Méta-heuristiques et intelligence artificielle

Jin-Kao Hao, Christine Solnon

Chapitre 1

Méta-heuristiques et intelligence artificielle

Auteurs : Jin-Kao Hao et Christine Solnon

1.1 Introduction

Une méta-heuristique est une méthode générique pour la résolution de problèmes combinatoires NP-difficiles. La résolution de ces problèmes nécessite l'examen d'un très grand nombre (exponentiel) de combinaisons. Tout un chacun a déjà été confronté à ce phénomène d'explosion combinatoire qui transforme un problème apparemment très simple en un véritable casse-tête dès lors que l'on augmente la taille du problème à résoudre. C'est le cas par exemple quand on cherche à concevoir un emploi du temps. S'il y a peu de cours à planifier, le nombre de combinaisons à explorer est faible et le problème est très rapidement résolu. Cependant, l'ajout de quelques cours seulement peut augmenter considérablement le nombre de combinaisons à explorer de sorte que le temps de résolution devient excessivement long.

L'enjeu pour résoudre ces problèmes est de taille : un très grand nombre de problèmes industriels sont confrontés à ce phénomène d'explosion combinatoire comme, par exemple, les problèmes consistant à planifier une production en minimisant les pertes de temps, ou à découper des formes dans des matériaux en minimisant les chutes. Il est donc crucial de concevoir des approches "intelligentes", capables de contenir ou de contourner l'explosion combinatoire afin de résoudre ces problèmes difficiles en un temps acceptable.

Les problèmes d'optimisation combinatoires peuvent être résolus par deux principales familles d'approches. Les approches "exactes" explorent de façon systématique l'espace des combinaisons jusqu'à trouver une solution optimale. Afin de (tenter de) contenir l'explosion combinatoire, ces approches structurent l'espace des combinaisons en arbre et utilisent des techniques d'élagage, pour réduire cet espace, et des heuristiques, pour déterminer l'ordre dans lequel il est exploré. Ces approches exactes permettent de résoudre en pratique de nombreux problèmes d'optimisation combinatoires. Cependant, les techniques de filtrage et les heuristiques d'ordre ne réduisent pas toujours suffisamment la combinatoire, et certaines instances de problèmes ne peuvent être résolues en un temps acceptable par ces approches exhaustives.

Une alternative consiste à utiliser des méta-heuristiques qui contournent le problème de l'explosion combinatoire en n'explorant délibérément qu'une partie de l'espace des combinaisons. Par conséquent, elles peuvent ne pas trouver la solution optimale, et encore moins prouver l'optimalité de la solution trouvée ; en contrepartie, la complexité en temps est généralement faiblement polynomiale.

Organisation du chapitre. Il existe essentiellement deux grandes familles d'approches heuristiques : les *approches perturbatives*, présentées en 1.2, construisent des combinaisons en modifiant des combinaisons existantes ; les *approches constructives*, présentées en 1.3, génèrent des combinaisons de façon

incrémentale en utilisant pour cela un modèle stochastique. Ces différentes approches peuvent être hybridées, et nous présentons en 1.4 quelques uns des schémas d'hybridation les plus connus. Nous introduisons ensuite en 1.5 les notions d'intensification (exploitation) et de diversification (exploration), communes à toutes ces approches heuristiques : l'intensification vise à diriger l'effort de recherche aux alentours des meilleures combinaisons trouvées, tandis que la diversification vise à garantir un bon échantillonnage de l'espace de recherche. Enfin, nous développerons en 1.6 deux applications de ces méta-heuristiques à la résolution de problèmes classiques de l'intelligence artificielle, à savoir la satisfiabilité de formules booléennes et la satisfaction de contraintes .

Notations. Dans la suite de ce chapitre, nous supposerons que le problème à résoudre est défini par un couple (E, f) tel que E est un ensemble de combinaisons candidates, et $f : E \to \mathbb{R}$ est une fonction objectif associant à chaque combinaison de E une valeur numérique. Résoudre un tel problème consiste à chercher la combinaison $e^* \in E$ qui optimise (maximise ou minimise, selon les cas) f.

Nous illustrerons plus particulièrement les différentes méta-heuristiques introduites dans ce chapitre sur le problème du voyageur de commerce, qui constituera notre "fil rouge" : étant donnés un ensemble V de villes et une fonction $d : V \times V \to \mathbb{R}$ donnant pour chaque paire de villes différentes $\{i, j\} \subseteq V$ la distance $d(i, j)$ séparant les villes i et j, il s'agit de trouver le plus court circuit passant par chaque ville de V une et une seule fois. Pour ce problème, l'ensemble E des combinaisons candidates est défini par l'ensemble des permutations circulaires de V et la fonction objectif f à minimiser est définie par la somme des distances entre deux villes consécutives dans la permutation.

1.2 Méta-heuristiques perturbatives

Les approches perturbatives explorent l'espace des combinaisons E en perturbant itérativement des combinaisons déja construites : partant d'une ou plusieurs combinaisons initiales (généralement prises aléatoirement dans E), l'idée est de générer à chaque étape une ou plusieurs nouvelles combinaisons en modifiant une ou plusieurs combinaisons générées précédemment. Ces approches sont dites "basées sur les instances" dans [77]. Les approches perturbatives les plus connues sont les algorithmes génétiques, décrits en 1.2.1, et la recherche locale, décrite en 1.2.2.

1.2.1 Algorithmes génétiques

Les algorithmes génétiques [32, 24, 14] s'inspirent de la théorie de l'évolution et des règles de la génétique qui expliquent la capacité des espèces vivantes à s'adapter à leur environnement par la combinaison des mécanismes suivants :

- la *sélection naturelle* fait que les individus les mieux adaptés à l'environnement tendent à survivre plus longtemps et ont donc une plus grande probabilité de se reproduire ;
- la *reproduction par croisement* fait qu'un individu hérite ses caractéristiques de ses parents, de sorte que le croisement de deux individus bien adaptés à leur environnement aura tendance à créer un nouvel individu bien adapté à l'environnement ;
- la *mutation* fait que certaines caractéristiques peuvent apparaître ou disparaître de façon aléatoire, permettant ainsi d'introduire de nouvelles capacités d'adaptation à l'environnement, capacités qui pourront se propager grâce aux mécanismes de sélection et de croisement.

Les algorithmes génétiques reprennent ces mécanismes pour définir une méta-heuristique. L'idée est de faire évoluer une population de combinaisons, par sélection, croisement et mutation, la capacité d'adaptation d'une combinaison étant ici évaluée par la fonction objectif à optimiser. L'algorithme 1 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Initialisation de la population : en général, la population initiale est générée de façon aléatoire, selon une distribution uniforme assurant une bonne diversité des combinaisons.

Algorithme 1: Algorithme génétique

```
Initialiser la population avec un ensemble de combinaisons de E
tant que critères d'arrêt non atteints faire
    Sélectionner des combinaisons de la population
    Créer de nouvelles combinaisons par recombinaison et mutation
    Mettre à jour la population
retourner la meilleure combinaison ayant appartenu à la population
```

Sélection : cette étape consiste à choisir les combinaisons de la population qui seront ensuite candidates pour la recombinaison et la mutation. Il s'agit là de favoriser la sélection des meilleures combinaisons, tout en laissant une petite chance aux moins bonnes combinaisons. Il existe de nombreuses façons de procéder à cette étape de sélection. Par exemple, la sélection par tournoi consiste à choisir aléatoirement deux combinaisons et à sélectionner la meilleure des deux (ou bien à sélectionner une des deux selon une probabilité dépendant de la fonction objectif). La sélection peut également être réalisée selon d'autres critères comme la diversité. Dans ce cas de figure, seuls les individus "distancés" sont autorisés à survivre et à se reproduire.

Recombinaison (croisement) : cette étape vise à créer de nouveaux individus par un mélange de combinaisons sélectionnées. L'objectif est de conduire la recherche dans une nouvelle zone de l'espace où de meilleures combinaisons peuvent être trouvées. A cette fin, la recombinaison doit être bien adaptée à la fonction à optimiser et capable de transmettre de bonnes propriétés des parents vers la combinaison créée. De plus, l'opérateur de recombinaison devrait idéalement permettre de créer des descendants diversifiés. Du point de vue de l'exploration et l'exploitation, la recombinaison est destinée à jouer un rôle de diversification stratégique avec un objectif à long terme de renforcement de l'intensification.

