des opérateurs aléatoires, la mutation et le croisement sont adaptés en se basant sur des connaissances spécifiques du problème. De cette manière, la recherche est mieux guidée et donc plus efficace. Il est désormais établi que pour être efficace en optimisation, il est indispensable d'intégrer des connaissances du problème [GRE 87, DAV 91]. Une autre voie intéressante pour améliorer l'efficacité des algorithmes génétiques consiste à combiner le cadre génétique avec d'autres méthodes de résolution. Ce point sera développé ultérieurement dans la section sur les méthodes hybrides.

Les algorithmes génétiques spécialisés ou hybrides ont été appliqués à de nombreuses applications dans des domaines très variés [BÄC 93b]. Quant aux algorithmes génétiques purs, leurs résultats en optimisation combinatoire sont en général faibles.

4.2.2. *Programmation évolutive*

La programmation évolutive s'appuie sur un codage approprié du problème à résoudre et sur les opérations de mutation adaptées au codage [FOG 66, 94]. Le codage d'un tel algorithme dépend du problème à résoudre. Par exemple, pour un problème d'optimisation dans le domaine des réels, les individus d'une population seraient des vecteurs de réels. De manière similaire, une permutation serait utilisée pour représenter un tour dans le problème du voyageur de commerce. A partir d'un codage donné pour un problème, la mutation ou opérateur d'évolution spécifique sera définie. Par exemple, dans le cas du problème du voyageur de commerce, la mutation basée sur le codage de permutation pourrait être l'opération d'inversion : prendre deux villes dans le tour et inverser l'ordre du segment défini par les deux villes. Ainsi l'analogie est forte avec les méthodes de voisinage : une mutation correspond à un mouvement dans un algorithme de voisinage.

Un cycle d'évolution typique pour la programmation évolutive est le suivant : chaque configuration de la population courante est copiée dans une nouvelle population. Les configurations sont ensuite mutées, conduisant à de nouvelles configurations. L'ensemble des configurations entre ensuite dans une étape de compétition pour survivre dans la génération suivante.

La programmation évolutive a été initialement introduite pour simuler l'intelligence qui est définie sur l'hypothèse suivante : la caractéristique principale de l'intelligence est la capacité d'adaptation comportementale d'un organisme à son environnement [FOG 66]. Selon un modèle très simpliste, cette tâche de simulation revient à prédire une séquence de symboles appartenant à un alphabet fini à partir des séquences déjà observées. Dans ce but, des automates d'états finis sont choisis pour représenter les individus d'une population. Ainsi, à chaque automate de la population est donnée une série de symboles pris dans une séquence déjà observée et la sortie de l'automate est mesurée par rapport au résultat déjà connu. Cette mesure constitue l'adaptation de l'individu. L'étape suivante consiste à créer, pour chaque individu, un individu enfant par une mutation aléatoire de l'individu parent. La mutation est assurée par une des 5 opérations suivantes : changer le symbole d'une sortie, changer un état de transition, ajouter ou supprimer un état et changer l'état initial.

Aujourd'hui, la programmation évolutive s'est adaptée à l'optimisation combinatoire et a produit des résultats intéressants pour certains problèmes [FOG 94, BÄC 93b].

4.2.3. *Stratégies d'évolution*

Les stratégies d'évolution [REC 73, SCH 81, BÄC 91, 95] sont conçues dès le départ pour résoudre des problèmes d'optimisation continus. Dans un algorithme SE, les individus sont des points (vecteurs de réels). Comme la programmation évolutive, les SE n'utilisent que la mutation et la sélection.

L'algorithme le plus simple, noté (1+1)-ES, manipule un seul individu. A chaque génération (itération), l'algorithme génère par mutation un individu enfant à partir de l'individu parent et sélectionne l'un ou l'autre pour le conserver dans la population (selon l'adaptation de chaque individu). Le processus s'arrête quand la condition d'arrêt est vérifiée, définie souvent par le nombre d'itérations, le temps de calcul réalisé ou l'écart entre deux individus de deux itérations successives. La *mutation* dans un tel algorithme est aléatoirement appliquée à tous les composants de l'individu pour produire un enfant. Cette mutation fonctionne selon les principes suivants : un enfant ressemble à ses parents, et plus (moins) un changement est important, moins (plus) la fréquence de changement est élevée.

Cet algorithme (1+1)-ES se généralise en un algorithme (m+l)-ES qui signifie que m parents génèrent l enfants à chaque génération et qu'une sélection ramène ensuite la population de m+l individus à m individus [BÄC 91]. De plus, la recombinaison a été également introduite dans ces algorithmes [BÄC 93a].

Les SE ont été adaptées à l'optimisation combinatoire et appliquées à de nombreux problèmes de référence et pratiques, voir [BÄC 93b] pour une liste d'applications.

4.3. *Méthodes hybrides et autres*

Le mode d'hybridation qui semble le plus fécond concerne la combinaison entre les méthodes de voisinage et l'approche d'évolution. L'idée essentielle de cette hybridation consiste à exploiter pleinement la puissance de recherche de méthodes de voisinage et de recombinaison des algorithmes évolutifs sur une population de solutions[7]. Un tel algorithme utilise une ou plusieurs méthodes de voisinage sur les individus de la population pendant un certain nombre d'itérations ou jusqu'à la découverte d'un ensemble d'optima locaux et invoque ensuite un mécanisme de recombinaison pour créer de nouveaux individus. Comme pour les algorithmes génétiques spécialisés, la recombinaison doit impérativement être adaptée au problème traité.

Cette approche a permis de produire d'excellents voire les meilleurs résultats sur des *benchmarks* réputés de problèmes de référence. C'est le cas par exemple du problème du voyageur de commerce TSP [JOG 87, MÜH 88, JOG 91, ULD 91, FRE 96], de la coloration [COS 95, FLE 96b, DOR 98, GAL 99a], de la clique maximale [BAL 98], de l'affectation quadratique [MOS 93, GLO 95, MER 97, BAC 98], ou du problème de *bin-packing* [FAL 96].

Les algorithmes hybrides sont sans doute parmi les méthodes les plus puissantes. Malheureusement, les temps de calcul nécessaires peuvent devenir prohibitifs à cause du nombre d'individus manipulés dans la population. Une voie pour résoudre ce problème est la parallélisation de ces algorithmes sur des machines parallèles ou sur des systèmes distribués [VER 95, TAL 98].

Nous venons d'évoquer l'hybridation d'un algorithme évolutionniste avec des méthodes de voisinage. Il est également possible d'hybrider d'autres types d'approches, par exemple une heuristique gloutonne et une méthode de voisinage ou un algorithme évolutif. Un exemple typique est la méthode GRASP.

Avant de terminer la présentation sur les métaheuristiques, notons enfin qu'il existe d'autres métaheuristiques intéressantes. Citons ici deux entre elles : la méthode « *Scatter Search* » [GLO 97, GLO 98] et la méthode ACO (pour *ant colony optimization*) [DOR 99].

---

[7] Ce type de méthodes sont appelées aussi « *memetic algorithms* » [MOS 89].

La méthode « *Scatter Search* » consiste à : 1) construire un ensemble de solutions de bonne qualité, 2) sélectionner des sous-ensembles de ces solutions et 3) appliquer un opérateur de recombinaison à chacun de ces sous-ensembles. On retrouve donc dans cette méthode certaines des idées des algorithmes génétiques.

La méthode de colonie de fourmis ACO s'inspire du comportement des colonies de fourmis réelles. La méthode se caractérise par la combinaison d'une approche de construction (§ 3.2.) et des mécanismes d'apprentissage fondés sur mémorisation. Le principe de cette méthode est le suivant.

