REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUREET ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI, TIZI OUZOU

FACULTE DES SCIENCES

DEPARTEMENT MATHEMATIQUE

MEMOIRE DE MASTER

EN

MATHEMATIQUES

Spécialité

Recherche Opérationnelle : méthodes d'aide à la décision

Thème

Quelques méthodes de résolutions en optimisation combinatoire

Présenté par :

Kherbouche Lynda

Oubahri Zohra

Dérigé par :

Mr M. Aouane UMMTO Promoteur

Mr K.Kasdi UMMTO Président

Mr A.Belhadj UMMTO Examinateur

Soutenu: Octobre 2017

Dédicaces

Nous dédions ce modeste travail:
A nos familles,
A nos amis,
Et à nos professeurs.

Remerciements

Avant tout propos, nous remercions « Dieu » le tout puissant qui nous a donné la sagesse et la santé pour faire ce modeste travail.

C'est avec un grand plaisir que nous exprimons nos gratitudes et nos sincères remerciements à notre promoteur : Mr Aouane Mohammed pour son orientation et encadrement dans l'élaboration de ce mémoire de fin d'étude, nos remerciements vont aussi à tous les membres du jury pour avoir accepter d'honorer par leur jugement notre travail, et pour leur participations au jury.

Toutes nos reconnaissances sont adressées à tous les enseignants qui nous ont suivi infatigablement durant tout notre cursus universitaire.

Nous tenons à exprimer tout au fond de nos cœurs les reconnaissances à nos familles qui nous ont offert toujours un appui sûr par leurs soutiens et leurs encouragements. Nos vifs remerciements vont également à tous ceux qu'ont contribués de loin ou de prés à la réalisation de ce travail.

Table des matières

Chapitre3 : méthodes et algorithmes de résolution en optimisation combinatoire

Chapitre 4 : application de deux heuristiques pour le problème de voyageur de commerce

La recherche opérationnelle est une méthode scientifique d'aide à la décision. Son histoire est récente : elle remonte à la seconde guerre mondiale. C'est en Angleterre, que cette discipline reçoit son nom et prouve son efficacité par le rapprochement de scientifique et de militaires chargés de préparer les grandes décisions liées aux opérations. Dés la fin de la guerre, le succès des techniques de la recherche opérationnelle n'a cessé de s'étendre parmi l'éventail des domaines de décision.

L'optimisation peut être définie comme une branche mathématique orientée vers la recherche de la meilleure façon d'opérer des choix en vue d'aboutir au résultat visé ou au meilleur résultat possible. Elle fait partie de la science d'aides à la décision dans la mesure où elle propose des modèles conceptuels en vue d'analyser et de maîtriser des situations complexes pour permettre aux décideurs de comprendre et d'évaluer les enjeux et d'attribuer ou de faire les choix les plus efficaces.

La résolution des problèmes combinatoires est assez délicate puisque le nombre fini et ou dénombrable et ou infini 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). Ces approches de résolution peuvent être classées en deux catégories : les méthodes exactes et les méthodes approchées.

Les méthodes exactes gagnent en l'optimalité des solutions et perdent en temps d'exécution, ce qui est l'inverse pour les méthodes approchées qui ne garantissent pas de trouver une solution exacte, mais seulement une approximation en des temps raisonnables de calculs.

Notre travail est divisé en 4 chapitres :

Le premier chapitre est consacré à rappeler des définitions ainsi que les concepts de base utilisés dans la théorie des graphes.

Le deuxième chapitre est consacré aux généralités sur les problèmes d'optimisation combinatoire.

Le troisième chapitre traite quelques méthodes de résolution en optimisation combinatoire.

Dans le quatrième chapitre, on applique deux heuristiques sur un exemple de problème de voyageur de commerce : La méthode de descente et celle de la recherche taboue.

On termine notre travail par une conclusion générale.

Introduction :

Avant d'entamer notre travail il est nécessaire de commencer par énoncer certaines définitions et concepts que nous avons jugés être nécessaires à la compréhension de notre problématique.

