Université Mohammed V, Faculté des Sciences de Rabat
Laboratoire de Recherche Mathématiques, Informatique et Applications

Cours des Méthodes de Résolution Exactes Heuristiques et Métaheuristiques

MASTER CODES, CRYPTOGRAPHIE ET SÉCURITÉ DE L'INFORMATION

Sidi Mohamed Douiri et Souad Elbernoussi

Table des matières

Liste des figures

1

1.1 Introduction

Si les méthodes de résolution exactes permettent d'obtenir une solutions dont l'optimalité est garantie, dans certaines situations, on peut cependant chercher des solutions de bonne qualité, sans garantie d'optimalité, mais au profit d'un temps de calcul plus réduit. Pour cela, On applique des méthodes appelées métaheuristiques, adaptées à chaque problème traité, avec cependant l'inconvénient de ne disposer en retour d'aucune information sur la qualité des solutions obtenues.
Les heuristiques ou les méta-heuristiques exploitent généralement des processus aléatoires dans l'exploration de l'espace de recherche pour faire face à l'explosion combinatoire engendré par l'utilisation des méthodes exactes. En plus de cette base stochastique, les méta-heuristiques sont le plus souvent itératives, ainsi le même processus de recherche est répété lors de la résolution. Leur principal intérêt provient justement de leur capacité à éviter les minima locaux en admettant une dégradation de la fonction objectif au cours de leur progression.
L'optimisation combinatoire (OC) occupe une place très importante en recherche opérationnelle et en informatique. De nombreuses applications pouvant être modélisées sous la forme d'un problème d'optimisation combinatoire (POC) telles que le problème du voyageur de commerce, l'ordonnancement de tâches, le problème de la coloration de graphes, etc. (POC) comprend un ensemble fini de solutions, où chaque solution doit satisfaire un ensemble de contraintes relatives à la nature du problème, et une fonction objectif pour évaluer chaque solution trouvée. La solution optimale est celle dont la valeur de l'objectif est la plus petite (resp. grande) dans le cas de minimisation (resp. maximisation) parmi l'ensemble de solutions.
Un problème d'optimisation combinatoire peut être défini par :

- Vecteur de variables $x = (x_1, x_2, ..., x_n)$,
- Domaine des variables $D = (D_1, D_2, ..., D_n)$, où les $(D_i)_{i=1,...,n}$ sont des ensembles finis,
- Ensemble de contraintes,
- Une fonction objectif f à minimiser ou à maximiser,
- Ensemble de toutes les solutions réalisable possibles est $S = \{x = (x_1, x_2, , x_n) \in D \ / \ x$ satisfait toutes les contraintes$\}$, l'ensemble S est aussi appelé un espace de recherche.

La résolution de (POC) consiste à trouver la meilleure solution, définie comme la solution globalement optimale ou un optimum global.
La résolution des problèmes combinatoires est assez délicate puisque le nombre fini de solutions réalisables croît généralement avec la taille du problème, ainsi que sa complexité. Cela a poussé les chercheurs à développer de nombreuses méthodes de résolution en recherche opérationnelle (RO) et en intelligence artificielle (IA). Ces approches de résolution peuvent être classées en deux catégories : les méthodes exactes et les méthodes approchées.
Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable et rencontrent généralement des difficultés face aux applications de taille importante. En revanche les méthodes approchées ne garantissent pas de trouver une solution exacte, mais seulement une approximation.
Ces deux classes d'algorithmes de résolution de (POC) sont décrites dans les sections suivantes.

1.2 Notions sur la complexité

Avant d'aborder les différentes méthodes de résolution des problèmes d'optimisation combinatoire nous introduisons quelques définitions et notions sur la complexité des (POC).
Généralement, le temps d'exécution est le facteur majeur qui détermine l'efficacité d'un algorithme, alors la complexité en temps d'un algorithme est le nombre d'instructions nécessaires (affectation, comparaison, opérations algébriques, lecture et écriture, etc.) que comprend cet algorithme pour une résolution d'un problème quelconque.

Définition 1.1 *Une fonction $f(n)$ est $O(g(n))$ ($f(n)$ est de complexité $g(n)$), s'il existe un réel $c > 0$ et un entier positif n_0 tel que pour tout $n \geq n_0$ on a $|f(n)| \leq c.g(n)$.*

Définition 1.2 *Un algorithme en temps polynomial est un algorithme dont le temps de la complexité est en $O(p(n))$, où p est une fonction polynomiale et n est la taille de l'instance (ou sa longueur d'entrée).*
Si k est le plus grand exposant de ce polynôme en n, le problème correspondant est dit être résoluble en $O(n^k)$ et appartient à la classe P, un exemple de problème polynomial est celui de la connexité dans un graphe.

Définition 1.3 *La classe NP contient les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. C'est la classe des problèmes qui admettent un algorithme polynomial capable de tester la validité d'une solution du problème. Intuitivement, les problèmes de cette classe sont les problèmes qui peuvent être résolus en énumérant l'ensemble de solutions possibles et en les testant à l'aide d'un algorithme polynomial.*

Définition 1.4 *On dit qu'un problème de recherche P_1 se réduit polynomialement à un problème de recherche P_2 par la réduction de Turing s'il existe un algorithme A_1 pour résoudre P_1 utilisant comme sous programme un algorithme A_2 résolvant P_2 , de telle sorte que la complexité de A_1 est polynomiale, quand on évalue chaque appel de A_2 par une constante.*

Définition 1.5 ***La classe NP-complet :*** *parmi l'ensemble des problèmes appartenant à NP, il en existe un sous ensemble qui contient les problèmes les plus difficiles : on les appelle les problèmes NP-complets. Un problème NP-complet possède la propriété que tout problème dans NP peut être transformé (réduit) en celui-ci en temps polynomial. C'est à dire qu'un problème est NP-complet quand tous les problèmes appartenant à NP lui sont réductibles. Si on trouve un algorithme polynomial pour un problème NP-complet, on trouve alors automatiquement une résolution polynomiale de tous les problèmes de la classe NP.*

Définition 1.6 ***La classe NP-difficile :*** *un problème est NP-difficile s'il est plus difficile qu'un problème NP-complet, c'est à dire s'il existe un problème NP-complet se réduisant à ce problème par une réduction de Turing.*

1.3 Les méthodes de résolution exactes

Nous présentons d'abord quelques méthodes de la classe des algorithmes complets ou exacts, ces méthodes donnent une garantie de trouver la solution optimale pour une instance de taille finie dans un temps limité et de prouver son optimalité (Puchinger et Raidl 2005).

1.3.1 La méthode séparation et évaluation (Branch and Bound)

L'algorithme de séparation et évaluation, plus connu sous son appellation anglaise Branch and Bound ($B\&B$) (Land et Doig 1960), repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations, en représentant les états solutions par un arbre d'états, avec des noeuds, et des feuilles.
Le branch-and-bound est basé sur trois axes principaux :

- L'évaluation,
- La séparation,
- La stratégie de parcours.

- L'évaluation

L'évaluation permet de réduire l'espace de recherche en éliminant quelques sous ensembles qui ne contiennent pas la solution optimale. L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence. Le branch-and-bound utilise une élimination de branches dans l'arborescence de recherche de la manière suivante : la recherche d'une solution de coût minimal, consiste à mémoriser la solution de plus bas coût rencontré pendant l'exploration, et à comparer le coût de chaque noeud parcouru à celui de la meilleure solution. Si le coût du noeud considéré est supérieur au meilleur coût, on arrête l'exploration de la branche et toutes les solutions de cette branche seront nécessairement de coût plus élevé que la meilleure solution déjà trouvée.