Mutation : cette opération consiste à modifier de façon aléatoire certains composants des combinaisons obtenues par croisement.

Exemple 1 *Pour le voyageur de commerce, un opérateur de recombinaison simple consiste à recopier une sous-séquence du premier parent, et à compléter la permutation en plaçant les villes manquantes dans l'ordre où elles apparaissent dans le deuxième parent. Un opérateur de mutation classique consiste à choisir aléatoirement quelques villes et les échanger.*

Mise à jour de la population : cette étape détermine quelles nouvelles combinaisons doivent devenir membre de la population et quelles anciennes combinaisons de la population doivent être remplacées. La politique de mise-à-jour est essentielle pour maintenir une diversité appropriée de la population, pour prévenir une convergence prématurée du processus de recherche, et pour permettre à l'algorithme de toujours découvrir de nouvelles zones prometteuses dans l'espace de recherche. Ainsi, les décisions sont souvent prises selon des critères liés à la fois à la qualité et à la diversité. Par exemple, une règle bien connue de mise à jour basée uniquement sur la qualité consiste à remplacer la pire des combinaisons de la population, tandis qu'une règle fondée sur la diversité consiste à substituer les anciennes combinaisons de la population par de nouvelles combinaisons similaires, conformément à une mesure de similarité donnée. D'autres critères comme l'age peuvent aussi être considérés.

Critères d'arrêt : le processus d'évolution est itéré, de génération en génération, jusqu'à ce qu'un critère d'arrêt soit atteint. Il peut s'agir d'un nombre maximum de générations, un nombre maximum d'évaluations, un nombre maximum de générations sans améliorer la meilleure solution, une qualité de solution atteinte ou encore une diversité de population inférieure à un seuil donné.

1.2.2 Recherche locale

Une recherche locale explore l'espace des combinaisons de proche en proche, en partant d'une combinaison initiale et en sélectionnant à chaque itération une combinaison voisine de la combinaison courante, obtenue en lui appliquant une transformation élémentaire [33]. L'algorithme 2 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Algorithme 2: Recherche locale

Générer une combinaison initiale $e \in E$
tant que *critères d'arrêt non atteints* **faire**
  Choisir $e' \in v(e)$
  $e \leftarrow e'$
retourner *la meilleure combinaison construite*

Fonction de voisinage : l'algorithme de recherche locale est paramétré par une fonction de voisinage $v : E \rightarrow \mathcal{P}(E)$ définissant l'ensemble des combinaisons que l'on peut explorer à partir d'une combinaison donnée. Étant donnée une combinaison courante $e \in E$, $v(e)$ est un ensemble de combinaisons que l'on peut obtenir en appliquant une modification "élémentaire" à e. On peut généralement considérer différents opérateurs de modification comme, par exemple, changer la valeur d'une variable ou échanger les valeurs de deux variables. Chaque opérateur de modification différent induit un voisinage différent, qui peut contenir un nombre plus ou moins grand de combinaisons plus ou moins similaires à la combinaison courante. Ainsi, le choix de l'opérateur de voisinage a une influence forte sur les performances de l'algorithme. Il est généralement souhaitable que l'opérateur de voisinage permette d'atteindre la combinaison optimale à partir de n'importe quelle combinaison initiale de E, ce qui revient à imposer que le graphe orienté associant un sommet à chaque combinaison de E et un arc (e_i, e_j) à chaque couple de combinaisons telles que $e_j \in v(e_i)$, admette un chemin depuis n'importe lequel de ses sommets jusque la combinaison optimale.

Exemple 2 *Pour le problème du voyageur de commerce, l'opérateur* 2-opt *consiste à supprimer deux arêtes et les remplacer par les deux nouvelles arêtes qui reconnectent les deux chemins créés par la suppression des arêtes. Plus généralement, l'opérateur* k-opt *consiste à supprimer k arêtes mutuellement disjointes et à réassembler les différents morceaux de chemin ainsi créés en ajoutant k nouvelles arêtes de façon à reconstituer un tour complet. Plus k est grand, plus la taille du voisinage est importante.*

Génération de la combinaison initiale : la combinaison à partir de laquelle le processus d'exploration commence est souvent générée de façon aléatoire. Elle est parfois générée en suivant une heuristique constructive gloutonne (voir la section 1.3.1). Lorsque la recherche locale est hybridée avec une autre méta-heuristique, comme par exemple les algorithmes génétiques ou les algorithmes par colonies de fourmis, la combinaison initiale peut être le résultat d'un autre processus de recherche.

Choix du voisin : à chaque itération de la recherche locale, il s'agit de choisir une combinaison dans le voisinage de la combinaison courante. Ce choix d'une combinaison voisine est appelé "mouvement". Il existe un très grand nombre de stratégies de choix. On peut par exemple sélectionner à chaque itération le meilleur voisin [65], c'est-à-dire, celui qui améliore le plus la fonction objectif ou bien le premier voisin trouvé qui améliore la fonction objectif. De telles stratégies "gloutonnes" (aussi appelées "montées de gradients") risquent fort d'être rapidement piégées dans des *optima* locaux, c'est-à-dire, sur des combinaisons dont toutes les voisines sont moins bonnes. Pour s'échapper de ces *optima* locaux, on peut considérer différentes méta-heuristiques, par exemple, pour n'en citer que quelques-unes :

- la marche aléatoire (*random walk*) [64], qui autorise avec une très petite probabilité de sélectionner un voisin de façon complètement aléatoire ;
- le recuit simulé (*simulated annealing*) [1], qui autorise de sélectionner des voisins de moins bonne qualité selon une probabilité qui décroit avec le temps ;

— la recherche taboue (*tabu search*) [23], qui empêche de boucler sur un petit nombre de combinaisons autour des *optima* locaux en mémorisant les derniers mouvements effectués dans une liste taboue et en interdisant les mouvements inverses à ces derniers mouvements ;
— la recherche à voisinage variable [49], qui change d'opérateur de voisinage lorsque la combinaison courante est un optimum local par rapport au voisinage courant.

Répétition du processus de recherche locale : une recherche locale peut être répétée plusieurs fois à partir de combinaisons initiales différentes. Il peut s'agir de combinaisons initiales indépendantes, générées aléatoirement. On parle alors de recherche locale à plusieurs points de départ (*multi-start local search*). Il peut également s'agir de combinaisons initiales obtenues en perturbant une combinaison issue d'un processus de recherche locale précédent. On parle alors de recherche locale itérée (*iterated local search*) [43]. On peut par ailleurs faire plusieurs recherches locales en parallèle, en partant de différentes combinaisons initiales, et redistribuer régulièrement les combinaisons courantes en supprimant les moins bonnes et dupliquant les meilleures selon le principe "aller avec les meilleurs" (*go with the winner*) [3].

1.3 Méta-heuristiques constructives

Les approches constructives construisent une ou plusieurs combinaisons de façon incrémentale, c'est-à-dire, en partant d'une combinaison vide, et en ajoutant des composants de combinaison jusqu'à obtenir une combinaison complète. Ces approches sont dites "basées sur les modèles" dans [77], dans le sens où elles utilisent un modèle, généralement stochastique, pour choisir à chaque itération le prochain composant de combinaison à ajouter à la combinaison en cours de construction.

Il existe différentes stratégies pour choisir les composants à ajouter à chaque itération, les plus connues étant les stratégies gloutonnes aléatoires, décrites en 1.3.1, les algorithmes par estimation de distribution, décrits en 1.3.2 et la méta-heuristique d'optimisation par colonies de fourmis, introduite en 1.3.3.

1.3.1 Algorithmes gloutons aléatoires

Les algorithmes gloutons (*greedy*) construisent une combinaison en partant d'une combinaison vide et en choisissant à chaque itération un composant de combinaison pour lequel une heuristique donnée est maximale. Un algorithme glouton est capable de construire une combinaison très rapidement, les choix effectués n'étant jamais remis en cause. La qualité de la combinaison construite dépend de l'heuristique.

Exemple 3 *Un algorithme glouton pour le problème du voyageur de commerce peut être défini de la façon suivante : partant d'une ville initiale choisie aléatoirement, on se déplace à chaque itération sur la ville non visitée la plus proche de la dernière ville visitée, jusqu'à ce que toutes les villes aient été visitées.*

Les algorithmes gloutons aléatoires (*greedy randomized*) construisent plusieurs combinaisons et adoptent également une stratégie gloutonne pour le choix des composants à ajouter aux combinaisons en cours de construction, mais ils introduisent un peu d'aléatoire afin de diversifier les combinaisons construites. On peut par exemple choisir aléatoirement le prochain composant parmi les k meilleurs ou bien parmi ceux qui sont à moins de α pour cent du meilleur composant [15]. Une autre possibilité consiste à choisir le prochain composant en fonction de probabilités dépendant de la qualité des différents composants [37].