1-1 Introduction à la théorie des graphes :

L'histoire de la théorie des graphes débute avec les travaux d'Euler au XVIIIe siècle et trouve son origine dans l'étude de certains problèmes, tels que celui des ponts de Königsberg (les habitants de Königsberg se demandaient s'il était possible, en partant d'un quartier quelconque de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir à leur point de départ comme l'indique la figure 1-1), la marche du cavalier sur l'échiquier ou le problème de coloriage.

La théorie des graphes s'est alors développée dans diverses disciplines telles que la chimie, la biologie…etc.

De manière générale, un graphe permet de représenter la structure, les connexions d'un ensemble complexe en exprimant les relations entre ses éléments : réseau de communication, réseaux routiers, circuits électriques . . . etc.

Les graphes constituent donc une méthode qui permet de modéliser une grande variété de problèmes en se ramenant à l'étude de sommets et d'arcs.

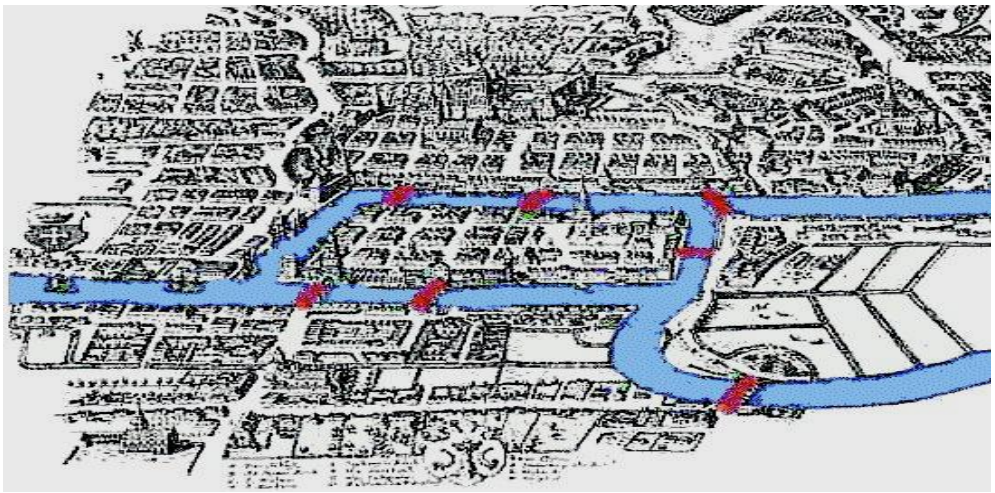


Figure1-1

1-2 Définitions générales

1-2-1 Graphes orientés :

Un graphe $G = (X, U)$ est le couple constitué:

1. Par un ensemble $X = (x_1, x_2, \dots, x_n)$appelés sommet.

2. Par une famille $U = (u_1, u_2, \dots, u_m)$d'éléments du produit cartésien $X * X = \{(x, y)/ \ x \in X$ et $y \in X$ appelés arcs.

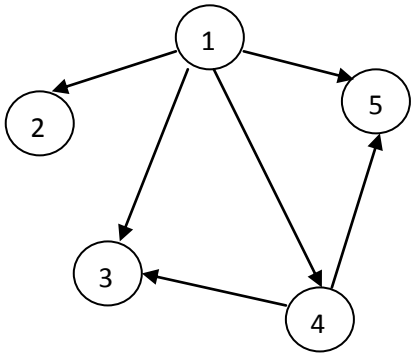


Figure1-2: Un graphe orienté.

Degré d'un sommet dans un graphe orienté :

Soient *G = (X, U); |X| = n* et *|U| = m* et les deux ensemble suivants:

x ∈ X; $D^+(x)$= {u∈ U/u = (x, y), avec *y ∈X}, $D^-(x)$ = {u ∈U/u = (y, x)et y ∈ X}*

$d_G^+(x)$ =$|D^+(x)|$ =demi-degré extérieur de *x* dans *G*.

$d_G^-(x)$ =$|D^-(x)|$ =demi-degré intérieur de *x* dans *G*.

$d(x) = d^-(x) + d^+(x)$, est appelé degré de *x* dans *G*.