- La séparation

La séparation consiste à diviser le problème en sous-problèmes. Ainsi, en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer

toutes les solutions. L'ensemble de neouds de l'arbre qu'il reste encore à parcourir comme étant susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser, est appelé ensemble des neouds actifs.

- **La stratégie de parcours**

  La largeur d'abord : Cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les deux autres stratégies présentées,

  La profondeur d'abord : Cette stratégie avantage les sommets les plus éloignés de la racine (de profondeur la plus élevée) en appliquant plus de séparations au problème initial. Cette voie mène rapidement à une solution optimale en économisant la mémoire,

  Le meilleur d'abord : Cette stratégie consiste à explorer des sous-problèmes possédant la meilleure borne. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une mauvaise évaluation par rapport à la valeur optimale.

Exemple du problème du sac à dos

Le problème de sac à dos consiste à remplir un sac à dos de capacité b avec des produits x_1, x_2,..., x_n, qui ont un poids b_1, b_2,..., b_n et rapportent un cout c_1, c_2,..., c_n par unité, de façon à maximiser le profit.
On considére l'exemple suivant :

$$\begin{cases} Max \;\; z = 4x_1 + 5x_2 + 6x_3 + 2x_4 \\ 33x_1 + 49x_2 + 60x_3 + 32x_4 \leq 130 \\ x_i \in \mathbb{N} \end{cases}$$

Pour ne pas violer la contrainte du problème chaque variable peut prendre les valeurs suivantes $x_1 \in \{0,1,2,3\}$, $x_2 \in \{0,1,2\}$, $x_3 \in \{0,1,2\}$ et $x_4 \in \{0,1,2,3,4\}$, voir la figure 1.1.

1. Le neoud $x_1 = 3$ avec ($x_2 = 0$, $x_3 = 0$, $x_4 = 0$), dans ce cas, on a mis un seul article dans le sac et prendre comme évaluation du noeud $3 \times 4 = 12$: c'est une évaluation exacte qui correspond à une solution. Cela permet d'initialiser la valeur de la meilleure solution courante à 12.
2. Le noeud $x_1 = 2$, On fixe l'article qui donne le meilleur rapport *cout/poids*, cela est vérifié pour l'article 2. L'évaluation du noeud est donc calculée par : $8 + (5 \times 64/49) = 14,53$. Puisque $14,53 > 12$, on divise ce noeud.
3. Le noeud $x_1 = 2$, $x_2 = 1$, pour le noeud $x_1 = 3$, on obtient une évaluation exacte égale à 13. La solution correspondante devient la nouvelle meilleure solution courante et la meilleure valeur est égale à 13.

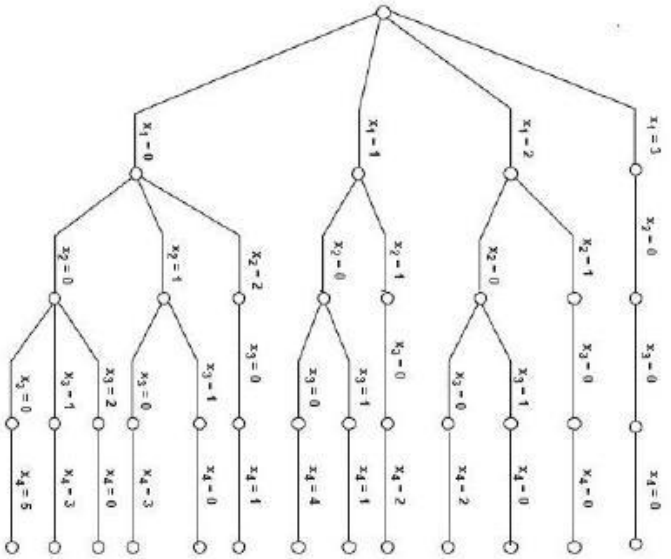

FIGURE 1.1 – *Solutions possibles de l'exemple de sac à dos.*

4. Le noeud $x_1 = 2$, $x_2 = 0$ a comme évaluation $8 + (6 \times 64/60) = 14,4$ ($x_3 = 64/60$, $x_4 = 0$). Puisque $14,4 > 13$, on divise ce noeud.
5. Le noeud $x_1 = 2$, $x_2 = 0$, $x_3 = 1$, a comme évaluation est exacte égale à 14. La solution correspondante devient la nouvelle meilleure solution courante et la meilleure valeur est égale à 14.
6. Le noeud $x_1 = 1$, a comme évaluation égale à $4 + (5 \times 97/49) = 13,89$. On passe au dernier noeud $x_1 = 0$, a comme évaluation égale à $5 \times 130/49 = 13,26$ ($x_2 = 130/49$).

La valeur de la solution optimale égale à 14 et elle consiste à prendre 2 unités du produit 1 et une du produit 3 ($x_1 = 2$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$), voir la figure 1.2.

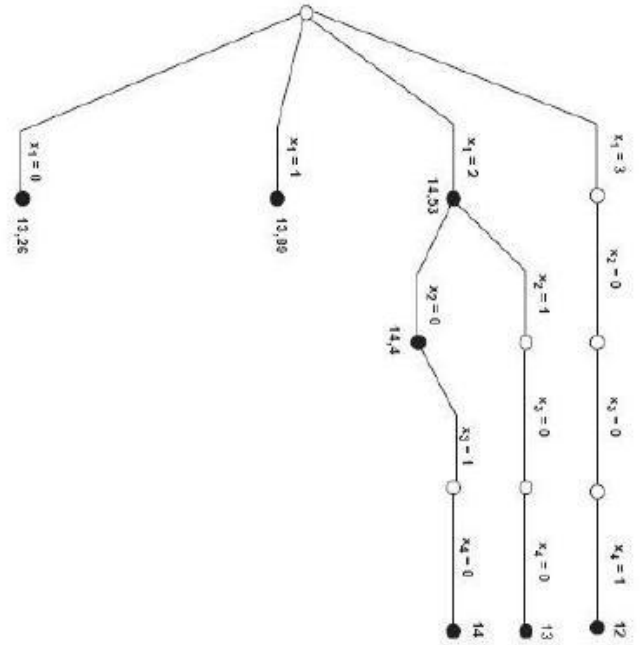

FIGURE 1.2 – *Résolution de l'exemple de sac à dos par Branch & Bound.*

1.3.2 La méthode de coupes planes (Cutting-Plane)

La méthode de coupes planes a été développée par (Schrijver 1986), elle est destinée à résoudre des problèmes d'optimisation combinatoire (POC) qui se formulent sous la forme d'un programme linéaire (PL) :

$$min\{c^T x : Ax \geq b, x \in R^n\} \tag{1.1}$$

Dans le cas, où (POC) est de grande taille pour le représenter explicitement en mémoire ou pour qu'il tient dans un solveur de programmation linéaire, on utilise une technique qui consiste à enlever une partie de ces contraintes et de résoudre le problème relaxé (POCR). La solution optimale de (PL) est contenue dans l'ensemble de solutions réalisables de cette relaxation. Pour un problème de minimisation la solution optimale du problème (POCR) est inférieure ou égale à la solution optimale donnée par (POC).
Cette méthode consiste à résoudre un problème relaxé, et à ajouter itérativement des contraintes du problème initial. On définit une contrainte pour le problème de minimisation (1.1) par le couple (s, s_0) où $s \in R^n$ et $s_0 \in R$, cette contrainte est dite violée par la solution courante $\overline{x}$ si pour tout $y \in \{x : Ax \geq b\}$ on a $s^T\overline{x} < s_0$ et $s^T y \geq s_0$, on appelle alors ces contraintes des coupes planes. On arrête l'algorithme lorsqu'il n'y a plus de contraintes violées par la solution courante, on obtient ainsi une solution optimale pour le problème initial.
La méthode des coupes planes est peu performante mais sa performance est améliorée lorsqu'elle est combinée avec la méthode "Branch and Bound".