Exemple 4 *Pour le problème du voyageur de commerce, si la dernière ville visitée est i, et si C contient l'ensemble des villes qui n'ont pas encore été visitées, on peut définir la probabilité de visiter la ville $j \in C$ par $p(j) = \frac{[1/d(i,j)]^\beta}{\sum_{k \in C}[1/d(i,k)]^\beta}$. β est un paramètre permettant de régler le niveau d'aléatoire : si $\beta = 0$, alors toutes les villes non visitées ont la même probabilité d'être choisies ; plus on augmente β et plus on augmente la probabilité de choisir les villes les plus proches.*

1.3.2 Algorithmes par estimation de distributions

Les algorithmes par estimation de distribution (*Estimation of Distribution Algorithms*; EDA) [42] sont des algorithmes gloutons aléatoires itératifs : à chaque itération un ensemble de combinaisons est généré selon un principe glouton aléatoire similaire à celui décrit en 1.3.1. Cependant, les EDA exploitent les meilleures combinaisons construites lors des itérations précédentes pour construire de nouvelles combinaisons. L'algorithme 3 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Algorithme 3: Algorithmes par estimation de distributions

Générer une population de combinaisons $P \subseteq E$
tant que *critères d'arrêt non atteints* **faire**
  Construire un modèle probabiliste M en fonction de P
  Générer de nouvelles combinaisons à l'aide de M
  Mettre à jour P en fonction des nouvelles combinaisons
retourner *la meilleure combinaison ayant appartenu à la population*

Génération de la population initiale P : en général la population initiale est générée de façon aléatoire selon une distribution uniforme, et seules les meilleures combinaisons générées sont gardées.

Génération du modèle probabiliste M : différents types de modèles probabilistes, plus ou moins simples, peuvent être considérés. Le modèle le plus simple, appelé PBIL [5], est basé sur la probabilité d'apparition de chaque composant sans tenir compte d'éventuelles relations de dépendances entre les composants. Dans ce cas, on calcule pour chaque composant sa fréquence d'apparition dans la population et on définit la probabilité de sélection de ce composant proportionnellement à sa fréquence. D'autres modèles plus fins, mais aussi plus coûteux, utilisent des réseaux bayésiens [54]. Les relations de dépendance entre composants sont alors représentées par les arcs d'un graphe auxquels sont associées des distributions de probabilités conditionnelles.

Exemple 5 *Pour le problème du voyageur de commerce, si la dernière ville visitée est i, et si C contient l'ensemble des villes qui n'ont pas encore été visitées, on peut définir la probabilité de visiter la ville $j \in C$ par $p(j) = \frac{freq_P(i,j)}{\sum_{k \in C} freq_P(i,k)}$, où $freq_P(i,j)$ donne le nombre de combinaisons de la population P contenant l'arête (i,j). Ainsi, la probabilité de choisir j est d'autant plus forte que la population contient de combinaisons utilisant l'arête (i,j).*

Génération de nouvelles combinaisons à l'aide de M : les nouvelles combinaisons sont construites selon un principe glouton aléatoire, les probabilités de sélection des composants étant données par le modèle probabiliste M.

Mise à jour de la population : en général, seules les meilleures combinaisons, appartenant à la population courante ou aux nouvelles combinaisons générées, sont conservées dans la population de l'itération suivante. Il est possible de maintenir par ailleurs une certaine diversité des combinaisons gardées dans la population.

1.3.3 Optimisation par colonies de fourmis

Il existe un parallèle assez fort entre l'optimisation par colonies de fourmis (*Ant Colony Optimization*; ACO) et les algorithmes par estimation de distribution [77]. Ces deux approches utilisent un modèle probabiliste glouton pour générer des combinaisons, ce modèle évoluant en fonction des combinaisons précédemment construites dans un processus itératif d'apprentissage. L'originalité et la contribution essentielle d'ACO est de s'inspirer du comportement collectif des fourmis pour faire évoluer le modèle probabiliste. Ainsi, la probabilité de choisir un composant est définie proportionnellement à une quantité

de phéromone représentant l'expérience passée de la colonie concernant le choix de ce composant. Cette quantité de phéromone évolue par la conjugaison de deux mécanismes : un mécanisme de renforcement des traces de phéromone associées aux composants des meilleures combinaisons, visant à augmenter la probabilité de sélection de ces composants ; et un mécanisme d'évaporation, visant à privilégier les expériences récentes par rapport aux expériences plus anciennes. L'algorithme 4 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Algorithme 4: Optimisation par colonies de fourmis

```
Initialiser les traces de phéromone à τ_0
tant que les conditions d'arrêt ne sont pas atteintes faire
    Construction de combinaisons par les fourmis
    Mise à jour de la phéromone
retourner la meilleure combinaison construite
```

Structure phéromonale : La phéromone est utilisée pour biaiser les probabilités de choix des composants de combinaisons lors de la construction gloutonne aléatoire d'une combinaison. Un point crucial pour la performance de l'algorithme réside donc dans le choix de la structure phéromonale, c'est-à-dire, dans le choix des données sur lesquelles des traces de phéromone seront déposées. Au début de l'exécution d'un algorithme ACO, toutes les traces de phéromone sont initialisées à une valeur τ_0 donnée.

Exemple 6 *Pour le problème du voyageur de commerce, une trace τ_{ij} est associée à chaque paire (i, j) de villes. Cette trace représente l'expérience passée de la colonie concernant le fait de visiter consécutivement les villes i et j.*

Construction de combinaisons par les fourmis : A chaque cycle d'un algorithme ACO, chaque fourmi construit une combinaison selon un principe glouton aléatoire similaire à celui introduit en 1.3.1. Partant d'une combinaison vide, ou d'une combinaison contenant un premier composant de combinaison choisi aléatoirement ou selon une heuristique donnée, la fourmi ajoute un nouveau composant de combinaison à chaque itération, jusqu'à ce que la combinaison soit complète. A chaque itération, le prochain composant de combinaison est choisi selon une règle de transition probabiliste : étant donnés un début de combinaison S, et un ensemble C de composants de combinaison pouvant être ajoutés à S, la fourmi choisit le composant $i \in C$ selon la probabilité :

$$p_S(i) = \frac{[\tau_S(i)]^\alpha \cdot [\eta_S(i)]^\beta}{\sum_{j \in C} [\tau_S(j)]^\alpha \cdot [\eta_S(j)]^\beta} \tag{1.1}$$

où $\tau_S(i)$ est le facteur phéromonal associé au composant de combinaison i par rapport au début de combinaison S (la définition de ce facteur dépend de la structure phéromonale choisie), $\eta_S(i)$ est le facteur heuristique associé à i par rapport au début de combinaison S (la définition de ce facteur dépend du problème), et α et β sont deux paramètres permettant de moduler l'influence relative des deux facteurs dans la probabilité de transition. En particulier, si $\alpha = 0$ alors le facteur phéromonal n'intervient pas dans le choix des composants de combinaison et l'algorithme se comporte comme un algorithme glouton aléatoire pur. A l'inverse, si $\beta = 0$ alors seules les traces de phéromone sont prises en compte pour définir les probabilités de choix.

Exemple 7 *Pour le problème du voyageur de commerce, le facteur phéromonal $\tau_S(i)$ est défini par la quantité de phéromone τ_{ki} déposée entre la dernière ville k ajoutée dans S et la ville candidate i. Le facteur heuristique est inversement proportionnel à la longueur de la route joignant la dernière ville ajoutée dans S à la ville candidate i.*

Mise à jour de la phéromone : Une fois que chaque fourmi a construit une combinaison, les traces de phéromone sont mises à jour. Elles sont tout d'abord diminuées en multipliant chaque trace par un

facteur $(1 - \rho)$, où $\rho \in [0; 1]$ est le taux d'évaporation. Ensuite, certaines combinaisons sont "récompensées" par un dépôt de phéromone. Il existe différentes stratégies concernant le choix des combinaisons à récompenser. On peut récompenser toutes les combinaisons construites lors du dernier cycle, ou bien seulement les meilleures combinaisons du cycle, ou encore la meilleure combinaison trouvée depuis le début de l'exécution. Ces différentes stratégies influent sur l'intensification et la diversification de la recherche. En général, la phéromone est déposée en quantité proportionnelle à la qualité de la combinaison récompensée. Elle est déposée sur les traces de phéromone associées à la combinaison à récompenser ; ces traces dépendent de l'application et de la structure phéromonale choisie.

