Chapitre 1

Méta-heuristiques

Jin-Kao Hao et Christine Solnon

1.1 Introduction

Une méta-heuristique est une méthode générique pour la résolution de problèmes combinatoires. 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" décrites au chapitre **??** 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. Il existe essentiellement deux grandes familles d'approches heuristiques :

- les approches perturbatives, présentées en 1.2, construisent des combinaisons en modifiant des combinaisons existantes (par croisement et mutation pour les algorithmes génétiques, par application de transformations élémentaires pour la recherche locale, par déplacement en fonction d'une vitesse pour les algorithmes par essaims de particules) ;
- les approches constructives, présentées en 1.3, génèrent des combinaisons de façon incrémentale en utilisant pour cela un modèle stochastique. Dans le cas des algorithmes gloutons aléatoires, ce modèle est statique ; dans le cas des algorithmes par estimation de distribution et des algorithmes par colonies de fourmis, il évolue en fonction des expériences passées.

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 et de diversification, 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 approches heuristiques à la résolution de problèmes classiques de l'intelligence artificielle, à savoir la satisfiabilité de formules booléennes et la satisfaction de contraintes, et nous discuterons en 1.7 de quelques perspectives de recherche.

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 $f : E \rightarrow \mathbb{R}$ 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 $d : V \times V \rightarrow \mathbb{R}$ 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.

1.2 Méta-heuristiques perturbatives

Les approches perturbativent explorent l'espace des combinaisons E en perturbant itérativement des combinaisons déja construites : partant d'une ou plu-

sieurs 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 [16]. 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 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 métaheuristique de résolution de problèmes d'optimisation combinatoire. 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.

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 croisement et mutation
    Mettre à jour la population
retourner la meilleure combinaison ayant appartenu à la population
```

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

Sélection : cette étape consiste à choisir les combinaisons de la population qui seront ensuite croisées et mutées. 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).

Croisement : cette opération consiste à générer de nouvelles combinaisons, à partir des combinaisons sélectionnées. Là encore, il existe de nombreux opérateurs de croisement. Une méthode simple consiste à choisir aléatoirement un point de croisement, à couper chaque combinaison parente en ce point, puis à reformer deux enfants en échangeant les parties composant les parents de part et d'autre du point de croisement.

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

Exemple 1 *Donner un exemple de croisement et de mutation pour le voyageur de commerce ?*

Mise à jour de la population : cette étape consiste à remplacer certaines combinaisons de la génération précédente par certaines combinaisons issues des opérations de croisement et de mutation, formant de la sorte une nouvelle génération. Là encore, il existe différentes stratégies de remplacement, favorisant plus ou moins la diversité, et plus ou moins élitistes. On peut par exemple choisir de ne garder que les meilleurs individus, qu'ils soient issus de la nouvelle génération ou de l'ancienne, ou bien ne garder que les individus de la nouvelle génération, indépendamment de leur qualité.

Critères d'arrêt : le processus d'évolution est itéré, de génération en génération, jusqu'à ce qu'une combinaison de qualité suffisante soit générée, ou bien jusqu'à ce qu'une limite de temps soit atteinte. On peut également utiliser des indicateurs de diversité (comme par exemple le taux de ré-échantillonnage ou la distance pair-à-pair) pour arrêter le processus lorsque la population est devenue trop uniforme.

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. L'algorithme 2 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Algorithme 2 : Recherche locale

Générer une combinaison initiale $e \in E$
$e^* \leftarrow e$
tant que *critères d'arrêt non atteints* **faire**
  Choisir $e' \in v(e)$
  si $f(e') > f(e^*)$ **alors** $e^* \leftarrow e'$ $e \leftarrow e'$
retourner e^*

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.

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.*

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.

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 [15], 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*) [14], qui autorise avec une très petite probabilité p_{bruit} 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*) [6], 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 [10], 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*) [9]. On peut par ailleurs faire plusieurs recherches locales en parallèle, en partant de différentes configurations initiales, et redistribuer régulièrement les configurations courantes en supprimant les moins bonnes et dupliquant les meilleures selon le principe "va avec les meilleurs" (*go with the winner*) [**?**].

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 [16], 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 et 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 et 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.

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.*

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.

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 [5]. Une autre possibilité consiste à choisir le prochain composant en fonction de probabilités dépendant de la qualité des différents composants [7].

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 $p(j) = \frac{[1/d(i,j)]^\beta}{\sum_{k \in C}[1/d(i,k)]^\beta}$. Pour choisir la prochaine ville à visiter, on calcule alors les probabilités associées à chacune des villes candidates, puis on sélectionne effectivement une de ces villes selon un principe de roulette : on associe à chaque candidat une proportion de roue correspondant à sa probabilité puis on tire un nombre aléatoire compris entre zéro et un selon une distribution uniforme. 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 les villes sont choisies selon un principe glouton favorisant les villes les plus proches.*

Les constructions gloutonnes aléatoires peuvent être itérées plusieurs fois, la meilleure combinaison étant finalement retournée à la fin.

1.3.2 Algorithmes par estimation de distributions

Les algorithmes par estimation de distribution (*Estimation of Distribution Algorithms* ; EDA) [8] 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**
  Construction d'un modèle probabiliste M en fonction de P
  Génération de nouvelles combinaisons à l'aide de M
  Mise à jour de P en fonction des nouvelles combinaisons
retourner *la meilleure combinaison ayant appartenu à la population*

Génération de la population initiale : 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 : différents types de modèles probabilistes, plus ou moins simples, peuvent être considérés. Le modèle le plus simple, appelé PBIL [2], 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 [12]. 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 contenant l'arête (i,j).*

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

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 [16]. 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
```