1.3.3 La méthode (Branch and Cut)

La méthode des coupes planes n'est pas toujours efficace face aux problèmes difficiles. De même, bien que l'algorithme du "Branch and Bound" puisse être très performant pour une certaine classe de problèmes, pour cela on utilise la méthode "Branch and Cut" qui combine entre l'algorithme du "Branch and Bound" et de la méthode des coupes planes. Pour une résolution d'un programme linéaire en nombres entiers, la méthode "Branch and Cut" commence d'abord par relaxer le problème puis appliquer la méthode des coupes planes sur la solution trouvée. Si on n'obtient pas une solution entière alors le problème sera divisé en plusieurs sous-problèmes qui seront résolus de la même manière.
On veut résoudre le problème d'optimisation $(minc^t x : Ax \geq b; x \in \mathbb{R}^n)$ avec $A \in \mathbb{R}^{mxn}$ et $b \in \mathbb{R}^m$.

Algorithme 1: Branch and Cut

Liste des problèmes $= \emptyset$;
Initialiser le programme linéaire par le sous problème de contraintes (A_1, b_1) avec $A_1 \in \mathbb{R}^{m_1 xn}$et $b_1 \in \mathbb{R}^{m_1})$ avec $m_1 << m$;
Etapes d'évaluation d'un sous problème
Calculer la solution optimale $\bar{x}$ du programme linéaire $c^t\bar{x} = min(c^t x : A_1 x \geq b_1, x \in \mathbb{R}^n)$;
Solution courante= Appliquer la méthode des coupes polyédrales();
Fin étapes d'évaluation
Si Solution courante est réalisable **alors**
$x^* = \bar{x}$ est la solution optimale de $min(c^t x : Ax \geq b, x \in \mathbb{R}^n)$;
Sinon
Ajouter le problème dans Liste des sous problèmes;
Fin Si
Tant que Liste des sous problèmes $\neq \emptyset$ **faire**
Sélectionner un sous problème;
Brancher le problème;
Appliquer les étapes d'évaluation;
Fin Tant que

1.3.4 La méthode de la génération de colonnes

Le principe de la génération de colonnes repose sur le fait que toutes les variables d'un programme linéaire ne seront pas forcément utilisées pour atteindre la solution optimale. L'objectif de cette méthode est de résoudre un problème réduit avec un ensemble limité de variables. Le problème initial est appelé problème maître, et le problème réduit est appelé problème restreint. Le problème restreint est plus simple à résoudre, mais si l'ensemble de ses variables ne contient pas celles qui donne la solution optimale pour le problème maître, pour atteindre la solution optimale du problème maître, il faut rajouter au problème restreint des variables pouvant être utilisées pour améliorer la solution.

Le problème consistant à chercher la meilleure variable à rajouter dans le problème restreint est appelé sous-problème associé au problème maître

(ou oracle). Il a comme objectif de trouver la variable (ou colonne) de coût réduit minimum (c-à-d. la plus prometteuse pour améliorer la solution). Le coût réduit des variables est calculé à l'aide des variables duales obtenues après la résolution du problème restreint. Le point du dual ainsi utilisé dans le sous problème est appelé point de séparation. Souvent, il s'agit d'une solution optimale du dual du problème restreint.
On considère le programme linéaire continu (LP) suivant :

$$(LP)\begin{cases} Min \sum_{i \in T} c_i x_i \\ \sum_{i \in T} a_{ij} x_i \geq, b_j \quad j = 1, ..., n \\ x_i \geq 0, \forall i \in T \end{cases}$$

Nous supposons que le nombre de variables de T est trop grand pour que le problème (LP) puisse être résolu en temps raisonnable, et que nous voulions le résoudre par génération de colonnes. Nous cherchons donc à résoudre le problème restreint associé au problème maître avec un ensemble restreint de variables noté R_l. Il faut que le problème restreint soit réalisable. Il est possible d'utiliser des colonnes simples par exemple des colonnes aléatoires, ou encore celles issues d'une solution faisable obtenue à partir d'une heuristique.
Le problème restreint (RLP) est donné sous la forme suivante :

$$(RLP)\begin{cases} Min \sum_{i \in R_l} c_i x_i \\ \sum_{i \in R_l} a_{ij} x_j \geq, b_j \quad j = 1, ..., n \\ x_i \geq 0, \forall i \in R_l \end{cases}$$

Le problème (RLP) est maintenant de taille réduite et sera plus facile à résoudre par un solveur. Cette résolution nous fournira les valeurs optimales des variables duales v_j associées aux contraintes. Ces valeurs sont passées au sous problème qui nous permet d'obtenir la ou les colonnes à rajouter dans l'ensemble R_l.
Le calcul du coût réduit nous permet de savoir si une colonne a fait diminuer la valeur de l'objectif (et donc de l'améliorer). Prenons par exemple la colonne xi du problème maître (LP), son coût réduit vaut :

$$\bar{c_i} = c_i - \sum_{j=1}^{n} a_{ij} v_j$$

Puisque (LP) est un problème de minimisation, le sous problème cherche aussi à minimiser ce coût réduit. Si le coût réduit minimum est positif, alors aucune colonne ne peut être ajoutée au problème restreint (RLP) pour améliorer l'objectif. La solution optimale du problème restreint est donc une solution optimale du problème maître (LP). Sinon, on rajoute une ou des colonnes parmi celles ayant un coût réduit négatif en faisant une mise à jour de l'ensemble R_l et on résout après le nouveau problème restreint (RLP).

Remarque

La méthode de la génération de colonnes peut être combinée avec un processus de *Branch&Bound* pour résoudre un programme linéaire en nombres entiers, cette méthode est appelée *Branch&Price*.

1.4 Heuristiques

En optimisation combinatoire, une heuristique est un algorithme approché qui permet d'identifier en temps polynomial au moins une solution réalisable rapide, pas obligatoirement optimale. L'usage d'une heuristique est efficace pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte. Généralement une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre sans offrir aucune garantit quant à la qualité de la solution calculée. Les heuristiques peuvent être classées en deux catégories :

- Méthodes constructives qui génèrent des solutions à partir d'une solution initiale en essayant d'en ajouter petit à petit des éléments jusqu'à ce qu'une solution complète soit obtenue,
- Méthodes de fouilles locales qui démarrent avec une solution initialement complète (probablement moins intéressante), et de manière répétitive essaie d'améliorer cette solution en explorant son voisinage.

1.5 Métaheuristiques

Face aux difficultés rencontrées par les heuristiques pour avoir une solution réalisable de bonne qualité pour des problèmes d'optimisation difficiles, les métaheuristiques ont fait leur apparition. Ces algorithmes sont plus complets et complexes qu'une simple heuristique, et permettent généralement d'obtenir une solution de très bonne qualité pour des problèmes issus des domaines de la recherche opérationnelle ou de l'ingénierie dont on ne connait pas de méthodes efficaces pour les traiter ou bien quand la résolution du problème nécessite un temps élevé ou une grande mémoire de stockage.
Le rapport entre le temps d'exécution et la qualité de la solution trouvée d'une métaheuristique reste alors dans la majorité des cas très intéressant par rapport aux différents types d'approches de résolution.
La plupart des métaheuristiques utilisent des processus aléatoires et itératifs comme moyens de rassembler de l'information, d'explorer l'espace de recherche et de faire face à des problèmes comme l'explosion combinatoire. Une métaheuristique peut être adaptée pour différents types de problèmes, tandis qu'une heuristique est utilisée à un problème donné. Plusieurs d'entre elles sont souvent inspirées par des systèmes naturels dans de nombreux domaines tels que : la biologie (algorithmes évolutionnaires et génétiques) la physique (recuit simulé), et aussi l'éthologie (algorithmes de colonies de fourmis).
Un des enjeux de la conception des métaheuristiques est donc de faciliter le choix d'une méthode et le réglage des paramètres pour les adapter à un problème donné.