Exemple 8 *Par exemple, pour le problème du voyageur de commerce, on dépose de la phéromone sur chaque trace τ_{ij} telle que les villes i et j ont été visitées consécutivement lors de la construction de la combinaison à récompenser.*

1.4 Méta-heuristiques hybrides

Les différentes méta-heuristiques présentées en 1.2 et 1.3 peuvent être combinées pour former de nouvelles méta-heuristiques. Deux exemples classiques de telles hybridations sont décrits en 1.4.1 et 1.4.2.

1.4.1 Algorithmes mémétiques

Les algorithmes mémétiques combinent une méthode à population (algorithmes évolutionnaires) et la recherche locale [50, 51]. Cette hybridation vise à tirer profit du potentiel de diversification fourni par l'approche à population et de la capacité d'intensification offerte par la recherche locale.

Un algorithme mémétique peut être considéré comme un algorithme génétique (tel que celui décrit dans l'algorithme 1) étendu par un processus de recherche locale. Tout comme pour un algorithme génétique, une population de combinaisons est utilisée pour échantillonner l'espace de recherche, et un opérateur de recombinaison permet de créer de nouvelles combinaisons à partir de deux ou plusieurs combinaisons de la population. Des mécanismes de sélection et de remplacement permettent de déterminer les combinaisons qui sont recombinées ainsi que celles qui sont éliminées de la population. Cependant, l'opérateur de mutation des algorithmes génétiques est remplacé par un processus de recherche locale, qui peut être considéré comme un processus guidé de macro-mutation. L'objectif de la recherche locale est d'améliorer la qualité des combinaisons générées. En comparaison avec l'opérateur de recombinaison, la recherche locale joue essentiellement le rôle de l'intensification de la recherche en exploitant les chemins de recherche délimités par l'opérateur de voisinage considéré. Comme la recombinaison, la recherche locale est un autre élément clé de l'approche mémétique.

1.4.2 Hybridation entre approches perturbatives et approches constructives

Les approches perturbatives et constructives sont très facilement hybridables : à chaque itération, une ou plusieurs combinaisons sont construites selon un principe glouton aléatoire (qui peut être guidé par EDA ou ACO), puis certaines de ces combinaisons sont améliorées par une procédure de recherche locale. Cette hybridation est connue sous le nom de GRASP (*Greedy Randomized Adaptive Search Procedure*) [58]. Un point important concerne le choix de l'opérateur de voisinage considéré et de la stratégie de choix des mouvements de l'approche perturbative. Il s'agit là de trouver un compromis entre le temps mis par la recherche locale pour améliorer les combinaisons, et la qualité des améliorations. Typiquement, on choisira d'effectuer une simple recherche locale gloutonne, améliorant les combinaisons construites jusqu'à arriver sur un optimum local.

Notons que les approches EDA et ACO les plus performantes comportent bien souvent une telle hybridation avec de la recherche locale.

1.5 Intensification *versus* diversification

Pour les différentes approches heuristiques présentées dans ce chapitre, un point critique dans la mise au point de l'algorithme consiste à trouver un juste compromis entre deux tendances duales :

— il s'agit d'une part d'intensifier l'effort de recherche vers les zones les plus "prometteuses" de l'espace des combinaisons, c'est-à-dire, aux alentours des meilleures combinaisons trouvées ;

— il s'agit par ailleurs de diversifier l'effort de recherche de façon à être capable de découvrir de nouvelles zones contenant (potentiellement) de meilleures combinaisons.

La façon d'intensifier/diversifier l'effort de recherche dépend de l'approche considérée et se fait en modifiant certains paramètres de l'algorithme. Pour les approches perturbatives, l'intensification de la recherche se fait en favorisant l'exploration des meilleurs voisins : pour les algorithmes génétiques, on adopte des stratégies de sélection et de remplacement élitistes qui favorisent la reproduction des meilleures combinaisons ; pour la recherche locale, on adopte des stratégies gloutonnes qui favorisent la sélection des meilleurs voisins. La diversification d'une approche perturbative se fait généralement en introduisant une part d'aléatoire : pour les algorithmes génétiques, la diversification est essentiellement assurée par la mutation ; pour la recherche locale, la diversification est assurée en autorisant avec une faible probabilité la recherche à choisir des voisins de moins bonne qualité.

Pour les approches constructives, l'intensification de la recherche se fait en favorisant, à chaque étape de la construction, le choix de composants ayant appartenu aux meilleurs combinaisons précédemment construites. La diversification se fait en introduisant une part d'aléatoire permettant de choisir avec une faible probabilité de moins bons composants.

En général, plus on intensifie la recherche d'un algorithme en l'incitant à explorer les combinaisons proches des meilleures combinaisons trouvées, et plus il converge rapidement, trouvant de meilleures combinaisons plus tôt. Cependant, si l'on intensifie trop la recherche, l'algorithme risque fort de "stagner" autour d'*optima* locaux, concentrant tout son effort de recherche sur une petite zone autour d'une assez bonne combinaison, sans plus être capable de découvrir de nouvelles combinaisons. Notons ici que le bon équilibre entre intensification et diversification dépend clairement du temps de calcul dont on dispose pour résoudre le problème. Plus ce temps est petit et plus on a intérêt à favoriser l'intensification pour converger rapidement, quitte à converger vers des combinaisons de moins bonne qualité.

Le bon équilibre entre intensification et diversification dépend également de l'instance à résoudre, ou plus particulièrement de la topologie de son paysage de recherche. En particulier, une recherche fortement intensifiée donnera des résultats d'autant meilleurs que le paysage de recherche comporte peu d'*optima* locaux et qu'on observe une forte corrélation entre la qualité d'une combinaison et sa distance à la combinaison optimale ; on parle alors de paysages de recherche de type "massif central". En revanche, quand le paysage de recherche comporte un très grand nombre d'*optima* locaux répartis uniformément, les meilleurs résultats seront obtenus par une recherche fortement diversifiée... pour ne pas dire une recherche complètement aléatoire !

Différentes approches ont proposé d'adapter dynamiquement le paramétrage au cours de la résolution, en fonction de l'instance à résoudre. On parle alors de recherche réactive [6]. Par exemple, l'approche taboue réactive de [7] ajuste automatiquement la longueur de la liste taboue, tandis que la recherche locale *IDwalk* de [52] adapte automatiquement le nombre de voisins considérés à chaque mouvement.

1.6 Applications en intelligence artificielle

Les méta-heuristiques ont été appliquées avec succès à un très grand nombre de problèmes classiques NP-difficiles et à des applications réelles dans des domaines extrêmement variés. Dans cette partie, nous évoquons leur application à deux problèmes centraux en intelligence artificielle : la satisfiabilité de formules booléennes (SAT) et la satisfaction des contraintes (CSP).

1.6.1 Satisfiabilité de formules booléennes

SAT est un des problèmes centraux en intelligence artificielle. Il s'agit, pour une formule booléenne, de déterminer un modèle, à savoir une affectation d'une valeur booléenne (vrai ou faux) à chaque variable telle que la valuation de la formule soit vraie. Pour des raisons pratiques, on suppose que la formule booléenne à traiter est donnée sous la forme CNF même si d'un point de vue général, la forme CNF n'est pas nécessaire pour appliquer une métaheuristique.

Remarquons que SAT n'est pas un problème d'optimisation et ne possède pas de fonction objectif à optimiser. Les méta-heuristiques étant généralement conçues pour résoudre des problèmes d'optimisation, elles considèrent un problème plus général appelé MAXSAT et dont l'objectif est de trouver la valuation maximisant le nombre de clauses satisfaites. Dans ce contexte, le résultat d'un algorithme méta-heuristique à une instance MAXSAT peut être de deux sortes : soit l'assignation retournée à la fin satisfait toutes les clauses de la formule, auquel cas une solution est trouvée à l'instance SAT, soit elle ne satisfait pas toutes les clauses, auquel cas on ne peut pas savoir si l'instance SAT est satisfiable ou non.