Malgré la vision très limitée de chaque fourmi, une colonie de fourmis parvient a minimiser la longueur du chemin conduisant à une source de nourriture, grâce aux traces chimiques (phéromones) laissées par chacune des fourmis. Un principe analogue a été utilisé pour traiter le problème du voyageur de commerce ainsi que d'autres problèmes d'optimisation. La méthode consiste à réitérer un algorithme de construction (assimilé à l'action d'une fourmi) dans lequel chacun des choix est déterminé en tenant compte à la fois 1) d'un critère glouton, 2) d'une part d'aléatoire et 3) des traces laissées par les fourmis précédentes. Dans le cas du voyageur de commerce, une fourmi qui a emprunté une arête incite les fourmis suivantes à emprunter cette même arête à leur tour. Il faut noter que les colonies de fourmis sont souvent hybridées avec la recherche locale.

4.4. *Le point de vue de la complexité*

En ce qui concerne les algorithmes exacts, la théorie de la complexité est un outil largement utilisé pour caractériser le comportement de l'algorithme, *i.e.* le temps de calcul sur l'ensemble des instances. La théorie de la NP-complétude ne permet pas directement la classification des problèmes d'optimisation. En effet, on observe en pratique qu'il existe des problèmes NP-complets (problème du sac à dos, partition d'un ensemble d'entiers...) plus faciles à résoudre que d'autres.

Cette remarque a justifié de nombreuses extensions de la théorie de la NP-complétude.

La notion de pseudo-polynomialité [GAR 79] a permis de séparer les problèmes du type sac à dos et partition des autres problèmes NP-complets. Cependant cette classe des problèmes pseudo-polynomiaux est très restreinte.

L'étude de l'existence pour un problème donné de schémas polynomiaux d'approximation a permis une meilleure connaissance des problèmes NP-complets [MOT 95, CRE 96].

La définition de la classe PLS (*Polynomial Local Search*) [JOH 88, YAN 90, YAN 95, SCH 91] développée spécifiquement afin de mieux comprendre les méthodes de recherche locale. Cette théorie permet de construire des réductions entre des problèmes de recherche locale. Un tel problème, défini à l'aide d'un problème d'optimisation et d'une fonction de voisinage, consiste simplement ici à rechercher un optimum local. La plupart des problèmes classiques munis de leur fonction de voisinage standard, comme par exemple le problème de partition de graphe avec le voisinage de Kernighan-Lin, sont PLS-complets (c'est-à-dire que sur chacun d'entre eux, on peut polynomialement simuler les recherches locales des autres). Ces problèmes sont équivalents sur le plan de la recherche locale.

D'autres auteurs se sont intéressés à la structure combinatoire des solutions d'un problème d'optimisation, tentant de mesurer la distance d'un optimum local à un optimum global [KER 93, STE 90].

Outre la théorie de la complexité, il existe des preuves de convergence, concernant le recuit simulé, les algorithmes génétiques. Ces preuves sont fondées sur des hypothèses rarement réalisables, ce qui limite largement leur utilité pratique.

Pour les problèmes de satisfaction (par exemple le problème CSP), il existe par ailleurs pour les instances aléatoires des résultats intéressants concernant la transition de phase (phénomène de seuil) [HUB 96]. La plupart des problèmes difficiles sont situés dans la zone de transition de phase. Lorsqu'on s'éloigne de part et d'autre de la zone du seuil, la plupart des instances se révèlent faciles : les problèmes sous-contraints tendent à être facilement prouvés satisfiables car ils possèdent généralement beaucoup de solutions et les problèmes sur-contraints tendent à être facilement prouvés insatisfiables. Signalons enfin que dans la zone « facile » (« difficile »), il existe également des instances difficiles (faciles).

5. Analyse des métaheuristiques

Dans cette section, nous proposons quelques éléments d'analyse des métaheuristiques, puis nous comparons les performances sur deux problèmes de référence. Enfin, nous donnons quelques indications pratiques pour faciliter le choix d'une métaheuristique.

5.1. *Exploitation et exploration*

Les inventeurs des algorithmes génétiques ont introduit les notions de l'exploitation et de l'exploration. L'exploitation insiste sur la capacité d'examiner en profondeur par une méthode des zones de recherche particulières alors que l'exploration met en avant la capacité de découvrir des zones de recherche prometteuses. Ces deux notions complémentations concernent au fait l'ensemble des méthodes de recherche. Il est donc pertinent d'analyser l'ensemble des métaheuristiques en fonction de ces deux notions.

Les méthodes de voisinage entendent exploiter les bonnes propriétés de la fonction de voisinage pour découvrir rapidement des configurations de bonne qualité. Elles reposent sur l'hypothèse que les zones les plus prometteuses sont situées à proximité des configurations (déjà visitées) qui sont les plus performantes. Le principe d'exploitation consiste à examiner en priorité ces zones.

Dans les algorithmes génétiques, la sélection a pour effet de concentrer la recherche autour des configurations de meilleure performance. Dans les méthodes de voisinage, l'exploitation est principalement assurée par la préférence accordée à une configuration de bonne performance dans la procédure de réparation. Plusieurs méthodes introduisent des mécanismes spécifiques supplémentaires d'exploitation : c'est le cas de l'intensification dans le méthode tabou.

Cependant, l'application systématique du seul principe d'exploitation ne permet pas une recherche efficace. En effet, l'exploitation conduit à confiner la recherche dans une zone limitée qui finit par s'épuiser. Le cas de l'amélioration itérative rapidement piégée dans un optimum local illustre cruellement ce phénomène. Une autre illustration souvent évoquée est fournie par le problème de convergence prématurée des algorithmes génétiques : du fait de la sélection, la population finit par n'être constituée que d'individus tous similaires. L'une des préoccupations majeures dans les algorithmes génétiques consiste d'ailleurs à préserver le plus longtemps possible une diversité suffisante dans la population. Face à ce type de difficulté, la solution consiste à diriger la poursuite de la recherche vers de nouvelles zones, *i.e.*, à recourir à l'exploration.

En plus de la relance, les heuristiques emploient principalement deux autres stratégies dans le but d'explorer : la première consiste à perturber aléatoirement et la seconde à caractériser les régions visitées pour pouvoir ensuite s'en éloigner.

La première stratégie qui est aussi la plus simple consiste à introduire des perturbations aléatoires : c'est le cas pour les mutations aléatoires dans les algorithmes génétiques ainsi que pour la génération aléatoire d'un voisin dans le recuit simulé. Dans les deux cas, la configuration courante est altérée de manière aléatoire et un mécanisme d'acceptation est appliqué *a posteriori*. Ces deux méthodes admettent donc en permanence des perturbations aléatoires. Dans la méthode GSAT, le mécanisme de base repose au contraire sur une règle beaucoup plus déterministe (dite agressive) consistant à choisir systématiquement le meilleur voisin. Une manière efficace d'améliorer GSAT de base est l'option « *random walk* » qui consiste à alterner les mouvements de base et des mouvements totalement aléatoires.

La deuxième stratégie pour explorer consiste à mémoriser au cours de la recherche des caractéristiques des régions visitées et à introduire un mécanisme permettant de s'éloigner de ces zones. C'est ce que fait la méthode tabou avec la liste tabou (court terme) et avec la diversification (long terme). La recherche locale guidée reprend cette même idée. Citons également la pondération qui utilise une variante de cette idée. Nous pouvons remarquer que cette seconde stratégie nécessite l'utilisation d'une mémoire.

Notons enfin que la recombinaison est parfois présentée comme une manière supplémentaire de contribuer à l'exploration.

Nous venons de voir qu'il existe différents modes d'exploration. Les métaheuristiques se différencient également selon la manière dont elles font varier l'intensité de l'exploration au cours de la recherche. Le recuit simulé présente une manière spécifique de procéder : le niveau des perturbations aléatoires (responsables de l'exploration), lié au paramètre de température, est initialement fixé à un niveau élevé et abaissé progressivement au cours de la recherche. Le même principe de perturbations aléatoires décroissantes se retrouve dans les algorithmes d'acceptation avec seuil et, sous une autre forme, dans la méthode de bruitage. La méthode tabou illustre une manière très différente de doser l'exploration : un mécanisme particulier d'exploration (la diversification) succède régulièrement à de longues phases de recherche.