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. En fonction de l'application à résoudre, on peut envisager différentes structures phéromonales.

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.*

Au début de l'exécution d'un algorithme ACO, toutes les traces de phéromone sont initialisées à une valeur τ_0 donnée.

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 de composants de combinaison C pouvant être ajoutés à S, la fourmi choisit le composant $i \in C$ selon la probabilité :

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

$\tau_S(i)$ est le facteur phéromonal associé au composant de combinaison i par rapport au début de combinaison S ; la définition de ce facteur phéromonal dépend de la structure phéromonale choisie.

Exemple 7 *Par exemple, 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.*

$\eta_S(i)$ est le facteur heuristique associé au composant de combinaison i par rapport au début de combinaison S ; la définition de ce facteur dépend de l'application.

Exemple 8 *Par exemple, pour le problème du voyageur de commerce, le facteur heuristique est inversement proportionnel à la longueur de la route joignant la dernière ville ajoutée dans S à la ville candidate i.*

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 une 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.

Mise à jour de la phéromone : Une fois que chaque fourmi a construit une combinaison, éventuellement améliorée par recherche locale, 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. On peut également adopter une stratégie plus élitiste où seules les meilleures combinaisons du cycle sont récompensées. On peut intensifier encore plus la recherche en récompensant 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 9 *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

1.4.1 Algorithmes mémétiques

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, 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*) [13]. 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.

Des mécanismes peuvent être utilisés pour tirer des leçons des constructions précédentes lors de la construction d'une nouvelle combinaison. On peut par exemple utiliser une table de hachage pour mémoriser les combinaisons précédemment construites et n'appliquer l'étape de recherche locale que s'il s'agit d'une nouvelle combinaison. On peut également modifier de façon réactive la valeur du paramètre (généralement appelé α) déterminant le degré d'aléatoire dans la construction gloutonne. On peut encore faire des statistiques sur les composants utilisés dans les constructions précédentes pour biaiser les probabilités de choisir ces composants, par exemple en diminuant la probabilité de choisir les composants qui ont été les plus souvent choisis afin de diversifier la recherche.

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

1.4.3 Hybridation avec des approches complètes

1.5 Mécanismes d'intensification et de 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 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 ;
- pour l'optimisation par essaims de particules, on augmente les valeurs des coefficients d'accélération qui déterminent les influences des meilleures combinaisons de la particule et de son voisinage.

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 l'optimisation par essaims de particules, la diversification est assurée par le coefficient d'inertie ω et par les nombres aléatoires f_1 et f_2.

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 [3]. Par exemple, l'approche taboue réactive de [4] ajuste automatiquement la longueur de la liste taboue, tandis que la recherche locale *IDwalk* de [11] adapte automatiquement le nombre de voisins considérés à chaque mouvement.

1.6 Applications en intelligence artificielle

1.6.1 Satisfiabilité de formules booléennes

1.6.2 Problèmes de satisfaction de contraintes

1.7 Discussion

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] 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.

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

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

[5] 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.

[6] 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.

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

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

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.