L'espace de recherche d'une instance SAT est défini par l'ensemble des affectations de valeurs booléennes à l'ensemble des variables. Ainsi, pour une instance ayant n variables, la taille de l'espace est 2^n. La fonction objectif à optimiser évalue le nombre de clauses satisfaites. Cette fonction introduit un ordre total sur les combinaisons de l'espace de recherche. On peut également considérer la fonction objectif duale, évaluant le nombre de clauses non satisfaites, et correspondant à une fonction de pénalité à minimiser : chaque clause non satisfaite a un poids de pénalité égal à 1, et une combinaison dont l'évaluation est 0 est une solution. Cette fonction peut être affinée par un mécanisme de pénalité dynamique ou encore une extension incluant d'autres informations que le nombre de clauses non satisfaites.

Beaucoup de travaux ont été réalisés depuis une vintaine d'années pour la résolution pratique de SAT. Les défis régulièrement organisés par la communauté scientifique autour de SAT dynamisent continuellement les activités de production, évaluation et comparaison d'algorithmes de résolution pour SAT. Il en résulte un très grand nombre de contributions qui améliorent chaque année la performance et la robustesse des algorithmes, notamment de recherche locale stochastique (RLS).

Recherche locale stochastique

Les algorithmes RLS dédiés à SAT considèrent généralement la même fonction de voisinage : deux combinaisons sont voisines si la distance de Hamming entre elles est exactement de 1. Le passage d'une combinaison à une autre est donc défini par le choix de la variable à modifier. Pour une formule à n variables, la taille du voisinage est égale à n. Ce choix peut être limité à l'ensemble des variables contenues dans au moins une clause falsifiée, donnant ainsi un voisinage plus restreint de taille variable.

Les algorithmes RLS dédiés à SAT diffèrent essentiellement dans les techniques mises en oeuvre pour trouver un bon compromis entre d'une part une exploration de l'espace de recherche suffisamment large pour réduire le risque de stagner dans une région dépourvue de solution, et d'autre part l'exploitation des informations disponibles pour découvrir une solution dans la région en cours d'exploration. Pour classer ces différents algorithmes, nous pouvons considérer d'une part la manière dont les contraintes sont exploitées, d'autre part la manière dont l'historique de recherche est utilisé [4].

Exploitation de la structure de l'instance. La structure de l'instance à résoudre est définie par les clauses de la formule. On peut distinguer trois niveaux d'exploitation de cette structure.

La recherche peut être simplement *guidée par la fonction objectif*, de sorte que seul le nombre de clauses falsifiées par la combinaison courante est pris en compte. L'exemple typique est l'algorithme emblématique GSAT [65] qui, à chaque itération, sélectionne aléatoirement une configuration voisine parmi celles minimisant le nombre de clauses falsifiées. Cette stratégie de descente gloutonne constitue une technique d'exploitation agressive qui peut être facilement piégée dans les optima locaux. Pour contourner ce problème, GSAT utilise la simple technique de diversification basée sur le redémarrage de la recherche à partir d'une nouvelle configuration initiale au bout d'un nombre fixe d'itérations. Plusieurs variantes de GSAT comme CSAT et TSAT [22] ainsi que le recuit simulé [70] utilisent ce même principe de minimisation du

nombre de clauses falsifiées.

La recherche peut être également *guidée par les conflits*, de sorte que les clauses falsifiées sont explicitement prises en compte. Cette approche est particulièrement utile quand le voisinage ne contient plus de combinaison améliorant la valeur d'objectif. Pour mieux orienter la recherche, on peux recourir à d'autres informations obtenues via, par exemple, une analyse des clauses falsifiées. L'archétype de ce type d'algorithme est WALKSAT [47] qui, à chaque itération, choisit aléatoirement une clause parmi celles falsifiées par la combinaison courante. Une heuristique est ensuite utilisée pour choisir une des variables de cette clause, dont la valeur sera modifiée pour obtenir la nouvelle combinaison courante. L'algorithme GWSAT (GSAT with random walk) [63] utilise aussi une marche aléatoire guidée par les clauses falsifiées qui, à chaque itération, modifie une variable appartenant à une clause falsifiée avec une probabilité p (appelée bruit) et utilise la stratégie de descente de GSAT avec une probabilité $1 - p$.

Enfin, *l'exploitation déductive des contraintes* permet d'utiliser des règles de déduction pour modifier la combinaison courante. C'est le cas d'UNITWALK [31], qui utilise la résolution unitaire pour déterminer la valeur de certaines variables de la combinaison courante d'après les valeurs d'autres variables fixées par recherche locale. Un autre type d'approche déductive est utilisé dans un algorithme récent nommé NON-CNF ADAPT NOVELTY [55], qui analyse la formule à traiter pour y rechercher des liens de dépendance entre variables.

Exploitation de l'historique de recherche. Pour classer les approches RLS dédiées à SAT, on peut également considérer la manière dont est exploité l'historique des actions réalisées depuis le début de la recherche et, le cas échéant, de leurs effets. On peut distinguer trois niveaux d'exploitation de cet historique.

Pour les *algorithmes Markoviens (ou sans mémoire)*, le choix de chaque nouvelle combinaison ne dépend que de la combinaison courante. Les algorithmes GSAT, GRSAT, WALKSAT/SKC, de même que l'algorithme plus général du recuit simulé sont des exemples typiques d'algorithmes RLS sans mémoire. S'ils ont l'avantage de minimiser le temps de traitement nécessaire à chaque itération, ils ont été supplantés en pratique, au cours de la dernière décennie, par les algorithmes à mémoire.

Pour les *algorithmes avec mémoire à court terme*, le choix de chaque nouvelle combinaison prend en compte un historique de tout ou partie des modifications des valeurs des variables. Les solveurs SAT inspirés de la méthode taboue, tels que WALKSAT/TABU [47] ou TSAT [46] sont des exemples typiques. Certains algorithmes comme WALKSAT, NOVELTY et RNOVELY, et G2WSAT [34] intègrent également l'*ancienneté* dans le critère de choix. Typiquement, ce critère est utilisé pour privilégier parmi plusieurs variables candidates celle qui a fait l'objet de la modification la plus ancienne.

Pour les *algorithmes avec mémoire à long terme*, le choix de chaque nouvelle combinaison dépend des choix réalisés depuis le début de la recherche et de leurs effets, notamment en terme de clauses satisfaites et falsifiées. L'idée est d'utiliser un mécanisme d'apprentissage pour tirer parti des échecs et accélérer la recherche. En pratique, l'historique des clauses falsifiées est exploité en associant des poids aux clauses, l'objectif étant de diversifier la recherche en l'obligeant à prendre de nouvelles directions. Un premier exemple est weighted-GSAT [62] où, après chaque modification d'une variable, les poids des clauses falsifiées sont incrémentés. Le score utilisé par l'heuristique de recherche est simplement la somme des poids des clauses falsifiées. Comme le poids des clauses souvent falsifiées augmente, le processus tend à privilégier la modification des variables appartenant à ces clauses, jusqu'à ce que l'évolution des poids des autres clauses influence à nouveau la recherche. D'autres exemples comprennent notamment DLM (*discrete lagrangian method*) [66], DDWF (*Divide and Distribute Fixed Weight*) [36], PAWS (*Pure Additive Weighting Scheme*) [71], ESG (*Exponentiated Subgradient Algorithm*) [61], SAPS (*Scaling and Probabilistic Smoothing*) [35] et WV (*weighted variables*) [57].

Algorithmes à base de population

Les algorithmes génétiques ont été appliqués à plusieurs reprises à SAT [38, 75, 10, 29]. Comme les algorithmes de recherche locale, ces AG adoptent une représentation de l'espace de recherche sous forme d'affectation de valeurs 0/1 à l'ensemble de variables. Par contre, les premiers AG traitent directement

des formules logiques générales, non limitées à la forme CNF. Malheureusement, les résultats obtenus par ces algorithmes sont généralement décevants quand ils sont appliqués à des benchmarks sous forme CNF.

Le premier AG traitant les formules CNF est présenté [16]. L'originalité de cet algorithme est l'utilisation d'un croisement spécifique et original qui tente d'exploiter la sémantique des deux parents à croiser. Étant donnés deux parents p_1 et p_2, on examine les clauses qui sont satisfaites par un parent, mais falsifiées par l'autre parent. Lorsqu'une clause c est vraie chez le parent p_1 et fausse chez le parent p_2, les valeurs de vérité de toutes variables de cette clause sont directement transmises à l'enfant. L'algorithme mémétique intégrant ce croisement et une recherche taboue a donné des résultats très intéressants lors du deuxième challenge d'implantation DIMACS.