1.5.1 Les algorithmes génétiques

Les algorithmes génétiques (AG) sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique. Ils ont été adaptés à l'optimisation par John Holland (Holland 1975), également les travaux de David Goldberg ont largement contribué

à les enrichir (Goldberg 1989a), (Goldberg 1989b).
Le vocabulaire utilisé est le même que celui de la théorie de l'évolution et de la génétique, on emploie le terme individu (solution potentielle), population (ensemble de solutions), génotype (une représentation de la solution), gène (une partie du génotype), parent, enfant, reproduction, croisement, mutation, génération, etc.
Leur fonctionnement est extrêmement simple, on part d'une population de solutions potentielles (chromosomes) initiales, arbitrairement choisies. On évalue leur performance (Fitness) relative. Sur la base de ces performances on crée une nouvelle population de solutions potentielles en utilisant des opérateurs évolutionnaires simples : la sélection, le croisement et la mutation. Quelques individus se reproduisent, d'autres disparaissent et seuls les individus les mieux adaptés sont supposés survivre. On recommence ce cycle jusqu'à ce qu'on trouve une solution satisfaisante. En effet, l'héritage génétique à travers les générations permet à la population d'être adaptée et donc répondre au critère d'optimisation, la figure 1.3 illustre les principales étapes d'un algorithme génétique. Un algorithme génétique recherche le ou les extrema d'une fonction définie sur un espace de données. Son mise en oeuvre nécessite :

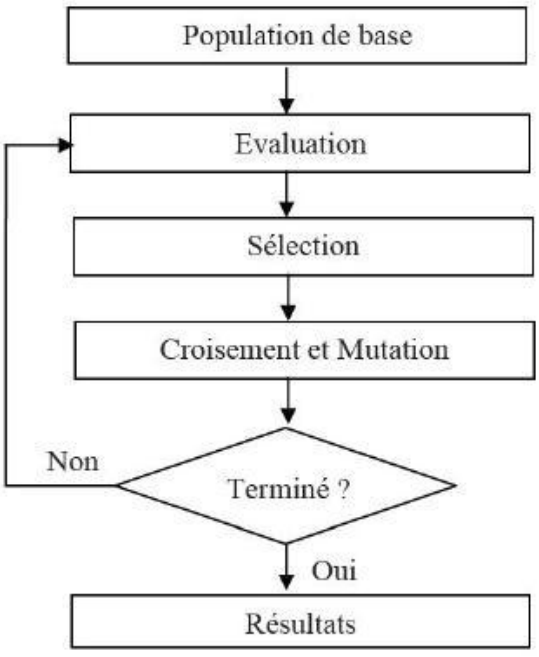

FIGURE 1.3 – *Fonctionnement d'un algorithme génétique.*

- **Le codage des données**

  La première étape est de définir et coder convenablement le problème. Cette étape associe à chaque point de l'espace de recherche une structure de données spécifique, appelée génotype ou ensemble de chromosomes, qui caractérisera chaque individu de la population.

Le codage de chaque individu en séquence est essentielle dans l'élaboration d'un algorithme génétique dont dépend notamment l'implémentation des opérateurs de transformations. Ainsi, cette phase détermine la structure de données qui sera utilisée pour coder le génotype des individus de la population. Le codage doit donc être adapté au problème traité.
Plusieurs types de codages sont utilisés dans la littérature, les premiers résultats théoriques sur les algorithmes génétiques ont opté pour un codage par une séquence binaire de longueur fixe à travers la notion de schéma (Goldberg 1989a). L'efficacité de l'algorithme génétique dépend donc du choix convenable du type de codage.

- **Génération de la population initiale**
  La génération de la population initiale, c'est-à-dire le choix des dispositifs de départ que nous allons faire évoluer, ce choix de la population initiale d'individus conditionne fortement la rapidité de l'algorithme. Néanmoins, une initialisation aléatoire est plus simple à réaliser : les valeurs des gènes sont tirées au hasard selon une distribution uniforme. Toutefois, il peut être utile de guider la génération initiale vers des sous domaines intéressants de l'espace de recherche. Par exemple lors d'une recherche d'optima dans un problème d'optimisation sous contraintes, il est préférable de produire des éléments satisfaisant les contraintes. La population initiale doit être suffisamment diversifiée et de taille assez importante pour que la recherche puisse parcourir l'espace d'état dans un temps limité.

- **Fonction d'adaptation (Fitness)**
  L'évaluation de la Fitness est généralement l'étape dans laquelle on mesure la performance de chaque individu. Pour pouvoir juger la qualité d'un individu et ainsi le comparer aux autres, il faut établir une mesure commune d'évaluation. Aucune règle n'existe pour définir cette fonction, son calcul peut ainsi être quelconque, que ce soit une simple équation ou une fonction affine. La manière la plus simple est de poser la fonction d'adaptation comme la formalisation du critère d'optimisation.

- **Sélection**
  La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais, pendant le passage d'une génération à une autre, ce processus est basé sur la performance de l'individu. L'opérateur de sélection doit être conçu pour donner également une chance aux mauvais éléments, car ces éléments peuvent, par croisement ou mutation, engendrer une descendance pertinente par rapport au critère d'optimisation. Il existe différentes techniques de sélection, on propose quelques unes.

  Sélection uniforme : On ne s'interesse pas à la valeur d'adaptation fitness et la sélection s'effectue d'une manière aléatoire et uniforme telle que chaque individu i a la même probabilité $Prob(i) = 1/T_{pop}$ comme tous les autres individus (T_{pop} est la taille de la population).

Sélection par tournoi : Deux individus sont choisis au hasard, on compare leurs fonctions d'adaptation et le mieux adapté est sélectionné.

Élitisme : Cette méthode de sélection permet de favoriser les meilleurs individus de la population. Ce sont donc les individus les plus prometteurs qui vont participer à l'amélioration de notre population. On peut constater que cette méthode induisait une convergence prématurée de l'algorithme.

Sélection par roulette : La sélection des individus par la méthode de la roulette s'inspire de la roue de loterie sur laquelle chaque individu est représenté par un secteur proportionnel à sa fitness. On fait tourner la roue et on sélectionne un individu. Les individus les mieux évalués ont statistiquement plus de chance d'être sélectionnés, mais donne aussi une possibilité aux individus mal adaptés d'être choisis. À chaque individu i une probabilité est associée, d'être choisi proportionnelle à son adaptation f_i :
$Prob(i) = f_i / \sum f_j$, où $\sum f_j$ désigne la somme des adaptations de la population.

Reste stochastique sans remplacement : Cette sélection associe la sélection par roulette et la sélection déterministe. Un nombre minimal de représentants de chaque individu parmi les futurs parents est déterminé par avance en fonction de son adaptation, puis la population sera complétée par un tirages aléatoires. Pour un individu i, le nombre de représentation dans la future génération est donné par $E(f_i)$, où E désigne la partie entière et f_i désigne l'adaptation de i rapportée à la moyenne des adaptations de tous les individus. La génération suivante est alors complétée par la méthode de sélection par roulette telle que l'évaluation d'un individu est donnée par le reste stochastique $r_i = f_i - E(f_i)$.