Successeurs et prédécesseurs d'un sommet :

L'ensemble de successeurs d'un sommet, noté $\Gamma^+(x)$, regroupe toutes les extrémités finales des arcs ayant comme extrémité initiale *x*.

Symétriquement, l'ensemble de prédécesseurs d'un sommet *x*, noté $\Gamma^-(x)$, regroupe toutes les extrémités initiales des arcs ayant comme extrémité finale *x*.

Le voisinage de *x* est l'ensemble $\Gamma(x) = \Gamma^+(x) \cup \Gamma^-(x)$.

Si *G* n'est pas orienté on définit simplement $\Gamma(x)$ comme le sous ensemble des sommets de *X* adjacents à *x* (*N(x)*).

1-2-2 Graphes non orientés :

Un **graphe** fini $G = (X, E)$ est défini par l'ensemble fini $X = \{x_1, x_2, \dots, x_n\}$dont les éléments sont appelés **sommets,** et par l'ensemble fini.$E = \{e_1, e_2, \dots, e_m\}$ dont les éléments sont appelés **arêtes.** Une arête *e* de l'ensemble *E* est définie par une paire non ordonnée de sommets, appelés les extrémités de *e*. Si l'arête *e* relie deux sommets, on dira que ces sommets sont **adjacents**, ou **incidents** avec *e*, ou bien que l'arête *e* est incidente avec ces sommets. On appelle **ordre** d'un graphe le nombre de sommets *n* de ce graphe.

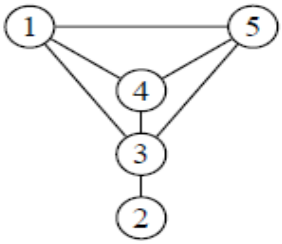


Figure1-3

1-2-3 Graphe complémentaire :

Soit $G = (X, E)$ un graphe, son complémentaire noté $\bar{G}$ contenant les même sommets mais des arêtes différentes.

Figure1-4 : un graphe *G* et son complémentaire

Degré d'un graphe

Le degré d'un graphe est le degré maximum de tous ses sommets. Dans l'exemple ci-dessous le degré du graphe est 4, à cause du sommet v_3.

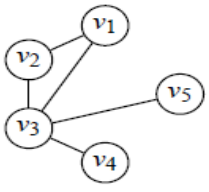


Figure1-5

Un graphe dont tous les sommets ont le même degré est dit **régulier**. Si le degré commun est *k*, alors on dit que le graphe est *k*-régulier.

1-2-4 Chaînes et cycles

Une chaîne dans un graphe *G*, est une suite ayant pour éléments alternativement des sommets et des arêtes, notée $C_n = x_1, e_1, x_2 \ldots\ldots, e_n, x_{n+1}$. La taille d'une chaîne est le nombre de ses arêtes.

On dira que la chaîne relie le premier sommet de la suite au dernier sommet.

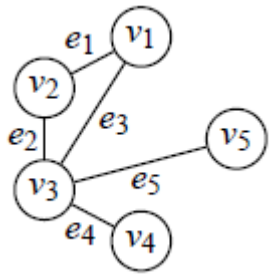


Figure 1-6

Une chaîne élémentaire (respectivement simple) : si chacun de ses sommets (respectivement arêtes) apparaît au plus une fois.

Dans le graphe précédent, $(v_1, e_1, v_2, e_2, v_3)$ est une chaîne simple et élémentaire.

Chaîne hamiltonienne : dans un graphe *G* une chaîne est hamiltonienne si, elle passe une et une seule fois par tous les sommets de *G*.

Chaîne eulérienne: dans un graphe *G*, une chaîne passant une et une seule fois par chacune des arêtes de *G* est appelée chaîne eulérienne.

Distance : La distance entre deux sommets est la longueur de la plus petite chaîne les reliant.

Diamètre : Le diamètre d'un graphe est le maximum des distances.

Cycle : on appelle cycle une chaîne dont les extrémités se confondent. Dans le graphe précédent, $(v_1, e_1, v_2, e_2, v_3, e_3, v_1)$est un cycle.