La recombinaison constitue un autre principe général qui complète l'exploitation et l'exploration. Elle consiste à construire de nouvelles configurations en combinant la structure de deux ou plusieurs bonnes configurations déjà trouvées. Cette idée issue des algorithmes génétiques (le croisement) a été introduite dans des méthodes de voisinage, par exemple dans GSAT et dans tabou (« *relinking paths* »). Des travaux récents ont montré qu'une recombinaison appropriée de solutions peut améliorer sensiblement l'efficacité de la recherche. Rappelons également que ce principe est aussi la base principale des algorithmes hybrides qui allient une méthode de voisinage et la recombinaison.

5.2. *Méthodes générales et méthodes spécifiques*

Contrairement aux algorithmes traditionnels dédiés à un problème spécifique, les métaheuristiques constituent des mécanismes très généraux qui peuvent être adaptés pour traiter de nombreux problèmes différents. Pour expliquer l'efficacité d'un algorithme spécifique, on invoque souvent le fait qu'il utilise des connaissances spécifiques du problème. Pour expliquer l'efficacité d'une métaheuristique, deux types d'arguments opposés sont généralement avancés.

Selon certains, des mécanismes généraux suffisamment puissants ont par eux-mêmes la faculté de mener efficacement la recherche sans disposer d'information spécifique du problème considéré (contexte de boîte noire) : c'était notamment le point de vue classique porté sur les algorithmes génétiques [GOL 89]. Malheureusement, de même qu'il ne peut pas exister de stratégie avantageuse dans un jeu de hasard comme la roulette, il existe également des limitations théoriques fondamentales qui ruinent les espoirs d'une méthode aveugle dans le cas le plus général. En fait, le théorème « *No Free Lunch (NFL)* » [WOL 95] montre que pour les problèmes de type boîte noire, toutes les méthodes sont équivalentes et font aussi bien, ou plutôt aussi mal, que l'énumération aléatoire[8].

Selon le point de vue opposé, la puissance d'une métaheuristique est d'abord liée à son aptitude à intégrer des connaissances spécifiques du problème. La connaissance du problème la plus fondamentale réside dans le codage du problème et dans le choix de la fonction de voisinage. Plusieurs auteurs insistent sur l'importance du codage du problème, voir par exemple [RAD 95]. Dans le cas du voyageur de commerce, plusieurs types de codages différents ont été proposés et conduisent à des performances très variées [MIC 92]. En général, il n'existe pas de codage universellement efficace. Un « bon » codage doit permettre de restreindre l'espace de recherche et d'intégrer des connaissances du problème.

Les métaheuristiques tentent également d'améliorer leur efficacité en incorporant des connaissances supplémentaires dans leurs opérateurs. Plus les opérateurs d'une méthode utilisent des connaissances spécifiques, plus la méthode dispose de moyens potentiels pour conduire efficacement la recherche. En contrepartie, l'intégration de ces connaissances spécifiques (en supposant que ces connaissances soient disponibles) nécessite un effort pour spécialiser ou adapter la méthode. La méthode tabou vise à incorporer le plus possible de connaissances du problème pour atteindre le maximum d'efficacité : l'utilisateur doit notamment définir de façon pertinente le type de caractéristique figurant dans la liste tabou. De leur coté, les algorithmes génétiques se sont éloignés du modèle standard pour intégrer également des connaissances du problème : codage non binaire, opérateurs spécifiques. Au contraire, le recuit simulé est parfois présenté comme une méthode facile à adapter à un problème et qui ne tente pas d'exploiter de connaissances spécifiques.

En général, une méthode offrant des possibilités d'intégrer des connaissances du problème a plus de chance de produire de bons résultats, mais demande un effort

[8] Une autre façon d’interpréter le théorème NFL consiste à dire qu’il n’existe pas de méthode surpassant les autres sur tous les problèmes.

d'adaptation et de spécialisation. Au contraire, une méthode très générale qui prétend n'intégrer aucune connaissance propre ne peut pas être compétitive.

5.3. *Etude de cas : coloration et affectation de fréquences*

Dans cette sous-section, nous comparons les résultats obtenus par les différentes méthodes de résolution pour deux problèmes particuliers : la coloration qui est une référence pour les problèmes NP-complets et l'affectation de fréquences pour les réseaux radio-mobiles qui est une application réelle dans le domaine des télécommunications. Bien que ces deux cas aient une portée limitée comme exemples en optimisation, ce sont deux représentants intéressants de la classe des problèmes d'affectation sous contraintes.

Colorier un graphe consiste à assigner une couleur à chacun de ses sommets sans réutiliser la même couleur pour deux sommets adjacents. Le problème de k-coloration consiste à colorier un graphe avec au plus k couleurs et le problème de coloration avec un nombre minimum de couleurs. Des techniques classiques ont été utilisées pour la coloration : les méthodes de recherche exhaustive (SEP) et les heuristiques gloutonnes. Rappelons que les techniques de SEP sont rapidement limitées par la taille des instances. Parmi les méthodes gloutonnes, citons la méthode de saturation de Brélaz [BRE 79] et la technique RLF (*Recursive Largest First*) de Leighton [LEI 79]. Ces heuristiques spécifiques sont rapides mais fournissent des résultats médiocres.

Les métaheuristiques ont été introduites plus récemment. Plusieurs types de stratégies ont été testés, *i.e.*, plusieurs manières de définir l'espace des configurations et la fonction de coût, ainsi que différents voisinages et différentes métaheuristiques. Hertz et de Werra ont introduit le voisinage le plus simple, analogue à celui qui est présenté dans cet article, pour traiter le problème de k-coloration. Ils ont utilisé successivement le recuit simulé [CHA 87] puis tabou [HER 87] et ont obtenu de meilleurs résultats avec tabou. Johnson et al., ont effectué des comparaisons détaillées entre plusieurs types de voisinage avec le recuit simulé [JOH 91]. Morgenstern combine deux types de voisinages spécifiques de manière particulièrement efficace [MOR 96].

Davis a utilisé un algorithme génétique pour résoudre le problème de coloration avec des résultats assez faibles [DAV 91]. Peu d'études utilisant les algorithmes génétiques ont été publiées depuis. Des approches hybrides ont été proposées plus récemment [FLE 96a, 96b, COS 95]. Fleurent et Ferland utilisent un algorithme hybride combinant un algorithme tabou avec des recombinaisons. Cette technique est coûteuse en temps de calcul mais se révèle capable d’améliorer sur plusieurs instances les résultats obtenus par tabou. Des études plus récentes ont proposé un nouveau type de croisement dont le principe général est de combiner les classes de coloration des configurations parentes pour constituer un descendant [DOR 98, GAL 99a]. Ces croisements hautement spécialisés au problème de coloration permettent, combinés avec un algorithme tabou, ont permis d'obtenir les meilleurs résultats sur certains jeux de test réputés. De plus, l’algorithme est généralement plus rapide que tabou utilisé seul pour obtenir des solutions de même qualité [GAL 99a].

Les résultats comparatifs ont montré que les métaheuristiques sont quasiment les seules méthodes compétitives pour traiter les instances difficiles de grande taille du problème de coloration. Les méthodes de voisinage ont montré une grande efficacité.

Les algorithmes génétiques n'ont pas fourni de résultats compétitifs, mais les méthodes hybrides comptent parmi les techniques les plus prometteuses malgré leur coût. Les résultats montrent également l'importance du choix du voisinage.

Le problème d'affectation de fréquences dans les réseaux radio-mobiles est un exemple réel de problème de type PASC. L'affectation de fréquences est une extension du problème de coloration [HAL 80]. L'objectif de base consiste à affecter des valeurs de fréquences aux stations d'un réseau radio-mobile tout en respectant un ensemble de *contraintes* pour minimiser les interférences et éventuellement le nombre de fréquences utilisées. Les contraintes imposent que les fréquences affectées à deux stations adjacentes soient éloignées d'une distance prédéterminée.