Un autre algorithme génétique plus récent est GASAT [17]. Comme l'algorithme de [16], GASAT accorde une importance prépondérante à la conception d'un croisement sémantique. Ainsi, une nouvelle classe de croisements est introduite visant à corriger les "erreurs" des parents et combiner leurs bonnes caractéristiques. Par exemple, si une clause c est fausse chez deux parents de bonne qualité, on peut forcer cette clause à être vraie chez l'enfant par l'inversion d'une variable de c. L'argument intuitif qui justifie ce croisement consiste à considérer une telle clause comme difficilement satisfiable autrement et par conséquent préférer la rendre vraie par force. De même, des mécanismes de recombinaison sont développés pour exploiter la sémantique liée à une clause qui est rendue vraie simultanément par les deux parents. Muni de ce type de croisement et d'un algorithme de recherche taboue, GASAT se montre très compétitif sur certaines catégories de benchmarks.

FlipGA [59] est un algorithme génétique qui repose sur un croisement standard (uniforme) qui génère, par un mélange équiprobable des valeurs des deux parents, un enfant auquel est appliqué un algorithme de recherche locale.

D'autres AG sont présentés dans [13, 25, 60], mais il s'agit en réalité d'algorithmes de recherche locale puisque la population est réduite à une seule combinaison. Leur intérêt réside dans les techniques utilisées pour affiner la fonction d'évaluation de base (nombre de clauses falsifiées) par un ajustement dynamique durant le processus de recherche. On trouve dans [26] une comparaison expérimentale de ces quelques algorithmes génétiques dédiés à SAT. Leur intérêt pratique reste néanmoins à démontrer puisqu'ils se sont rarement confrontés directement aux algorithmes RLS de l'état de l'art.

On remarque que dans ces algorithmes à base de population, la recherche locale constitue souvent un élément indispensable dans le but de renforcer la capacité d'intensification. Le rôle du croisement peut être différent selon qu'il est entièrement aléatoire [59] auquel cas il sert essentiellement à diversifier la recherche, ou qu'il est basé sur la sémantique du problème [16, 17] auquel cas il permet à la fois de diversifier et d'intensifier la recherche.

1.6.2 Problèmes de satisfaction de contraintes

Un problème de satisfaction de contraintes (*Constraint Satisfaction Problem*; CSP) est un problème modélisé sous la forme d'un ensemble de contraintes posées sur des variables, chacune de ces variables prenant ses valeurs dans un domaine. Résoudre un CSP consiste à affecter des valeurs aux variables, de telle sorte que toutes les contraintes soient satisfaites.

Comme pour SAT, on considère généralement le problème d'optimisation MaxCSP dont l'objectif est de trouver l'affectation complète (affectant une valeur à chaque variable) maximisant le nombre de contraintes satisfaites. L'espace de recherche est défini par l'ensemble des affectations complètes, tandis que la fonction objectif à maximiser est définie par le nombre de contraintes satisfaites.

Algorithmes génétiques

Dans sa forme la plus simple, un algorithme génétique pour la résolution de CSP utilise une population d'affectations complètes qui sont recombinées par simple croisement, la mutation consistant à changer la valeur d'une variable. [9] propose une comparaison expérimentale de onze algorithmes génétiques pour la résolution de CSP binaires quelconques. Ils ont montré que les trois meilleurs algorithmes (*Heuristics GA*

version 3, *Stepwise Adaptation of Weights* et *Glass-Box*) ont des performances équivalentes et sont significativement meilleurs que les huit autres algorithmes. Cependant, ces meilleurs algorithmes génétiques ne sont clairement pas compétitifs, ni avec les approches exactes basées sur une recherche arborescente, ni avec d'autres approches heuristiques, comme par exemple la recherche locale ou l'optimisation par colonies de fourmis [72].

D'autres algorithmes génétiques ont été proposés pour des problèmes particuliers de satisfaction de contraintes. Ces algorithmes spécifiques exploitent des connaissances sur les contraintes du problème à résoudre pour définir des opérateurs de croisement et de mutation mieux adaptés, permettant d'obtenir de meilleurs résultats. On peut citer notamment [76] qui obtient des résultats compétitifs pour un problème d'ordonnancement de voitures.

Recherche locale

Il existe de très nombreux travaux concernant la résolution de CSP par des techniques de recherche locale, et cette approche permet généralement d'obtenir d'excellents résultats. Les algorithmes de recherche locale pour la résolution de CSP diffèrent essentiellement par le voisinage et la stratégie de choix considérés.

Voisinage. En général, les combinaisons explorées sont des affectations totales, et un mouvement consiste à modifier la valeur d'une variable, de sorte que le voisinage d'une affectation totale est l'ensemble des affectations que l'on peut obtenir en changeant la valeur d'une variable. En fonction de la nature des contraintes, on peut considérer d'autres voisinages, comme par exemple le voisinage consistant à échanger les valeurs de deux variables lorsque ces variables sont impliquées dans une contrainte de permutation imposant qu'une suite de variables prenne pour valeur une permutation d'une suite donnée de valeurs.

Certains travaux considèrent des voisinages non pas uniquement entre affectations totales, mais également entre affectations partielles. En particulier, l'approche *decision repair* proposée dans [39] combine des techniques de filtrage telles que celles décrites au chapitre **??** avec une recherche locale sur l'espace des affectations partielles. Etant donnée une affectation partielle courante A, si le filtrage détecte une incohérence alors le voisinage de A est l'ensemble des affectations résultant de la suppression d'un couple variable/valeur de A, sinon le voisinage de A est l'ensemble des affectations résultant de l'ajout d'un couple variable/valeur à A.

Stratégies de choix. Un grand nombre de stratégies différentes pour le choix du prochain mouvement ont été proposées. La stratégie *min-conflict* [48] consiste à choisir aléatoirement une variable impliquée dans au moins une violation de contrainte et à choisir pour cette variable la valeur qui minimise le nombre de violations. Cette stratégie gloutonne, célèbre pour avoir permis la résolution du problème des reines pour un million de reines, a cependant tendance à rester piégée dans des *optima* locaux. Une façon simple et classique pour permettre à la stratégie *min-conflict* de sortir de ces minima locaux consiste à la combiner avec la stratégie marche aléatoire [73]. D'autres strategies pour le choix du couple variable/valeur sont étudiées dans [27].

La recherche locale utilisant une stratégie taboue (TabuCSP) [18] obtient d'excellents résultats pour la résolution de CSPs binaires. Partant d'une affectation initiale, l'idée est de choisir à chaque itération le mouvement non tabou qui augmente le plus le nombre de contraintes satisfaites. Un mouvement consiste à changer la valeur d'une *variable en conflit*, et il est dit tabou si le couple variable/valeur a déjà été choisi lors des k derniers mouvements (k étant la longueur de la liste taboue). Un critère d'aspiration est ajouté, permettant à un mouvement tabou d'être sélectionné s'il amène à une affectation satisfaisant plus de contraintes que la meilleure affectation trouvée depuis le début.

Le langage COMET [30] permet de concevoir rapidement des algorithmes de recherche locale pour la résolution de CSP. Il introduit notamment pour cela le concept de variable incrémentale permettant une évaluation incrémentale des voisinages. D'autres systèmes génériques de résolutions de CSP fondés sur la recherche locale sont présentés dans [11, 53, 28, 20].

Algorithmes de construction gloutonne

On peut construire une affectation totale pour un CSP selon le principe glouton suivant : partant d'une affectation vide, on sélectionne à chaque itération une variable non affectée et une valeur à affecter à cette variable, jusqu'à ce que toutes les variables soient affectées. Pour choisir à chaque étape une variable et une valeur, on peut utiliser les heuristiques d'ordre généralement utilisées par les approches exactes décrites au chapitre **??**. Un exemple bien connu d'instanciation de ce principe est l'algorithme DSATUR [8] pour résoudre le problème de coloriage de graphes. Cet algorithme construit une combinaison de façon gloutonne en choisissant à chaque itération le sommet non colorié ayant le plus grand nombre de voisins coloriés avec des couleurs différentes (le voisin le plus saturé). Les *ex aequo* sont départagés en choisissant le sommet de plus fort degré. Le sommet choisi est alors colorié par la plus petite couleur possible.