Cycle hamiltonien : c'est un cycle passant une et une seule fois par chacun des sommets de *G* sauf le sommet de départ.

Cycle eulérien: c'est un cycle passant une et une seule fois par chacune des arêtes de *G*.

1-2-5 Chemins et circuits

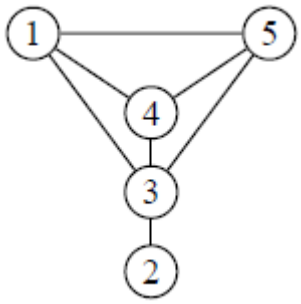
Chemin : c'est une chaîne orientée dans un sens unique.

Circuit : c'est un chemin fermé (l'extrémité finale et initiale représentées par le même sommet).

1-3 La représentation matricielle d'un graphe

1-3-1 Matrice d'adjacences :

On peut représenter un graphe simple par une **matrice d'adjacence**. Une matrice ($n \times m$) est un tableau de n lignes et m colonnes. (i, j) désigne l'intersection de la ligne i et de la colonne j . Dans une matrice d'adjacences, les lignes et les colonnes représentent les sommets du graphe. Un « 1 » à la position (i, j) signifie que le sommet i est adjacent au sommet J



$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 1-7

Cette matrice a plusieurs caractéristiques :

Elle est carrée, elle est symétrique : $m_{i\,j} = m_{ji}$. Une fois que l'on fixe l'ordre des sommets, il existe une matrice d'adjacences unique pour chaque graphe. Celle-ci n'est la matrice d'adjacences d'aucun autre graphe.

1-3-2 Liste d'adjacence :

On peut aussi représenter un graphe simple en donnant pour chacun de ses sommets la liste des sommets auxquels il est adjacent. Ce sont les **listes d'adjacences**.

Si on prend l'exemple précédent on aura comme listes d'adjacences :

1 :3, 4, 5 ;

2 :3 ;

3 :2, 4, 1, 5 ;

4 :1, 5, 3 ;

5 :1 ,4, 3.

1-4 Quelques types de graphes

1-4-1 graphe simple: un graphe est dit simple si, au plus, une arête relie deux sommets et s'il n'y a pas de boucle sur un sommet. Si deux sommets sont reliés par plusieurs arêtes alors le graphe est dit un multi graphe.

1-4-2 graphe connexe : s'il est possible à partir de n'importe quel sommet, de rejoindre tous les autres en suivant les arêtes. Un graphe non connexe se décompose en **composantes connexes**.

Un graphe est **complet** si chaque sommet du graphe est adjacent à tous les autres sommets comme le montre la figure suivante :

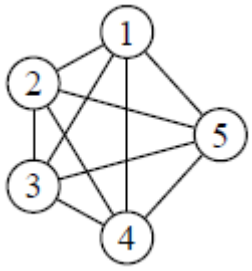


Figure1-8: Graphe complet à cinq sommets.

1-4-3 Graphe biparti

Soit *G = (V, E)* un graphe. On dit que *G* est biparti si on peut trouver une partition $\{V_1, V_2\}$ de *V* telle que : $\forall\, x, y \in V, (x, y) \in E$ si et seulement si $(x, y) \in V_1 * V_2 \cup V_2 * V_1$ Autrement dit, une arête ne peut relier qu'un sommet de V_1 à un sommet de V_2. On notera plus simplement

$G = (V_1, V_2, E)$ si *G* est biparti.

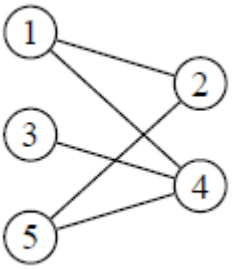
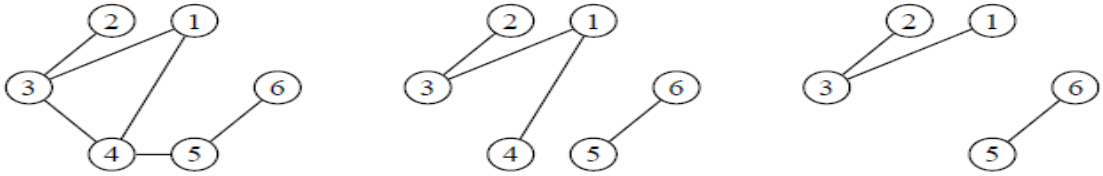