Etant donné la relation forte entre l'affectation de fréquences et la coloration, il n'est pas étonnant de voir qu'il existe d'abondantes méthodes venant du domaine du coloriage dans la littérature [GAM 82, 86]. D'autres approches plus récentes ont été également proposées, citons le recuit simulé [DUQ 93], la recherche locale adaptative [WAN 96], la méthode tabou [CAS 96], les algorithmes génétiques [CRO 94] ou encore les réseaux de neurones [KUN 91, FUN 92].

Dans le cadre d'une étude lancée par le CNET de France Télécom, plusieurs méthodes ont été développées pour le problème d'affectation de fréquences. Ces méthodes sont basées sur les algorithmes évolutifs hybrides [DOR 95, HAO 95], tabou [HAO 97], le recuit simulé [KAR 95, ORT 95], les algorithmes de coloriage [ORT 95] et la programmation par contraintes (ILOG Solver) [CAM 95]. Les tests ont été effectués sur des instances d'environ un millier de variables et plusieurs dizaines de milliers de contraintes. Les meilleurs résultats sur les jeux du CNET ont été obtenus par les algorithmes évolutifs, le recuit simulé et surtout tabou.

5.4. *Quelle métaheuristique utiliser ?*

Le premier problème pratique qui se pose à un utilisateur confronté à une application concrète est d'effectuer un choix parmi les différentes métaheuristiques disponibles. Ce choix est d'autant plus difficile qu'il n'existe pas de comparaison générale et fiable des différentes métaheuristiques. Cependant, il est possible de caractériser les métaheuristiques selon quelques critères généraux, ce qui pourrait faciliter ce choix. Le tableau de synthèse ci-dessous met en relation cinq critères avec cinq métaheuristiques parmi les plus représentatives. Les indications sont présentées à titre purement indicatif et correspondent à notre expérience et aux résultats publiés. Il doit être clair que le choix d'une métaheuristique appropriée ne constitue qu'une condition nécessaire. La qualité des solutions trouvées par une méthode peut être très variable selon l'implémentation réalisée.

	AI	RS	tabou	AG	AH
Simplicité	0	-	-	-	- -
Facilité d'adap.	0	0	-	-	-
Connaissance	0	0	+	+	+
Qualité	0	+	++	+	++ (+++)
Rapidité	0	-	-	- -	- -

Tableau 1. *Comparaison générale des principales métaheuristiques*

Dans ce tableau, les six métaheuristiques comparées sont les suivantes :
- AI : amélioration itérative (descente) avec relance,
- RS : recuit simulé,
- tabou : méthode tabou,
- AG : algorithme génétique adapté,
- AH : algorithme hybride.

Les critères de comparaisons retenus sont les suivants :
- simplicité de la métaheuristique, *i.e.* simplicité de la méthode elle-même,
- facilité d'adaptation au problème,
- possibilité d'intégrer des connaissances spécifiques du problème,
- qualité des meilleures solutions trouvées,
- rapidité, *i.e.*, le temps de calcul nécessaire pour trouver une telle solution (sur une machine séquentielle).

La méthode d'amélioration itérative est utilisée comme point de référence pour l'ensemble des méthodes : les signes -, 0, + indiquent des performances respectivement inférieures, égales, supérieures à celles obtenues par l'amélioration itérative.

Le critère de *qualité* utilisé dans le tableau correspond à la meilleure qualité qu'il est possible d'obtenir par une exécution prolongée. Le critère de *rapidité* représente le temps de calcul typiquement nécessaire pour obtenir une telle solution.

5.5. *Bilan*

Les métaheuristiques présentent de nombreux avantages mais aussi un certain nombre de limitations. Nous résumons ci-dessous les différentes caractéristiques de ces méthodes :
- généralité et application possible à une large classe de problèmes,
- efficacité pour de nombreux problèmes,
- possibilité de compromis entre qualité des solutions et temps de calcul,
- existence de preuves de convergence asymptotique pour certaines méthodes,
- optimum non garanti,
- nécessité d'adaptation de la méthode au problème traité,
- nécessité de réglage des paramètres,
- difficulté de prévoir la performance.

Tout d'abord, rappelons qu'il s'agit d'heuristiques dont l'objectif n'est pas le même que celui des méthodes exactes. En particulier, ces méthodes ne garantissent pas de trouver une solution optimale ni de prouver l'optimalité de la solution trouvée - sauf si une borne inférieure de la fonction de coût est connue et qu'une solution trouvée atteint cette borne.

Dans ces conditions, méthodes complètes et métaheuristiques sont plutôt complémentaires que véritablement concurrentes. Ces deux approches peuvent d'ailleurs également coopérer de différentes manières : l'une des plus courantes consiste à utiliser une méthode de voisinage dans une méthode complète afin de déterminer une borne ou de fixer la valeur d'une variable.

6. Conclusions et perspectives

Les métaheuristiques constituent une classe de méthodes approchées adaptables à un très grand nombre de problèmes combinatoires et de problèmes d'affectation sous contraintes. Elles ont révélé leur grande efficacité pour fournir des solutions approchées de bonne qualité pour un grand nombre de problèmes d'optimisation classiques et d'applications réelles de grande taille. C'est pourquoi l'étude de ces méthodes est actuellement en plein développement.

Afin de résoudre des instances de taille et de difficulté croissantes, il faut mettre au point des méthodes toujours plus puissantes. Pour atteindre cet objectif, au moins deux voies privilégiées se développent : l'hybridation de méthodes et la parallélisation.

Si on peut constater la grande efficacité des métaheuristiques pour de nombreuses classes de problèmes, il existe en revanche très peu de résultats permettant de comprendre la raison de cette efficacité. Cette question constitue sans doute un défi important qu'il revient à la recherche de relever. Nous possédons bien des comparaisons entre métaheuristiques sur différentes classes de problèmes mais nous ne savons généralement ni expliquer le fonctionnement d'une métaheuristique, ni prévoir son efficacité sur une instance donnée. Pour améliorer cette situation, une tendance se dessine qui consiste à privilégier les tests descriptifs : ceux-ci consistent à identifier les facteurs supposés influencer les performances et à effectuer des comparaisons en faisant varier ces facteurs [BAR 95]. Certains auteurs proposent d'aller plus loin et réclament une véritable science empirique des algorithmes devant aboutir à des modèles descriptifs de leur comportement [HOO 94, 95].

En matière de perspectives de recherche, trois voies méritent d'être menées en parallèle. Il faut d'une part poursuivre des études expérimentales des propriétés des métaheuristiques. Il faut d'autre part renforcer et élargir les recherches sur les paysages de recherche. Ces points ont fait l'objet de certaines études récentes (voir par exemple [KAU 89, WEI 90, STA 92, 96, HER 94, JON 95, ANG 98a, 98b, BEL 99]) et méritent sûrement davantage d'investigation. Enfin, il faut développer des recherches fondamentales sur le plan de la complexité des algorithmes et heuristiques [JOH 88, YAN 90, 95].

Remerciements

Nous souhaitons remercier les rapporteurs de ce papier pour leurs critiques et leurs remarques qui ont permis d'améliorer la présentation. Nous remercions R. Dorne pour de nombreuses discussions intéressantes. Les deux premiers auteurs, membres du groupe RESSAC du PRCIA, ont bénéficié d'une subvention de l'ANVAR sur le thème « Nouvelles heuristiques pour l'optimisation combinatoire ».

Bibliographie

[AAR 89] E.H.L. AARTS, J. KORST, *Simulated annealing and boltzmann machines: a stochastic approach to combinatorial and neural computing.* Wiley, Chichester, 1989.

[AAR 97] E.H.L. AARTS, J.K. LENSTRA (Eds.), *Local search in combinatorial optimization*, John Wiley & Sons, 1997.

[ACK 87] D.H. ACKLEY, *A connectionist machine for genetic hillclimbing*, Kluwer Academic Publishers, 1987.

[ALT 91] I. ALTHOFER, K.U. KOSCHNICK, On the convergence of "threshold accepting". *Applied Mathematics and Optimization* 24 : 183-195, 1991.

[ANG 98a] E. ANGEL, V. ZISSIMOPOULOS, On the quality of local search for the quadratic assignment problem. *Discrete Applied Mathematics* 82 : 15-25, 1998.