Optimisation par colonies de fourmis

La méta-heuristique ACO a été appliquée à la résolution de CSP dans [69, 68]. L'idée est de construire des affectations complètes selon un principe glouton aléatoire : partant d'une affectation vide, on sélectionne à chaque itération une variable non affectée et une valeur à affecter à cette variable, jusqu'à ce que toutes les variables soient affectées. La contribution principale d'ACO est de fournir une heuristique de choix de valeur : celle-ci est choisie selon une probabilité qui dépend d'un facteur heuristique (inversement proportionnel au nombre de nouvelles violations introduites par la valeur), et d'un facteur phéromonal qui reflète l'expérience passée concernant l'utilisation de cette valeur. La structure phéromonale est générique et peut être utilisée pour résoudre n'importe quel CSP. Elle associe une trace phéromonale à chaque variable x_i et chaque valeur v_i pouvant être affectée à x_i : intuitivement, cette trace représente l'expérience passée de la colonie concernant le fait d'affecter la variable x_i à la valeur v_i. D'autres structures phéromonales ont été proposées pour la résolution de CSP particuliers tels que, par exemple, les problèmes d'ordonnancement de voitures [67] ou de sac-à-dos multidimensionnels [2].

L'optimisation par colonies de fourmis a été intégrée dans les bibliothèques de résolution de CSP et de problèmes d'optimisation sous contraintes IBM/Ilog Solver et CP Optimizer [40, 41]. Cette intégration permet d'utiliser des langages de haut niveau pour définir de façon déclarative le problème à résoudre, la résolution du problème étant prise en charge de façon automatique par un algorithme ACO intégré au langage.

1.7 Discussion

Les méta-heuristiques ont été utilisées avec succès pour résoudre de nombreux problèmes d'optimisation difficiles, et ces approches obtiennent bien souvent de très bons résultats lors des compétitions, que ce soit pour la résolution de problèmes réels tels que ceux posés par la société française de Recherche Opérationnelle et d'Aide à la Décision (challenges ROADEF), ou pour la résolution de problèmes plus académiques tels que le problème SAT.

La variété des méta-heuristiques pose naturellement le problème du choix de celle qui sera la plus efficace pour résoudre un nouveau problème. Evidemment, cette question est complexe et se rapproche d'une quête fondamentale en intelligence artificielle, à savoir la résolution automatique de problèmes. Nous n'abordons dans cette discussion que quelques éléments de réponse.

En particulier, le choix d'une méta-heuristique dépend de la pertinence de ses mécanismes de base pour le problème considéré, c'est-à-dire, de leur capacité à générer de bonnes combinaisons :

- Pour les AG, l'opérateur de recombinaison doit être capable d'identifier les bons motifs qui doivent être assemblés et hérités par recombinaison. Par exemple, pour la coloration de graphes, un motif intéressant est un groupe de sommets de même couleur et partagé par de bonnes combinaisons ; ce type d'information a permis la conception de croisements très performants avec deux parents [12, 19] ou avec plusieurs parents [21, 45, 44, 56].
- Dans le cas de la recherche locale, le voisinage doit favoriser la construction de meilleurs combinaisons. Par exemple, pour les problèmes SAT et CSP, le voisinage centré sur les variables en conflit

(voir Section 1.6.2) est pertinent car il favorise l'élimination des conflits.
— Pour ACO, la structure phéromonale doit être capable de guider la recherche vers de meilleures combinaisons. Par exemple, pour les CSP, la structure phéromonale associant une trace de phéromone à chaque couple variable/valeur est pertinente car elle permet d'apprendre progressivement quelles sont les bonnes valeurs à affecter à chaque variable.

Un autre point crucial réside dans la complexité en temps des opérateurs élémentaires de la méta-heuristique considérée (recombinaison et mutation pour les AG, mouvement pour la recherche locale, ajout d'un composant de combinaison pour les approches constructives, ...). Cette complexité dépend des structures de données utilisées pour représenter les combinaisons. Ainsi, le choix d'une méta-heuristique dépend de l'existence de structures de données qui permettent une évaluation incrémentale de la fonction d'évaluation après chaque application des opérateurs élémentaires de cette méta-heuristique.

D'autres éléments déterminants pour les performances sont partagés par toutes les méta-heuristiques. En particulier, la fonction d'évaluation (ne pas confondre avec la fonction objectif du problème) est un point clé car elle mesure la pertinence d'une combinaison. Par exemple, pour les CSP, la fonction d'évaluation peut simplement compter le nombre de contraintes non-satisfaites mais, dans ce cas, l'importance relative des contraintes n'est pas distinguée. Un affinement intéressant consiste à introduire dans la fonction d'évaluation une pénalité pour quantifier le degré de violation d'une contrainte [20]. Cette fonction peut être encore affinée, comme dans le cas du problème SAT (voir la section 1.6.1), par des pondérations qui s'ajustent dynamiquement en fonction de l'historique des violations de chaque contrainte.

Enfin, les méta-heuristiques ont bien souvent de nombreux paramètres qui ont une influence prépondérante sur leur efficacité. Ainsi, on peut voir le problème de la mise au point de toute méta-heuristique comme un problème de configuration : il s'agit de choisir les bonnes briques à combiner (opérateurs de recombinaison ou de mutation, voisinages, stratégies de choix des mouvements, facteurs heuristiques, ...) ainsi que le bon paramétrage (taux de mutation, d'évaporation, bruit, longueur de la liste taboue, poids des facteurs phéromonaux et heuristiques, ...). Une approche prometteuse pour résoudre ce problème de configuration consiste à utiliser des algorithmes de configuration automatiques pour concevoir l'algorithme le mieux adapté à un ensemble d'instances données [74].

Bibliographie

[1] Emile H.L. Aarts and Jan H.M. Korst. *Simulated annealing and Boltzmann machines : a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Chichester, U.K., 1989.

[2] I. Alaya, C. Solnon, and K. Ghedira. Optimisation par colonies de fourmis pour le problème du sac-à-dos multi-dimensionnel. *Techniques et Sciences Informatiques (TSI)*, 26(3-4) :271–390, 2007.

[3] David Aldous and Umesh V. Vazirani. "go with the winners" algorithms. In *35th Annual Symposium on Foundations of Computer Science*, pages 492–501, 1994.

[4] Oliver Bailleux and Jin-Kao Hao. Algorithmes de recherche stochastiques. In Lakhdar Saïs, editor, *Problème SAT : procès et défis*, chapter 5. Hermès - Lavoisier, 2008.

[5] Shumeet Baluja. Population-based incremental learning : A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[6] Roberto Battiti, Mauro Brunato, and Franco Mascia. *Reactive Search and Intelligent Optimization*. Operations research/Computer Science Interfaces. Springer Verlag, 2008. in press.

[7] Roberto Battiti and Marco Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4) :610–637, 2001.

[8] Daniel Brélaz. New methods to color the vertices of a graph. *journal of the CACM*, 22(4) :251–256, 1979.

[9] Bart G.W. Craenen, A.E. Eiben, and Jano I. van Hemert. Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 7(5) :424–444, 2003.

[10] J. M. Crawford and L. Auton. Experimental results on the cross-over point in satisfiability problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 22–28, 1993.

[11] Andrew Davenport, Edward Tsang, Chang J. Wang, and Kangmin Zhu. Genet : A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 325–330. AAAI, 1994.

[12] Raphaël Dorne and Jin-Kao Hao. A new genetic local search algorithm for graph coloring. In *5th International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 1498 of *Lecture Notes in Computer Science*, pages 745–754. Springer, 1998.

[13] A. Eiben and J. van der Hauw. Solving 3-sat with adaptive genetic algorithms. In *Proceedings of the Fourth IEEE Conference on Evolutionary Computation*, pages 81–86. IEEE Press, 1997.

[14] Agoston E. Eiben and Jim E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[15] Tomas A. Feo and Mauricio G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letter*, 8 :67–71, 1989.

[16] Charles Fleurent and Jacques A. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26 :619–652, 1996.

[17] Frédéric Saubion Frédéric Lardeux and Jin-Kao Hao. Gasat : a genetic local search algorithm for the satisfibility problem. *Evolutionary Computation*, 14(2) :223–253, 2006.

[18] Philippe Galinier and Jin-Kao Hao. Tabu search for maximal constraint satisfaction problems. In *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1330 of *LNCS*, pages 196–208. Springer, 1997.

[19] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4) :379–397, 1999.

[20] Philippe Galinier and Jin-Kao Hao. A general approach for constraint solving by local search. *Journal of Mathematical Modelling and Algorithms*, 3(1) :73–88, 2004.