- **Croisement**

L'opérateur de croisement favorise l'exploration de l'espace de recherche et enrichit la diversité de la population en manipulant la structure des chromosomes, le croisement fait avec deux parents et génère deux enfants, en espérant qu'un des deux enfants au moins héritera de bons gènes des deux parents et sera mieux adapté qu'eux. Il existe plusieurs méthodes de croisement par exemple le croisement en un point, ou en multiples points voir les figures 1.4 et 1.5.

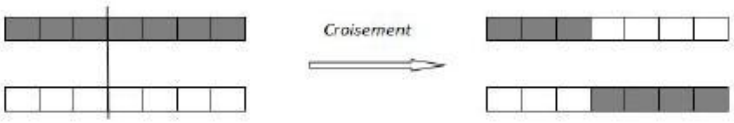

FIGURE 1.4 – *Croisement en un point.*

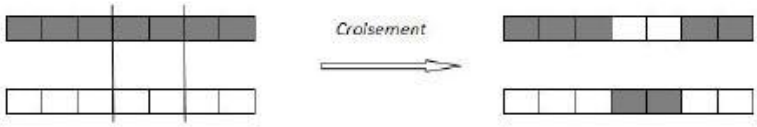

FIGURE 1.5 – *Croisement en deux points.*

Croisement uniforme : Il constitue la généralisation du principe d'échange introduit par le croisement en un point. Il procède en effet à l'échange de chaque élément selon une probabilité fixée. Le jeu de paramètres se réduit donc à la donnée de cette probabilité.

Croisement MPX : Le croisement MPX (Maximal Preservative X), a été proposé par (Mülhenbein 1993) pour résoudre le problème du voyageur de commerce. L'idée de cet opérateur est d'insérer une partie du chromosome d'un parent dans le chromosome de l'autre parent de telle façon que le chromosome résultant soit le plus proche possible de ses parents tout en évitant les doublons.

Exemple

Nous considérons par exemple les deux parents :

$$P_1 = (34|165|27) \;\; et \;\; P_2 = (12|356|47)$$

Après le croisement les deux parents vont produire deux enfants en échangeant tout d'abord les deux parties à l'interieure de chaque parent comme suit :

$$E_1 = (xx|356|xx) \;\; et \;\; E_2 = (xx|165|xx)$$

L'échange définit ainsi une série de permutations : $1 \leftrightarrow 3$, $6 \leftrightarrow 5$, $5 \leftrightarrow 6$. Certaines valeurs de génes sont donc inchangées (car elles sont différentes aux valeurs du bolc intérieur) :

$$E_1 = (x4|356|27) \;\; et \;\; E_2 = (x2|165|47)$$

Finalement, le premier x de l'enfant E_1 est remplacé par 1 en raison de la permutation $1 \leftrightarrow 3$. De la même manière, on réalise les permutations pour les autres x. Les individus enfants finaux sont donc de la forme :

$$E_1 = (14|356|27) \;\; et \;\; E_2 = (32|165|47)$$

- Mutation

L'opérateur de mutation est un processus où un changement mineur du code génétique appliqué à un individu pour introduire de la diversité et ainsi d'éviter de tomber dans des optimums locaux. Cet opérateur est appliqué avec une probabilité P_m généralement inférieure à celle du croisement P_c, voir la figure 1.6.

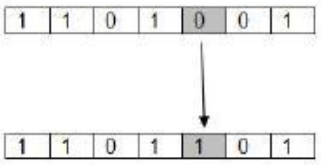

FIGURE 1.6 – *Représentation schématique d'une mutation dans le cas d'un codage binaire.*

L'efficacité des algorithmes génétiques dépend fortement du réglage des différents paramètres caractérisant ces algorithmes, et qui sont parfois difficiles à déterminer. Des paramètres comme la taille de la population, le nombre maximal des générations, la probabilité de mutation p_m, et la probabilité de croisement p_c.
Les deux premiers paramètres dépendent directement de la nature du problème et de sa complexité, et leurs choix doit représenter un compromis entre la qualité des solutions et le temps d'exécution.
La probabilité de croisement p_c est liée à la forme de la fonction dévaluation. Son choix est en général heuristique. Plus sa valeur est élevée, plus la population subit des changements importants.
La probabilité de mutation p_m est généralement faible puisqu'un taux élevé risque de conduire vers un optimum local. En revanche, une probabilité faible permet d'assurer une bonne exploration de l'espace de recherche sans perturber la convergence.
Le succès des algorithmes génétiques dépend aussi de la manière du codage des individus. Dans les problèmes combinatoires, ce codage est souvent suggéré par la nature même du problème, ce qui peut induire de meilleurs résultats.

Exemple de la mise en oeuvre des algorithmes génétiques

Cet exemple est dû à Goldberg (1989). Il consiste à trouver le maximum de la fonction f(x)=x sur l'intervalle [0,31], où x est un entier naturel. On a 32 valeurs possibles pour x, on choisit donc un codage discret sur 5 bits : on obtient par exemple la séquence 0,1,1,0,1 pour 13, la séquence 1,1,0,0,1 pour 25, etc.

Initialisation de la population se fait d'une manière aléatoire et en fixant sa taille à 4 individus. On définit simplement la fitness comme étant la valeur de x, vu qu'on en cherche la valeur maximum sur l'intervalle [0,31].
Soit la population initiale suivante :
Nous utilisons après une sélection par roulette :
Nous faisons tourner la roue 4 fois de suite, nous obtenons une nouvelle population :

Individu	Séquence	Adaptation	% du total
1	01011	11	18.6
2	10011	19	32.2
3	00101	5	8.5
4	11000	24	40.7
Total		59	100

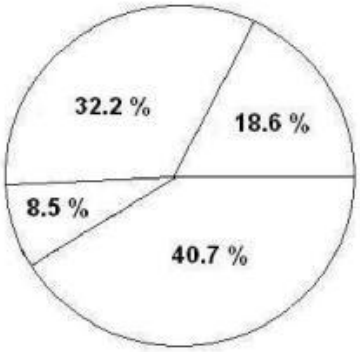

FIGURE 1.7 – *Application de la roulette sur la population.*

Individu	Séquence
1	11000
2	00101
3	11000
4	01011

Nous appliquons l'opérateur de croisement en un seul point. Les point de crossover sont choisis aléatoirement :

- Couple 1 : l'individu 2 avec l'individu 3
- Couple 2 : l'individu 1 avec l'individu 4

Nous obtenons alors les anfants suivants :
Nous appliquons ensuite l'opérateur de mutation qui choisit de manière

Parents	Enfants
001 | 01	00100
110 | 00	11001
110 | 00	01000
010 | 11	11011

aléatoire l'individu et le géne où on doit faire une mutation :

Parents	Enfants
00100	00100
11001	11101
01000	01100
11011	11011

Nous remplaçons entièrement l'ancienne population P par une nouvelle P' de même taille : En une seule génération le maximum est passé

P'	Séquence	Adaptation
1	00100	4
2	11101	29
3	01100	12
4	11011	27
Total		72

de 24 à 29, et la fitness globale de la population a relativement augmentée pour passer de 59 à 72. Nous continuons à engendrer des générations successives jusqu'à obtenir le maximum global qui vaut 31.