[ANG 98b] E. ANGEL, V. ZISSIMOPOULOS, Autocorrelation coefficient for the graph bipartitioning problem, *Theoritical Computer Science* 191 : 229-243, 1998.

[BÄC 91] T. BÄCK, F. HOFFMEISTER, H-P. SCHWEFEL, A Survey of evolutionary strategies. *Proc. of 4th Intl. Conference on Genetic Algorithms (ICGA'91),* R. Belew, L. Booker (Eds.), Morgan Kaufmann, p. 2-9, 1991.

[BÄC 93a] T. BÄCK, H-P. SCHWEFEL, An overview of evolutionary algorithms for parameter optimization. *Evolutionay Computation* 1(1) : 1-24, 1993.

[BÄC 93b] T. BÄCK, F. HOFFMEISTER, H.P. SCHWEFEL (Eds.), Applications of evolutionary algorithms. Report of the System Analysis Research Group (Sys), Univ. of Dortmund, ftp//lumpi.informatik.uni-dortmund.de/pub/EA/papers/ea-app.ps.gz, 1993.

[BÄC 95] T. BÄCK, *Evolutionary algorithms in theory and practice*. Oxford University Press, New York, 1995.

[BAL 98] E. BALAS, W. NIEHAUS, Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics* 4 : 107-122, 1998.

[BAR 95] R.S. BARR, R.L. GOLDEN, J.P. KELLY, M.G.C. RESENDE, W.R. STEWART, Jr., Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics* 1(1) : 9-32, 1995.

[BAT 97] R. BATTITI, M. PROTASI, Reactive search, a history-sensitive heuristic for MAX-SAT, *ACM Journal of Experimental Algorithmics* 2, article 2, 1997.

[BEA 93] J.E. BEASLEY, Langrangean relaxation. Dans C.R. Reeves (Ed.) *Modern heuristics for combinatorial problems*, Blackwell Scientific Publishing, Oxford, 1993.

[BEL 99] M. BELAIDOUNI, J.K. HAO, Landscapes of the maximal constraint satisfaction problem. Artificial Evolution'99, *Lecture Notes in Computer Science*, Springer-Verlag, 1999.

[BER 64] P. BERTIER, B. ROY, Procédure de résolution pour une classe de problème pouvant avoir un caractère combinatoire. *Cahiers du Centre d'Etudes de Recherche Opérationnelle 6* : 202-208, 1964.

[BIT 75] J.R. BITNER, E.M. REINGOLD, Backtracking programming techniques. *Communications of the ACM* 18(11) : 651-656, 1975.

[BON 84] E. BONOMI, J.L. LUTTON, The N-city travelling salesman problem, statistical mechanics and the metropolis algorithm, *SIAM Review* 26(4) : 551-568, 1984.

[BRE79] D. BRELAZ, New methods to color vertices of a graph. *Communications of ACM* 22 : 251-256, 1979.

[CAM 95] A. CAMINADA, Résolution du problème de l'affectation des fréquences par programmation par contraintes. Note Technique FT.CNET/BEL/POH/CDI/71-95/CA, CNET Belfort, 1995.

[CAS 96] D.J. CASTELINO, S. HURLEY, N.M. STEPHENS, A tabu search algorithm for frequency assignment. G. Laporte, I. H. Osman, (Eds.), *Metaheuristics in Combinatorial Optimization*, *Annals of Operations Research* 63 : 301-319, 1996.

[CER 94] R. CERF, Une théorie asymptotique des algorithmes génétiques, Thèse de Doctorat, Université de Montpellier II, 1994.

[CER 85] V. CERNY, A thermodynamical approach to the travelling salesman problem: an efficient simulated annealing algorithm. *Journal of Optimization Theory and Applications* 45 : 41-51, 1985.

[CHA 87] M. CHAMS, A. HERTZ, D. de WERRA, Some experiments with simulated annealing for coloring graphs. *EJOR* 32 : 260-266, 1987.

[CHA 93] I. CHARON, O. HUDRY, The noising method: a new method for combinatorial optimization, *Operations Research Letters* 14 : 133-137, 1993.

[CHA 95] I. CHARON, O. HUDRY, Mixing different component of metaheuristics. I.H. Osman, J.P. Kelly (Eds.), *Metaheuristics, Theory and Applications*, Kluwers Academic Publishers, p. 589-604, 1995.

[CHE 91] P. CHEESEMAN, B. KANEFSKY, W.M. TAYLOR, Where the really hard problems are? *Proc. of IJCAI'91*, Sydney, Springer-Verlag, p. 331-337, 1991.

[COL 88] N.E. COLLINS, R.W. EGLESE, B.L. GOLDEN, Simulated annealing: an annotated bibliography. *American Journal of Mathematical and Management Sciences* 8(3-4) : 209-307, 1988.

[COL 90] A. COLMERAUER, An introduction to Prolog III, *Communications of the ACM* 33(7) : 69-90, 1990.

[CON 90] D. CONNOLLY, An improved annealing scheme for the QAP. *European Journal of Operational Research* 46 : 93-100, 1990.

[COO 71] S.A. COOK, The complexity of theory-proving procedures. Proc. of 3rd Annual *ACM Symposium on Theory of Computing*, ACM, Ney York, p. 151-158, 1971.

[COS 95] D. COSTA, A. HERTZ, O. DUBUIS, Embedding of a sequential procedure within an evolutionary algorithms for coloring problems in graphs. *Journal of Heuristics* 1(1) : 105-128, 1995.

[CRE 96] P. CRESCENZI, V. KANN, A compendium of NP-optimization problems. Research Report, University La Sapienza, Roma, 1996.

[CRO 94] W. CROMPTON, S. HURLEY, N.M. STEPHENS, A parallel genetic algorithm for frequency assignment problems. *Proc. of IMACS SPRANN'94*, p. 81-84, 1994.

[DAV 91] L. DAVIS, *Handbook of genetic algorithms*, Van Nostrand Reinhold, New York, 1991.

[DIN 96] M. DINCBAS, P. VAN HENTENRYCK, H. SIMONIS, A. AGGOUN, T. GRAF, F. BERTHIER, The constraint logic programming language CHIP. *Proc. of Intl. Conf. on Fifth Generation Computer Systems*, p. 693-702, Tokyo, Japan, 1988.

[DEJ 93] K.A. DE JONG, W. SPEARS, On the state of evolutionary computation. *Proc. of International Conference on Genetic Algorithms (ICGA'93)*, p. 618-623, 1993.

[DEJ 94] K.A. DE JONG, Genetic algorithms : A 25 year perspective. Dans J.M. Zurada, R.J. Marks II, C.J. Robinson (Eds.), *Computional Intelligence-Imitating Life*, IEEE Press, p. 125-134, 1994.

[DOR 94] U. DORNDORF, E.PESCH, Fast clustering algorithms. *ORSA Journal on Computing* 6 : 141-153, 1994.

[DOR 95] R. DORNE, J.K. HAO, An evolutionary approach for frequency assignment in cellular radio networks. *Proc. of IEEE Intl. Conf. on Evolutionary Computation (ICEC'95)*, p. 539-544, Perth, Australie, 1995.

[DOR 96] R. DORNE, J.K. HAO, Constraint handling in evolutionary search: A case study on frequency assignment. *Lecture Notes in Computer Science 1141* : 801-810, Springer-Verlag, 1996.

[DOR 98] R. DORNE, J.K. HAO, A new genetic local search algorithm for graph coloring. *Lecture Notes in Computer Science 1498 :* 745-754, Springer-Verlag, 1998.

[DOR 99] M. DORIGO, G. DI CARO, The ant colony optimization meta-heuristic. Dans D. Corne, M. Dorigo, F. Glover (Eds.), *New Ideas in Optimization*, McGraw-Hill, 1999.

[DUC 90] G. DUCEK, T. SCHEUER, Threshold accepting: a general purpose optimization algorithm. *Journal of Computational Physics* 90 : 161-175, 1990.

[DUC 93] G. DUCEK, New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics* 104 :86-92, 1993.