[21] Philippe Galinier, Alain Hertz, and Nicolas Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2) :267–279, 2008.

[22] I. P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for sat. In *Proceedings of AAAI-93*, 1993.

[23] Fred Glover and Manuel Laguna. Tabu search. In C.R. Reeves, editor, *Modern Heuristics Techniques for Combinatorial Problems*, pages 70–141. Blackwell Scientific Publishing, Oxford, UK, 1993.

[24] David. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[25] J. Gottlieb and N. Voss. Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. In A. et al In Eiben, editor, *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 755–764. Springer, 1998.

[26] Jens Gottlieb, Elena Marchiori, and Claudio Rossi. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10 :35–50, 2002.

[27] Jin-Kao Hao and Raphaël Dorne. Empirical studies of heuristic local search for constraint solving. In *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1118 of *LNCS*, pages 194–208. Springer, 1996.

[28] Jin-Kao Hao and Raphaël Dorne. Yet another local search method for constraint solving. In *International Symposium on Stochastic Algorithms : Foundations and Applications (SAGA)*, volume 2264 of *LNCS*, pages 342–344. Springer, 2001.

[29] Jin-Kao Hao and Raphaël Dorne. An empirical comparison of two evolutionary methods for satisfiability problems. In *Proc. of IEEE Intl. Conf. on Evolutionary Computation*, pages 450–455. IEEE Press, 1994.

[30] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search.* MIT Press, 2005.

[31] Edward A. Hirsch. Unitwalk : A new sat solver that uses local search guided by unit clause elimination. In *Proceedings of SAT 2002*, 2004.

[32] John H. Holland. *Adaptation and artificial systems.* University of Michigan Press, 1975.

[33] Holger H. Hoos and Thomas Stützle. *Stochastic local search, foundations and applications.* Morgan Kaufmann, 2005.

[34] hu Min Li and Wen Qi Huang. Diversification and determinism in local search for satisfiability. In *Proceedings of SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2005.

[35] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing : Efficient dynamic local search for sat. In *Proceedings of CP 2002, Principles and Practice of Constraints Programming*, Lecture Notes in Computer Science, pages 233–248. Springer Verlag, 2002.

[36] A. Ishtaiwi, J. Thornton, A. Sattar, and D. N. Pham. Neighbourhood clause weight redistribution in local search for sat. In *Proceedings of CP 2005*, pages 772–776, 2005.

[37] Arun Jagota and Laura A. Sanchis. Adaptive, restart, randomized greedy heuristics for maximum clique. *Journal of Heuristics*, 7(6) :565–585, 2001.

[38] K.A De Jong and W.M. Spears. Using genetic algorithms to solve np-complete problems. In *Intl. Conf. on Genetic Algorithms (ICGA'89)*, pages 124–132, Fairfax, Virginia, 1989.

[39] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139, 2002.

[40] M. Khichane, P. Albert, and C. Solnon. Integration of ACO in a constraint programming language. In *6th International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS'08)*, volume 5217 of *LNCS*, pages 84–95. Springer, 2008.

[41] M. Khichane, P. Albert, and C. Solnon. Strong integration of ant colony optimization with constraint programming optimization. In *7th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR)*, LNCS. Springer, 2010.

[42] Pedro Larranaga and Jose A. Lozano. *Estimation of Distribution Algorithms. A new tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[43] Helena R. Lourenco, Olivier Martin, and Thomas Stuetzle. *Handbook of Metaheuristics*, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, 2002.

[44] Zhipeng Lü and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 200(1) :235–244, 2010.

[45] Enrico Malaguti, Michele Monaci, and Paolo Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2) :302–316, 2008.

[46] Bertrand Mazur, Lakhdar Sais, and Eric Grégoire. Tabu search for sat. In *Proc. of the AAAI-97*, 1997.

[47] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proc. of the AAAI 97*, 1997.

[48] S. Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58 :161–205, 1992.

[49] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Comps. in Oerations Research*, 24 :1097–1100, 1997.

[50] Pablo Moscato. Memetic algorithms : a short introduction. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill Ltd., Maidenhead, UK., 1999.

[51] Ferrante Neri, Carlos Cotta, and Pablo Moscato (Eds.). Handbook of memetic algorithms. Springer, 2011.

[52] Bertrand Neveu, Gilles Trombettoni, and Fred Glover. Id walk : A candidate list strategy with a simple diversification device. In *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3258 of *LNCS*, pages 423–437. Springer Verlag, 2004.

[53] Koji Nonobe and Toshihide Ibaraki. A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, 106 :599–623, 1998.

[54] Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. BOA : The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532. Morgan Kaufmann Publishers, San Fransisco, CA, 13-17 1999.

[55] Duc Nghia Pham, John Thornton, and Abdul Sattar. Building structure into local search for sat. In *Proceedings of IJCAI 2007*, 2007.

[56] Daniel C. Porumbel, Jin-Kao Hao, and Pascale Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37(10) :1822–1832, 2010.

[57] Steve Prestwich. Random walk with continuously smoothed variable weights. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, 2005.

[58] Mauricio G.C. Resende and Celso C. Ribeiro. *Handbook of Metaheuristics*, chapter Greedy randomized adaptive search procedures, pages 219–249. Kluwer Academic Publishers, 2003.

[59] C. Rossi, E. Marchiori, and Kok. A flipping genetic algorithm for hard 3-sat problems. In *Proc. of the Genetic and Evolutionary Computation Conference*, volume 1, pages 393–400, 1999.

[60] C. Rossi, E. Marchiori, and Kok. An adaptive evolutionary algorithm for the satisfiability problem. In J. et al Carroll, editor, *Proceedings of ACM Symposium on Applied Computing*, pages 463–469. ACM, New York, 2000.

[61] D. Schuurmans, F. Southey, and R.C. Holte. The exponential subgradient algorithm for heuristic boolean programming. In *Proceedings of AAAI 2001*, pages 334–341, 2001.

[62] B. Selman and H.A. Kautz. Domain-independent extensions to gsat : Solving large structured satisfiability problems. In *Proceedings of IJCAI-93*, pages 290–295, 1993.

[63] Bart Selman and Henry Kautz. Noise strategies for improving local search. In *Proc. of the AAAI 94*, pages 337–343, 1994.

[64] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 337–343. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1994.

[65] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *10th National Conference on Artificial Intelligence (AAAI)*, pages 440–446, 1992.

[66] Y Shang and B. W. Wah. A discrete lagragian based global search method for solving satisfiability problems. *Journal of Global Optimization*, 12 :61–100, 1998.

[67] C. Solnon. Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. *European Journal of Operational Research (EJOR)*, 191 :1043–1055, 2008.

[68] C. Solnon. *Optimisation par colonies de fourmis - Collection Programmation par contraintes*. Hermès-Lavoisier, 2008.

[69] Christine Solnon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4) :347–357, 2002.

[70] William M. Spears. Simulated annealing for hard satisfiability problems. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26 :533–558, 1996.

[71] J. Thornton, D. N. Pham, S. Bain, and V. Jr. Ferreira. Additive versus multiplicative clause weighting for sat. In *Proceeding of AAAI 2004*, pages 191–196, 2004.

[72] Jano I. van Hemert and Christine Solnon. A study into ant colony optimization, evolutionary computation and constraint programming on binary constraint satisfaction problems. In *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004)*, volume 3004 of *LNCS*, pages 114–123. Springer-Verlag, 2004.

[73] Richard J. Wallace. Analysis of heuristics methods for partial constraint satisfaction problems. In *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1118 of *LNCS*, pages 308–322. Springer, 1996.

[74] Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Hydra : Automatically configuring algorithms for portfolio-based selection. In *24th AAAI Conference on Artificial Intelligence*, pages 210–216, 2010.

[75] R.A. Young and A. Reel. A hybrid genetic algorithm for a logic problem. In *Proc. of the 9th European Conf. on Artificial Intelligence*, pages 744–746, Stockholm, Sweden, 1990.

[76] Arnaud Zinflou, Caroline Gagné, and Marc Gravel. Crossover operators for the car sequencing problem. In *7th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, volume 4446 of *Lecture Notes in Computer Science*, pages 229–239, 2007.

[77] Mark Zlochin, Mauro Birattari, Nicolas Meuleau, and M. Dorigo. Model-based search for combinatorial optimization : A critical survey. *Annals of Operations Research*, 131 :373–395, 2004.