1.5.2 Les algorithmes de colonies de fourmis

Les algorithmes de colonies de fourmis ont été proposés par Colorni, Dorigo et Maniezzo en 1992 (Colorni et al. 1992b) et appliquées la première fois au problème du voyageur de commerce. Ce sont des algorithmes itératifs à population où tous les individus partagent un savoir commun qui leur permet d'orienter leurs futurs choix et d'indiquer aux autres individus des choix à suivre ou à éviter. Le principe de cette métaheuristique repose sur le comportement particulier des fourmis, elles utilisent pour communiquer une substance chimique volatile particulière appelée phéromone grâce à une glande située dans leur abdomen. En quittant leur nid pour explorer leur environnement à la recherche de la nourriture, les fourmis arrivent à élaborer des chemins qui s'avèrent fréquemment être les plus courts pour aller du nid vers une source de nourriture. Chaque fourmi dépose alors une quantité de phéromones sur ces pistes qui deviendront un moyen de communication avec leurs congénères, les fourmis choisissent ainsi avec une probabilité élevée les chemins contenant les plus fortes concentrations de phéromones à l'aide des récepteurs situés dans leurs antennes.
La figure 1.8 illustre et confirme ce constat, une expérience a été faite par gauss et deneubourg en 1989 appelée expérience du pont à double branche, les fourmis qui retournent au nid rapidement, après avoir visité la source de nourriture, sont celles qui ont choisi la branche courte

et les fourmis empruntant cette branche faisant plus d'aller retour, et par conséquent la quantité de phéromones déposée sur la plus courte branche est relativement supérieure que celle présente sur la plus longue branche. Puisque les fourmis sont attirées plus vers les pistes de plus grande concentration en phéromones, alors la branche courte sera la plus empruntée par la majorité des fourmis. Cette métaheuristique a permis

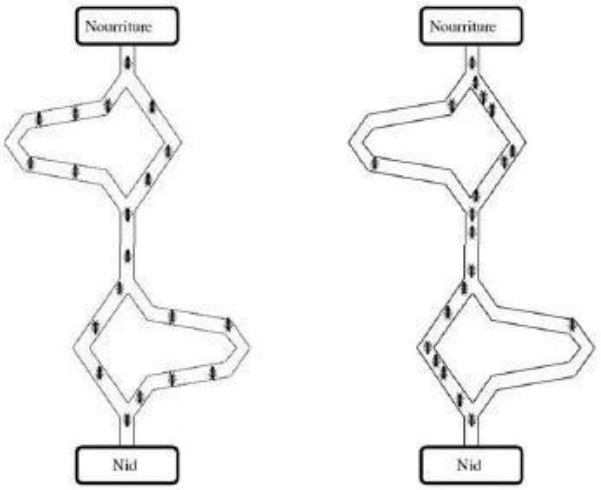

FIGURE 1.8 – *L'expérience du pont à double branche.*

de résoudre différents problèmes d'optimisation combinatoire à forte complexité, comme le problème du voyageur de commerce (Dorigo et al. 1996), le problème de coloration de graphe (Costa et Hertz 1997), le problème d'affectation quadratique (Gambardella et al. 1999) ou le problème de routage de véhicules (Mazzeo et Loiseau 2004), etc.

Principe de l'algorithme

Dans ce paragraphe on décrit l'implémentation d'un algorithme de colonie de fourmis original dit 'Ant System' (AS) (Colorni et al. 1992a), orienté pour résoudre le problème de voyageur de commerce (TSP), ce problème consiste à trouver le plus court cycle hamiltonien dans un graphe, où chaque sommet du graphe représente une ville. La distance entre les villes i et j est représentée par d_{ij}, et le couple (i, j) représente l'arête entre ces deux villes. Nous initialisons d'abord la quantité de phéromone sur les arêtes à $\tau_{init} > 0$, chaque fourmi parcourt le graphe est construit un trajet complet (une solution). A chaque étape de la construction des solutions, la fourmi doit décider à quel sommet elle va se déplacer, cette décision est prise d'une manière probabiliste fondée sur les valeurs de phéromone et d'une information statistique qui permet notamment de trouver une bonne solution.
La probabilité pour qu'une fourmi k se déplace du sommet i au sommet j, qui appartient à un ensemble de sommets qui ne sont pas encore visités

par la fourmi k noté S_i^k, est :

$$p_{ij}^k(t) = \frac{(\tau_{ij}(t))^\alpha.(\eta_{ij})^\beta}{\sum_{l \in S_i^k} (\tau_i(l))^\alpha.(\eta_{il})^\beta} \tag{1.2}$$

α et β sont deux paramètres qui influencent sur l'importance de l'intensité de phéromone τ_{ij}, et l'information statistique dite visibilité η_{ij}. Cette valeur guide le choix des fourmis vers des villes proches, et éviter le plus possible celles trop lointaines ($\eta_{ij} = \frac{1}{d_{ij}}$). Pour $\alpha = 0$, on prend en compte juste la visibilité c'est-à-dire que le choix sera tombé à chaque fois sur la ville la plus proche. Si $\beta = 0$, seules les pistes de phéromones jouent sur le choix. Pour éviter une sélection trop rapide d'un chemin, un compromis convenable entre ces deux paramètres est obligatoire.

Mise à jour de phéromones

Lorsque toutes les fourmis ont construit une solution, une quantité de phéromones $\Delta\tau_{ij}^k$ est déposée par chaque fourmi k sur son trajet.
Pour toute itération t, si le chemin (i, j) est dans la tournée de la fourmi k la quantité de phéromones déposée sur ce trajet est :

$$\Delta\tau_{ij}^k(t) = \frac{Q}{L^k(t)}$$

Où $L^k(t)$ est la longueur totale de la tournée de la fourmi k, et Q est une constant.
Donc l'ajout de la quantité de phéromones dépend certainement de la qualité de la solution obtenue c'est-à-dire plus la tournée parcourue est petite plus l'ajout de la quantité de phéromones est important.
Dans le but de ne pas négliger toutes les mauvaises solutions obtenues, et ainsi éviter la convergence vers des optima locaux de mauvaise qualité, le concept d'évaporation des pistes de phéromones est simulé à travers un paramètre ρ appelé le taux d'évaporation ($0 < \rho < 1$) comme suit :

$$\tau_{ij}(t+1) \leftarrow (1-\rho).\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{1.3}$$

Où $\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^k(t)$, t représente une itération donnée et m le nombre de fourmis.

Différentes variantes

- Ant System et élitisme

Cette variante a été introduite en 1996 par Dorigo et al. dans (Dorigo et al. 1996) dont seules les fourmis qui ont effectué le plus court chemin déposent une grande quantité de phéromones, ce qui permet d'augmenter la quantité de phéromones sur les meilleures solutions et favoriser ainsi l'exploration des solutions prometteuses par toutes les autres fourmis.

- **Ant-Q System**
  Pour cette variante proposée par (Dorigo et Gambardella 1997), (Dorigo et al. 1999), la règle de mise à jour locale est inspirée du "Q-learning". Aucune amélioration n'a été démontré pour cette variante par rapport à l'algorithme AS.

- **Max-Min Ant System (MMAS)**
  Cette variante basée sur l'algorithme (AS), a été proposée par Stützle et Hoos (Stützle et Hoos 1997), (Stützle et Hoos Holger 2000), et présente quelques différences :

  1. Déterminer des valeurs des pistes bornées par τ_{min} et τ_{max},
  2. Initialiser les valeurs de phéromones sur les pistes par la valeur maximale τ_{max},
  3. Seule la meilleure fourmi met à jour une piste de phéromone,
  4. La mise à jour d'une piste de phéromones est faite d'une manière inversement proportionnelle c'est-à-dire que les pistes les plus fortes sont moins renforcées que les plus faibles,
  5. Possibilité de réinitialiser les pistes des phéromones.