[DUQ 93] M. DUQUE-ANTON, D. KUNZ, B. RÜBER, Channel assignment for cellular radio using simulated annealing. *IEEE Trans. on Vehicular Technology* 42 :14-21, 1993.

[EGL 86] R. EGLESE, Heuristics in operational research. Dans V. Belton, B. O'keefe (Eds.), *Recent Developments in Operational Research*, Pergamon Press, 1986.

[EGL 90] R.W. EGLESE, Simulated annealing: a tool for operational research. *European Journal of Operational Research* 46 : 271-281, 1990.

[FAI 92] U. FAIGLE, W. KERN, Some convergence results for probabilistic Tabu Search. *ORSA Journal on Computing* 4 : 32-37, 1992.

[FAL 96] E. FALKENAUER, A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics* 2(1) : 5-30, 1996.

[FAR 95] H. FARGIER, D. DUBOIS, H. PRADE, Problèmes de satisfaction de contraintes flexibles : une approche égalitariste, *Revue d'Intelligence Artificielle* 9 : 311-354, 1995.

[FEO 89] T.A. FEO, M.G.C. RESENDE, A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* 8 : 67-71, 1989.

[FEO 94] T.A. FEO, M.G.C. RESENDE, S.H. SMITH, A greedy randomized adaptive search procedure for maximum independent set. *Operations Research* 42 : 860-878, 1994.

[FEO 94] T.A. FEO, M.G.C. RESENDE, Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6: 109-133, 1995.

[FIS 89] M.L. FISHER, A.H.G. RINNOY KAN, The Design, analysis and implementation of heuristics. *Management Science* 34 : 263-265, 1989.

[FLE 96a] C. FLEURENT, J.A. FERLAND, Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. Dans [JOH 96], p. 619-652, 1996.

[FLE 96b] C. FLEURENT, J.A. FERLAND, Genetic and hybrid algorithms for graph coloring. Dans G. Laporte, I.H. Osman, (Eds.), *Metaheuristics in Combinatorial Optimization, Annals of Operations Research,* 63 : 437-441, 1996.

[FOG 66] L.J. FOGEL, A.J. OWENS, M.J. WALSH, *Artificial intelligence through simulated evolution*, Wiley, New York, 1966.

[FOG 90] D.B. FOGEL, J.W. ATMAR, Comparing genetic operators with gaussian mutations in simulted evolutionary prosesses using linear systems. *Biol. Cybern* 63 : 111-114, 1990.

[FOG 94] D.B. FOGEL, Evolutionary programming: an introduction and some current directions. *Statistics and Computing* 4 : 11-129, 1994.

[FRE 96] B. FREISLEBEN, P. MERZ, New genetic local search operators for the traveling saleman problem *Lecture Notes in Computer Science 1141* : 890-899, Springer-Verlag, 1996.

[FRE 92] E.C. FREUDER, R.J. WALLACE, Partial constraint satisfaction. *Journal of Artificial Intelligence* 58(1-3) : 21-70, 1992.

[FUN 92] N. FUNABIKI, Y. TAKEFUJI, A neural network parallel algorithm for channel assignment problems in cellular radio network. *IEEE Transactions on Vehicular Technology* 41 : 430-437, 1992.

[GAL 97] P. GALINIER, J.K. HAO, Tabu search for maximal constraint satisfaction problems. *Lecture Notes in Computer Science 1330* : 196-208, Springer-Verlag, 1997.

[GAL 99a] P. GALINIER, J.K. HAO, Hybrid evolutionary algorithms for graph coloring. à paraître dans *Journal of Combinatorial Optimization*, 1999.

[GAL 99b] P. GALINIER, Etude des métaheuristiques pour la résolution du problème de satisfaction de contraintes et de la coloration de graphes. Thèse de Doctorat, Université de Montpellier II, janvier 1999.

[GAM 82] A. GAMST, W. RAVE, On the frequency assignment in mobile automatic telephone systems. *Proc. of GLOBECOM'92*, p. 309-315, 1982.

[GAM 86] A. GAMST, Some lower bounds for a class of frequency assignment problems. *IEEE Transactions on Vehicular Technology* 35 : 8-14, 1986.

[GAR 79] M.R. GAREY, D.S. JOHNSON, *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman and Company, New York, 1979.

[GLO 65] F. GLOVER, A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research* 13 : 879-919, 1965.

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

[GLO 86] F. GLOVER, Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* 13 : 533-549, 1986.

[GLO 89] F. GLOVER, Tabu search : part I. *ORSA J. on Computing* 1(3) : 190-206, 1989.

[GLO 89] F. GLOVER, Tabu Search; Part II. *ORSA J. on Computing* 2(1) : 4-32, 1989.

[GLO 93a] F. GLOVER, M. LAGUNA, Tabu Search, Dans C.R. Reeves (Ed.) *Modern heuristics for combinatorial problems*, Blackwell Scientific Publishing, Oxford, 1993.

[GLO 93b] F. GLOVER, E. TAILLARD, M. LAGUNA, D. de WERRA (Eds.), Tabu search. *Special Issue of Annals of Operations Research* 41, 1993.

[GLO 95a] F. GLOVER, J.P. KELLY, M. LAGUNA, Genetic algorithms and tabu search - hybrids for optimization. *Computers & Operations Research* 22 : 111-?. 1995.

[GLO 97] F. GLOVER, M. LAGUNA, *Tabu search*, Kluwer Academic Publishers, 1997.

[GLO 98] F. GLOVER, A template for scatter search and path relinking. Dans J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), Artificial Evolution'97, *Lecture Notes in Computer Sciences 1363* : 3-51, Spinger-Verlag, 1998.

[GOL 89] D.E. GOLDBERG, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, New York, 1989.

[GOE 95] M.X. GOEMANS, D.P. WILLIAMSON, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM* 42 : 1115-1145, 1995.

[GRE 87] J.J. GREFENSTETTE, Incorporating problem specific knowledge into genetic algorithms. Dans L. Davis (Ed.) *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, p. 42-60, 1987.

[HAJ 88] B. HAJEK, Cooling schedules for optimal annealing. *Mathematics of Operations Research* 13 : 311-329, 1988.

[HAL 80] W.K. HALE, Frequency assignment: theory and application. *Proceedings of the IEE*E 68(12) : 1498-1573, 1980.

[HAN 86] P. HANSEN, The steepest ascent mildest descent heuristic for combinatorial programming. *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italie, 1986.

[HAN 90] P. HANSEN, B. JAUMARD, Algorithms for the maximum satisfiability problem. *Computing* 44 : 279-303, 1990.

[HAO 95] J.K. HAO, R. DORNE, Study of genetic search for the frequency assignment problem. *Lecture Notes in Computer Science 1063* : 333-344, Springer-Verlag, 1996.

[HAO 96] J.K. HAO, R. DORNE, Empirical studies of heuristic local search for constraint solving. *Lecture Notes in Computer Science 1118* : 194-208, Springer-Verlag, 1996.

[HAO 97] J.K. HAO, R. DORNE, P. GALINIER, Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics* 4(1) : 47-62, 1998.

[HAR 87] J. HART, A. SHOGAN, Semi-greedy heuristics: an empirical study. *Operations Research Letters* 6 : 107-114, 1987.

[HEL 70] M. HELP, R.M. KARP, The travelling-saleman problem and minimum spanning trees. *Operations Research* 18 : 1138-1162, 1970.

[HEL 71] M. HELP, R.M. KARP, The travelling-saleman problem and minimum spanning trees: part II. *Mathematical Programming* 1 : 6-25, 1971.

[HER 87] A. HERTZ, D. De WERRA, Using Tabu search techniques for graph coloring. *Computing* 39 : 345-351, 1987.

[HER 94] A. HERTZ, B. JAUMARD, M.P. DE ARAGAO, Local Optima Topology for the k-coloring Problem. *Discrete Applied Mathematics* 49 : 257-280, 1994.

[HOL 75] J.H. HOLLAND, *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[HOL 92] J.H. HOLLAND, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* MIT Press/Bradford Books, 2nd Edition, Cambridge, MA, 1992.

[HOO 94] J.N. HOOKER, Needed: An empirical science of algorithms. *Operations Research* 42 : 201-212. 1994.

[HOO 95] J.N. HOOKER, Testing heuristics: we have it all wrong. *Journal of Heuristics* 1(1) : 33-42 1995.

[HUB 96] B.A. HUBERMAN, T. HOGG, C.P. WILLIAMS, *Artificial Intelligence*, *Special Issue on Phase Transition and Complexit*y 82(1-2), 1996.

[JOG 87] P. JOG, G. GUCHT, Parallelisation of probabilistic sequential search algorithms. *Proc. of ICGA'87,* San Mateo: Morgan Kaufmann, p. 170-176, 1987.

[JOG 91] P. JOG, J. SUH, G. GUCHT, Parallel genetic algorithms applied to the traveling salesman problem. *SIAM Journal on Optimization* 1(4) : 515-529, 1991.

[JOH 88] S. JOHNSON, C.H. PAPADIMITRIOU, M. YANNAKAKIS, How easy is local search? *Journal of Computer and System Sciences* 37(1) : 79-100, 1988.

[JOH 89] D.S. JOHNSON, C.R. ARAGON, L.A. MCGEOCH, C. SCHEVON, Optimization by simulated annealing: an experimental evaluation; Part I, graph partitioning. *Operations Research* 37 : 865-892, 1989.

[JOH 91] D.S. JOHNSON, C.R. ARAGON, L.A. MCGEOCH, C. SCHEVON, Optimization by simulated annealing: an experimental evaluation; Part II, graph coloring and number partitioning. *Operations Research* 39(3) : 378-406, 1991.

[JOH 96] D.S. JOHNSON, M.A. TRICK (Eds.), 2nd DIMACS implementation challenge – cliques, coloring and satisfiability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26,* American Mathematical Society, 1996.

[JON 95] T. JONES, S. FORREST, Fitness distance correlation as a measure of problem difficulty for genetic algorithms. *Proceedings of the 6th International Conference on Genetic Algorithms,* 184-192, 1995.

[KAU 89] S.A. KAUFFMAN, Adaptation on rugged fitness landscapes. Dans D. Stein (Ed.), *Lectures in the Sciences of Complexity,* Addison-Wesley Longman, 527-618, 1989.

[KAR 95] M. KARRAY, A. ORTEGA, J-M. RAIBAUD, Etudes d'algorithmes de recherche de cliques et de coloration des graphes. Note Technique NT/PAB/SRM/RRM/4027, CNET Paris, 1995.

[KER 93] W. KERN, On the depth of combinatorial optimization problems. *Discrete Applied Math.* 43: 115-129, 1993.

[KER 70] B.W. KERNIGHAN, S. LIN, An efficient heuristic for partitioning graphs. *Bell System Technology Journal* 49(2) : 291-307, 1970.

[KIR 83] S. KIRKPATRICK, C.D. GELATT, P.M. VECCHI, Optimization by simulated annealing. *Science* 220 : 671-680, 1983.

[KOL 94] A. KOLEN, E. PESCH, Genetic local search in combinatorial optimization, *Discrete Applied Mathematics*. 48 : 273-, 1994.

[KON 94] K. KONOLIGE, Easy to be hard : Difficult problems for greedy algorithms. *Proc. of 4th Intl. Conf. on Principles of Knowledge Representation and Reasoning*, p. 374-378, 1994.

[KON 95] G. KONTORAVDIS, J.F. BARD, Improved heuristics for the vehicle routing problem with time windows. *ORSA Journal on Computing* 7, 1995.

[KOR 91] B. KORTE, L. LOVASZ, R. SCHRADER, GREEDOIDS, *Algorithms and Combinatorics* 4, Springer Verlag, 1991.

[KOU 94] C. KOULAMAS, S.R. ANTHONY, R. JEAN, A survey of simulated annealing application to operations research problems. *OMEGA* 22 : 41-56, 1994.

[KUL 89] M. KUBALE, Interval vertex-coloring of a graph with forbidden colors. *Discrete Mathematics* 74: 125-136, 1989.

[KUN 91] D. KUNZ, Channel assignment for cellular radio using neural networks. *IEEE Trans. on Vehicular Technology* 40 : 88-193, 1991.

[LAG 94] M. LAGUNA, T.A. FEO, H.C. ELROD, A greedy randomized adaptive search procedure for the two-partition problem. *Operations Research* 42 : 677-687, 1994.

[LAW 66] E.L. LAWLER, D.E. WOOD, Branch and bound methods : a survey. *Operations Research* 14 : 699-719, 1996.

[LAP 96] G. LAPORTE, I.H. OSMAN, *Metaheuristics in combinatorial optimization, Annals of Operations Research* 63, J.C. Baltzer Science Publishers, Basel, Switzerland, 1996.

[LAU 78] J-L. LAURIERE, A language and a program for stating and solving combinatorial problems, *Artificial Intelligence* 10 : 29-127, 1978.

[LEI 79] F.T. LEIGHTON, A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau Standard* 84 : 489-505, 1979.

[LIN 65] S. LIN, Computer solutions of the travaleing salesman problem. *Bell System Technology Journal* 44(10) : 2245-2269, 1965.

[LIN 73] S. LIN, B.W. KERNIGHAN, An efficient heuristic for the traveling-salesman problem. *Operations Research* 21 : 498-516, 1973.

[LIN 93] F.T. LIN, C.Y.KAO, C.C. HSU, Applying the genetic approach to simulated annealing in solving some NP-hard problems. *IEEE Transactions on Systems, Man and Cybernetics* 23 : 1752-1767, 1993.

[LUN 86] S. LUNDY, A. MEES, Convergence of an annealing algorithm. *Mathematical Programming* 34 : 111-124, 1986.

[MAC 77] A.K. MACKWORTH, Consistency in networks of relations. *Journal of Artificial Intelligence* 8(1) : 99-118, 1977.

[MAC 87] A.K. MACKWORTH, Constraint satisfaction, Dans S.C. Shapiro (Ed.) *Encyclopedia on Artificial Intelligence*, John Wiley & Sons, NY, 1987.

[MAH 95] S.W. MAHFOUD, D.E. GOLDBERG, Parallel recombinative simulated annealing : a genetic algorithms. *Parallel Computing* 21 : 1-28, 1995.

[MET 53] W. METROPOLIS, A. ROENBLUTH, M. ROSENBLUTH, A. TELLER, E.TELLER, Equation of the state calculations by fast computing machines. *Journal of Chemical Physics* 21 : 1087-1092, 1953.

[MIC 92] Z. MICHALEWICZ, *Genetic Algorithms + data structures = evolution programs,* Springer Verlag, Berlin, 1992.

[MIN 90] S. MINTON, M.D. JOHNSTON, P. LAIRD, Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method. *Proc. of AAAI'90,* Boston, MA, 1990.

[MIN 92] S. MINTON, M.D. JOHNSTON, P. LAIRD, Minimizing conflicts : A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58(1-3) : 161-206, 1992.

[MOR 93] P. MORRIS, The breakout method for escaping from local minima. *Proc. of AAAI'93*, 1993.

[MOR 96] C. MORGENSTERN, Distributed coloration neighborhood search. Dans [JOH 96], p. 335-358, 1996.

[MOS 89] P. MOSCATO, On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech Concurrent Computation Program, C3P Report 826, 1989.

[MOS 93] P. MOSCATO, An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search. *Annals of Operations Research* 41 : 85-122, 1993.

[MOT 95] R. MOTVANI, Lecture notes on approximation algorithms, Volume I, Research Report, Standford University, 1995.

[MÜL 88] H. MÜLHENBEIN, M. GORGES-SCHLEUTER, O. KRÄMER, Evolution algorithms in combinatorial optimization. *Parallel Computing* 7 : 65-85, 1988.

[MÜL 95] H. MÜLHENBEIN, Evolutionary algorithms: theory and application. Dans E.H.L. Aarts, J.K. Lenstra (Eds.) *Local search in combinatorial optimization*, John Wiley & Sons, Chichester, 1995.