Ant Colony System ACS

L'Ant Colony System ACS (Système de Colonies de Fourmis) est une généralisation de la méthode AS, a été proposée par Dorigo en 1997, et fondée sur plusieurs modifications de AS au niveau du règle de transition et la gestion de pistes de phéromones. Le but est d'améliorer les performances de l'AS pour résoudre les problèmes de grandes tailles. Les principaux changements sont les suivants :

– Pour assurer l'équilibre entre la diversification et l'intensification, la méthode ACS introduit un paramètre q_0 dans la règle de transition pour qu'elle aura la forme suivante : Le choix d'une ville j après une ville i dans le chemin par chaque fourmi k est donné par la règle :

$$(*)\ \ j = \begin{cases} argmax_{z \in S_i^k}[(\tau_{iz})^{\alpha}(\eta_{iz})^{\beta}] & si\ \ q \leq q_0 \\ u\ \ si\ \ q > q_0 \end{cases}$$

où q est une variable aléatoire uniformément distribuée sur [0,1], et u est une ville dans S_i^k (l'ensemble de villes qui ne sont pas encore visités par la fourmi k), choisie si $q > q_0$ par l'algorithme AS. Ainsi le système tend à effectuer une diversification. Dans l'autre cas, c'est-à-dire si $q \leq q_0$, le système tend à effectuer une intensification, en exploitant plus l'information recoltée par le système.

– Cette modification est basée essentiellement sur la mise à jour locale de pistes de phéromones. Ainsi chaque fourmi effectue une mise à jour locale à chaque passage d'une ville à une autre, en déposant une quantité de phéromones donnée par la formule suivante :

$$\tau_{ij}(t+1) \leftarrow (1-\rho)\tau_{ij}(t) + \rho\tau_{init}$$

où τ_{init} est la valeur initiale de la piste. À chaque passage, les arêtes visitées voient leur quantité de phéromone diminuer, ce qui favorise la

diversification par la prise en compte des trajets non explorés. À chaque itération, la mise à jour globale s'effectue comme suit :

$$\tau_{ij}(t+1) \leftarrow (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t)$$

où les arêtes (i,j) appartiennent au meilleur tour de longueur L et où $\Delta\tau_{ij}(t) = 1/L$. Ici, seule la meilleure piste est donc mise à jour, ce qui participe à une intensification par sélection de la meilleure solution.

– La troisième modification apportée par la méthode ACS sur le système est l'utilisation d'une liste des candidats. Dans cette liste, nous stockons un certain nombre de villes les plus proches (en terme de distance) pour chaque ville. Donc chaque fourmi choisit la ville la plus proche à la ville déjà visitée s'il y a encore des villes non encore visitée dans cette liste. Dans l'autre cas la fourmi choisit une ville par la règle de tansition spéciale à l'ACS, donnée par (*). Donc une fourmi ne choisira pas une ville en dehors de liste de candidats que dans le cas quand celle ci est déjà explorée (visitée).

L'objectif principal de ces modifications sur la méthode AS est d'assurer la balance (l'équilibre) entre l'intensification et la diversification.

1.5.3 Recuit simulé (simulated annealing)

Le recuit simulé (SA) a été introduit par (Kirkpatrik et al. 1983) et (Cerný 1985) comme une méthode de recherche locale normale, utilisant une stratégie pour éviter les minima locaux. Cette métaheuristique est basée sur une technique utilisée depuis longtemps par les métallurgistes qui, pour obtenir un alliage sans défaut, faisant alterner les cycles de réchauffage (ou de recuit) et de refroidissement lent des métaux. Le recuit simulé s'appuie sur des travaux faites par (Metropolis et al. 1953), qui ont pu décrire l'évolution d'un système en thermodynamique.

Le principe du recuit simulé est de parcourir de manière itérative l'espace des solutions. On part avec une solution notée s_0 initialement générée de manière aléatoire dont correspond une énergie initiale E_0, et une température initiale T_0 généralement élevée. A chaque itération de l'algorithme, un changement élémentaire est effectué sur la solution, cette modification fait varier l'énergie du système ΔE. Si cette variation est négative (la nouvelle solution améliore la fonction objective, et permet de diminuer l'énergie du système), elle est acceptée. Si la solution trouvée est moins bonne que la précédente alors elle sera acceptée avec une probabilité P calculée suivant la distribution de Boltzmann suivante :

$$P(E,T) = \exp^{-\frac{\Delta E}{T}} \tag{1.4}$$

En fonction du critère de Metropolis, un nombre $\epsilon \in [0,1]$ est comparé à la probabilité $p = \exp^{-\frac{\Delta E}{T}}$. Si $p \leq \epsilon$ la nouvelle solution est acceptée.

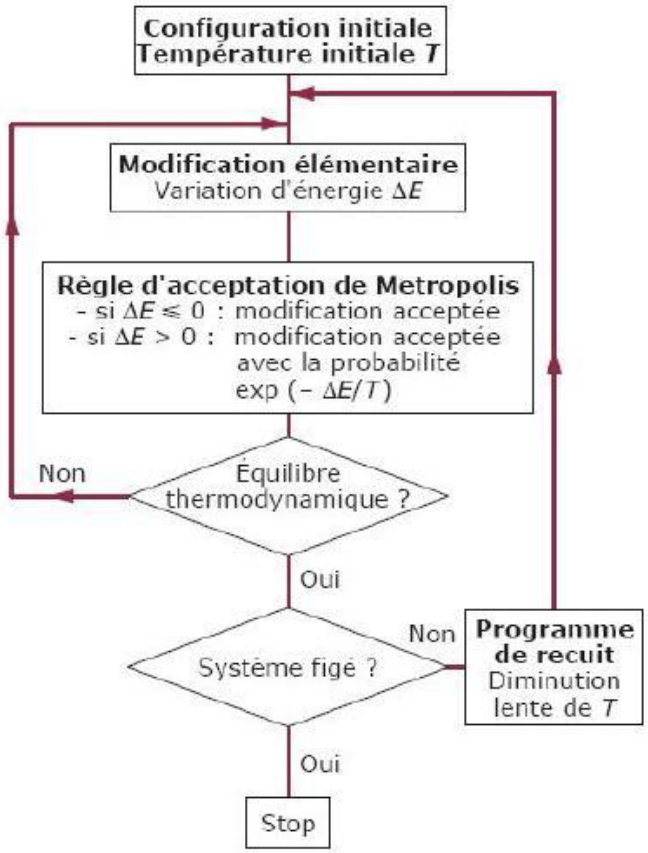

FIGURE 1.9 – *Fonctionnement de l'algorithme de recuit simulé.*

Algorithme 2: Recuit simulé

1. Engendrer une configuration initiale S_0 de S : $S \longleftarrow S_0$
2. Initialiser la température T en fonction du schéma de refroidissement
3. Répéter
4. Engendrer un voisin aléatoire S' de S
5. Calculer $\Delta E = f(S') - f(S)$
6. Si $\Delta E \leq 0$ alors $S \longleftarrow S'$
7. Sinon accepter S' comme la nouvelle solution avec la probabilité $P(E,T) = \exp^{-\frac{\Delta E}{T}}$
8. Fin si
9. Mettre T à jour en fonction du schéma de refroidissement (réduire la température)
10. Jusqu'à la condition d'arrêt
11. Retourner la meilleure configuration trouvée

Le choix de la température est primordial pour garantir l'équilibre entre l'intensification et la diversification des solutions dans l'espace de recherche. Premièrement, le choix de la température initiale dépend de la qualité de la solution de départ. Si cette solution est choisie aléatoirement, il faut prendre une température relativement élevée.
On utilise souvent la règle suivante :

$$T_{k+1} \longleftarrow T_k.\alpha$$

où $\alpha \in [0,1]$, un paramètre qui exprime la diminution de la température de l'itération k à $k+1$.

Avantages

– La méthode du recuit simulé a l'avantage d'être souple vis-à-vis des évolutions du problème et facile à implémenter,
– Contrairement aux méthodes de descente, SA évite le piège des optima locaux,
– Excellents résultats pour un nombre de problèmes complexes.