[NIC 71] T. NICHOLSON, Optimization techniques. dans *Optimization in Industry*, Longman Press, London. 1971.

[ORT 95] A. ORTEGA, J.M. RAIBAUD, M.KARRAY, M.MARZOUG, A. CAMINADA, Algorithmes de coloration des graphes et d'affectation des fréquences. Note Technique, NT/PAB/SRM/ RRM/4353, CNET Paris, Août 1995.

[OSM 95] I.H. OSMAN, An introduction to meta-heuristics. Dans M. Lawerence, C. Wilson (Eds.) *Operational Research Papers*, ORS Press, Birmingham, U.K., 1995.

[OSM 96] I.H. OSMAN, J.P. KELLY (Eds.), *Meta-heuristics: theory and applications*, Kluwers Academic Publishers, Boston, 1996.

[PAP 82] C.H. PAPADIMITRIOU, K. STEIGLITZ, *Combinatorial optimization - algorithms and complexity*. Prentice Hall, 1982.

[PAP 94] C.H. PAPADIMITRIOU, *Computional complexity*. Addison-Wesley, 1994.

[PEA 84] J. PEARL, *Heuristics*, Addison-Wesley, Reading, MA, 1984.

[PUG 94] J.P. PUGET, A C++ implementation of CLP, *Prof. of Singapore Intl. Conf. on Intelligent Systems*, 1994.

[RAD 95] N.J. RADCLIFF, P.D. SURRY, Fundamental limitations on search algorithms: evolutionary computing in perspective. *Lecture Notes in Computer Science 1000*, Springer-Verlag, 1995.

[RAR 93] R.L. RARDIN, M. SUDIT, Paroidal search: genetic local combinatorial optimization. *Discrete Applied Math.* 43 : 155-174, 1993.

[REC 73] I. RECHENBERG, *Evolutionasstrategie : optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, Stuttgart, 1973.

[REE 93a] C.R. REEVES (Ed.) *Modern heuristic techniques for combinatorial problems*, Blackwell Scientific Publications, Oxford, 1993.

[REE 93b] C.R. REEVES, J. BEASLEY, Introduction. Dans C.R. Reeves (Ed.) *Modern heuristic techniques for combinatorial problems*, Blackwell Scientific Publications, Oxford, 1993.

[RIB 94] C.C. RIBEIRO, N. MACULAN (Eds.), Applications of combinatorial optimization. *Annals of Operations Research* 50, 1994.

[SCH 91] A.A. SCHAFFER, M. YANNAKAKIS, Simple local search problems that are hard to solve. *SIAM J. Computing* 20(1) : 56-87, 1991.

[SCH 97] T. SCHIEX, H. FARGIER, G. VERFAILLIE, Problèmes de satisfaction de contraintes valués, *Revue d'Intelligence Artificielle* 11(3) : 339-373, 1997.

[SCH 97b] M. SCHOENAUER, Z. MICHALEWICZ, Evolutionary computation: an introduction. *Control and Cybernetics* 26(3) : 307-338, 1997.

[SCH 81] H.P. SCHWEFEL, *Numerical optimization of computer models*, John Wiley & Sons, 1981.

[SEL 92] B. SELMAN, H.J. LEVESQUE, M. MITCHELL, A new method for solving hard satisfiability problems. *Proc. of AAAI'92*, San Jose, CA, p. 440-446, 1992.

[SEL 93] B. SELMAN, H.A. KAUTZ, Domain-independent extensions to GSAT : solving large structured satisfiability problems. *Proc. of IJCAI'93*, Chambéry, p. 46-51, 1993.

[SEL 94] B. SELMAN, H.A. KAUTZ, C. BRAM, Noise strategies for improving local search. *Proc. of AAAI'94*, Seattle, WA, 1994.

[SHA 87] S.C. SHAPIRO (Ed.), *Encyclopedia of artificial intelligence*. John Wiley & Sons, NY, 1987.

[SIL 80] A. SILVER, R.V.V. VIDAL, D. de WERRA, A tutorial on heuristic methods. *European Journal of Operational Research* 5 : 153-162, 1980.

[SIN 93] M. SINCLAIR, Comparison of the performance of modern heuristics for combinatorial problem on real data. *Computers & Operations Research* 20 : 687-695, 1993.

[SPE 96] W.M. SPEARS, Simulated annealing for hard satisfiability problems. Dans [JOH 96], p. 533-558, 1996.

[STA 92] P.F. STADLER, Correlation in landscapes of combinatiorial optimization problems. *Europhys. Letters* 20 : 479-482, 1992.

[STA 96] P.F. STADLER, Landscapes and their correlation functions. *Journal of Math. Chem*. 20 : 1-45, 1996.

[STE 90] R.E. STEARNS, H.B. HUNT III, Power indices and easier hard porblem. *Math. Syst. Theory* 23: 209-225, 1990.

[STE 68] K. STERGLITZ, P. WEINER, Some improved algorithms for computer solution of the travelling salesman problem. *Proc. of 6th Allerton Conf. on Circuit and System Theory*, Urbana, Illinois, p. 814-921, 1968

[TAL 98] E-G. TALBI, Z. HAFIDI, J-M. GEIB, Parallel adaptive tabu search for large optimization problems. à paraître dans *Parallel Computing*.

[TSA 93] E. TSANG, *Foundations of constraint satisfaction*, Academic Press, 1993.

[ULD 91] N.L.J. ULDER, E.H.L. AARTS, H.J. BANDELT, P.J.M. van LAARHOVEN, E. PESCH, Genetic local search algorithms for the traveling salesman problem. *Lecture Notes in Computer Science 496*, 1991.

[VAN 87] P.J. van LAARHOVEN, E.H.L. AARTS, *Simulated annealing: theory and applications*, Reidl, Dordrecht, 1987.

[VER 95] M.G.A. VERHOEVEN, E.H.L. AARTS, Parallel local search. *Journal of Heuristics* 1(1) : 43-65, 1995.

[VID 93] R.V. VIDAL (Ed.) Applied simulated annealing, *Lectures Notes in Economics and Mathematical Systems 396* : Springer-Verlag, Berlin, 1993.

[VOU 95] C. VOUDOURIS, E. TSANG, Guided local search. TR CSM-247, University of Essex, UK, 1995.

[WAL 96] R. WALLACE, Analysis of heuristic methods for partial constrait satisfaction problem. *Lecture Notes in Computer Science 1118* : 482-486, Springer-Verlag, 1996.

[WAN 96] W. WANG, C.K. RUSHFORTH, An adaptive local-search algorithm for the channel assignment problem (CAP). *IEEE Trans. Vehicular Tech.* 45 : 459-466, 1996.

[WEI 90] E. WEINBERGER, Correlated and uncorrelated fitness landscape and how to tell the difference. *Biol. Cybern*. 63 : 325-336, 1990.

[WOL 95] D.H. WOLPERT, W. G. MACREADY, No free lunch theorems for optimization. *IEEE Transactions On Evolutionary Computation* 1(1) : 67-82, 1997.

[YAN 90] M. YANNAKAKIS, The analysis of local search problems and their heuristics. Proc. of 7th Symposium on Theoritical Aspects of Computer Science, *Lecture Notes in Computer Science 415* : 298-311, Springer-Verlag, 1990.

[YAN 95] M. YANNAKAKIS, Computational complexity of local search. Dans E.H.L. Aarts, J.K. Lenstra (Eds.), *Local Search in Combinatorial Optimization*, John Wiley & Sons, Chicester, 1995.

[YUG 94] N. YUGAMI, Y. OHTA, H. HARA, Improving repair-based constraint satisfaction methods by value propagation. *Proc. of AAAI'94*, Seattle, WA, p. 344-349, 1994.

[ZAN 89] H. S. ZANAKIS, J.R. EVANS, A.A. VAZACOPOULOS, Heuristic methods and applications: a categorized survey. *European Journal of Operational Research* 43 : 88-110, 1989.