Inconvénients

– Nombreux tests nécessaires pour trouver les bons paramètres,
– Définir les voisinages permettant un calcul efficace de ΔE.

1.5.4 La recherche Tabou (Tabu Search)

La recherche tabou (TS) est une méthode de recherche locale combinée avec un ensemble de techniques permettant d'éviter d'être piégé dans un minimum local ou la répétition d'un cycle. La recherche tabou est introduite principalement par Glover (Glover 1986), Hansen (Hansen 1986), Glover et Laguna dans (Glover et Laguna 1997). Cette méthode a montré une grande efficacité pour la résolution des problèmes d'optimisation difficiles. En effet, à partir d'une solution initiale s dans un ensemble de solutions local S, des sous-ensembles de solution $N(s)$ appartenant au voisinage S sont générés. Par l'intermédiaire de la fonction d'évaluation nous retenons la solution qui améliore la valeur de f, choisie parmi l'ensemble de solutions voisines $N(s)$.
L'algorithme accepte parfois des solutions qui n'améliorent pas toujours la solution courante. Nous mettons en oeuvre une liste tabou (tabu list) T de longueur k contenant les k dernières solutions visitées, ce qui ne donne pas la possibilité a une solution déjà trouvée d'être acceptée et stockée dans la liste tabou. Alors le choix de la prochaine solution est effectué sur un ensemble des solutions voisines en dehors des éléments de cette liste tabou. Quand le nombre k est atteint, chaque nouvelle solution sélectionnée remplace la plus ancienne dans la liste. La construction de la liste tabou est basée sur le principe FIFO, c'est-à-dire le premier entré est le premier sorti. Comme critère d'arrêt on peut par exemple fixer un nombre maxi-

Algorithme 3: Recherche tabou

1. Initialisation :
   s_0 une solution initiale dans S
   $s^* \longleftarrow s_0$, $c* \longleftarrow f(s_0)$
   $T = \emptyset$
2. Générer un sous-ensemble de solution en voisinage de s_0
   $s' \in N(s_0)$ tel que $\forall x \in N(s_0)$, $f(x) \geq f(s')$ et $s' \notin T$
   Si $f(s') < c^*$ alors $s^* \longleftarrow s'$ et $c^* \longleftarrow f(s')$
   Mise à jour de T
3. Si la condition d'arrêt n'est pas satisfaite retour à l'étape 2

mum d'itérations sans amélioration de s*, ou bien fixer un temps limite après lequel la recherche doit s'arrêter.

Avantages et inconvénients

La recherche tabou est une méthode de recherche locale, et la structure de son algorithme de base est proche de celle du recuit simulé, avec l'avantage d'avoir un paramétrage simplifié : le paramétrage consistera d'abord à trouver une valeur indicative t d'itérations pendant lesquelles les mouvements sont interdits. Il faudra également choisir une stratégie de mémorisation. En revanche, la méthode tabou exige une gestion de la mémoire de plus en plus lourde en mettant des stratégies de mémorisation complexe. L'efficacité de la méthode tabou offre son utilisation dans plusieurs problèmes d'optimisation combinatoire classiques tels que

le problème de voyageur de commerce, le problème d'ordonnancement, le problème de tournéees de véhicules, etc.

Conclusion du chapitre

Nous pouvons conclure que les métaheuristiques présentent une classe de méthodes approchées adaptables à une grande variété de problèmes d'optimisation combinatoire et mènent à des résultats pertinents. Mais il existe assez peu de contributions permettant de comprendre la raison de cette efficacité, et aucune preuve ne peut montrer qu'une métaheuristique sera plus efficace qu'une autre sur un problème donné. Certaines métaheuristiques présentent l'avantage d'être simples à mettre en oeuvre, comme le cas du recuit simulé, d'autres sont bien adaptées à la résolution de certaines classes de problème, très contraints, comme le système de colonies de fourmis. La qualité des solutions trouvées par les métaheuristiques dépend de leur paramétrage, et de l'équilibre entre un balayage de tout l'espace des solutions (diversification) et une exploration locale (l'intensification).

Bibliographie

V. Cerný. A thermodynamical approach to the traveling salesman problem : an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1) :41–51, 1985. (Cité page 20.)

A. Colorni, M. Dorigo, et V. Maniezzo. Distributed optimization by ant colonies. in varela, f. and bourgine, p., editors, proceedings of ECAL'91. *First European Conference on Artificial Life, Paris, France*, pages 134–142, 1992a. (Cité page 17.)

A. Colorni, M. Dorigo, et V. Maniezzo. An investigation of some proprieties of an "ant algorithm". *In Manner and Manderick*, pages 509–520, 1992b. (Cité page 16.)

D. Costa et A. Hertz. Ants can color graphs. *Journal of the Operational Research Society*, 48 :295–305, 1997. (Cité page 17.)

M. Dorigo, G. Caro, et L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2) :137–172, 1999. (Cité page 19.)

M. Dorigo, A. Colorni, et V. Maniezzo. The ant system : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26, No.1 :29–41, 1996. (Cité pages 17 et 18.)

M. Dorigo et L. M. Gambardella. Ant colony system : A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1) :53–66, 1997. (Cité page 19.)

L. M. Gambardella, E. D. Taillard, et M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2) :167–176, 1999. (Cité page 17.)

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13 :533–549, 1986. (Cité page 23.)

F. Glover et M. Laguna. Tabu search. *Kluwer Academic Publishers, Norwell, MA, second edition*, 1997. (Cité page 23.)

D. Goldberg. Genetic algorithms. addison wesley. *Addison Wesley*, ISBN : 0-201-15767-5, 1989a. (Cité pages 10 et 11.)

D. Goldberg. Genetic algorithms in search, optimization and machine learning. *Addison Wesley*, 1989b. (Cité page 10.)

P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. *présenté au Congress on Numerical Methods in Combinatorial Optimization, Capri, Italie*, 1986. (Cité page 23.)

J. H. Holland. Adaptation in natural and artificial systems. University of Michigan press, 1975. (Cité page 9.)

S. Kirkpatrik, C. D. Gelatt, et K. P. Vecchi. Optimization by simulated annealing. science. 220 :671–680, 1983. (Cité page 20.)

A. H. Land et A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28 :497–520, 1960. (Cité page 3.)

S. Mazzeo et I. Loiseau. An ant colony algorithm for the capacitated vehicle routing problem. *Electronic Notes in Discrete Mathematics*, 18 :181–186, 2004. (Cité page 17.)

N. Metropolis, M. N. Rosenbluth, et H. A. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21(6) : 10871092, 1953. (Cité page 20.)

H. Mülhenbein. *Evolutionary Algorithms : Theory and Applications. Wiley*, 1993. (Cité page 13.)

J. Puchinger et G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification, in proceedings of the first international work-coriference on the interplay between natural and artificial computation. Springer (Ed), Las Palmas, Spain, LNCS :41–53, 2005. (Cité page 3.)

A. Schrijver. Theory of linear and integer programming. *Wiley and Sons*, 1986. (Cité page 6.)

T. Stützle et H. Hoos. The max-min ant system and local search for the traveling salesman problem. *In Proceedings of ICEC'97. IEEE fourth international conference on evolutionary computation. New York : IEEE Press*, pages 308–313, 1997. (Cité page 19.)

T. Stützle et H. Hoos Holger. Max min ant system. *Future Generation Computer Systems*, 16(8) :889–914, 2000. (Cité page 19.)

Ce document a été préparé à l'aide de l'éditeur de texte GNU Emacs et du logiciel de composition typographique LaTeX 2_ε.