Figure1-9

1-4-4 Graphe partiel et sous-graphe

Soit $G = (V, E)$ un graphe. Le graphe $G' = (V, E')$ est un **graphe partiel** de G, si E' est inclus dans E. Autrement dit, on obtient G' en enlevant une ou plusieurs arêtes au graphe G.

Pour un sous-ensemble de sommets A inclus dans V, le **sous-graphe** de G induit par A est le graphe $G = (A, E(A))$ dont l'ensemble des sommets est A et l'ensemble des arêtes $E(A)$ est formé de toutes les arêtes de G ayant leurs deux extrémités dans A. Autrement dit, on obtient G' en enlevant un ou plusieurs sommets au graphe G, ainsi que toutes les arêtes incidentes à ces sommets.



Graphe G graphe partiel de G sous-graphe de G

Figure 1-10

1-4-5 Arbre

On appelle **arbre** tout graphe connexe sans cycle. Un graphe sans cycle mais non connexe est appelé une **forêt**.

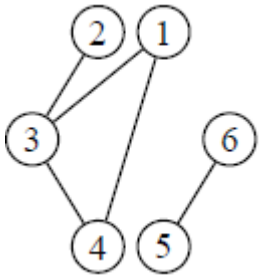
Une **feuille** ou **sommet pendant** est un sommet de degré 1.

Arbre forêt

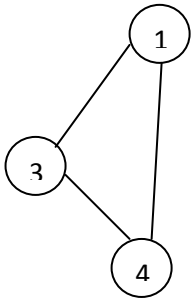
Figure1-11

1-4-6 Clique : Une partie *V'* de *V* définie une clique de *G* si le sous graphe engendre par *V'* est complet.

On note le cardinal d'une plus grande clique de *G* par $\omega(G)$.



Graphe *G*



clique de *G* *(ω(G) =3)*

Figure1-12

Partition de sommets en cliques (θ(G)) :

$C = (c_1, c_2, \ldots\ldots\ldots, c_k)$ est une partition en clique de *G* si :

$C_i \neq \emptyset \quad \forall\, i = 1\ldots\ldots k$

$C_i \cap C_j = \emptyset \quad , i \neq j$

$\bigcup_{i=1}^{k} C_i = V$

On note $\theta(G)$ le cardinal d'une partition minimale de *G*.

Corollaire

On a toujours $\propto(G) \leq \theta(G)$:

Si *S* est un ensemble stable, et *C* est une partition en clique avec $|S|=|C|$, alors S est un ensemble stable maximum, et C est une partition minimum.

En effet, pour un S stable et une partition $C = (c_1, c_2, \ldots, c_k)$, on a

$|S \cap C_i| \leq 1 \qquad (I = 1, 2\ldots, k)$

D'où $|S| \leq |C|$

D'où $\propto (G) = \max |S| \leq \min |C| = \theta(G)$

1-5 Coloration des graphes

1-5-1 La coloration simple

Soit $G = (X, E)$ un graphe simple non orienté, la coloration des sommets d'un graphe consiste à affecter à chaque sommet de ce graphe une couleur de telle sorte que deux sommets adjacents n'aient pas la même couleur.

Une coloration avec *K* couleurs notée *K*-coloration est donc une partition de l'ensemble des sommets en *K* parties stables.

Nombre chromatique :

On appelle nombre chromatique d'un graphe G le plus petit nombre de couleurs nécessaires pour colorier les sommets, de sorte que deus sommets adjacents ne soient pas de même couleur

On le désigne par γ(G).

Remarque :

Un graphe G vérifiant :

- $\propto$(G)= θ(G) ou $\gamma(G)$=$\omega(G)$ est dit un graphe parfait
- Les graphes bipartis sont des graphes parfaits.

Proposition :

Le nombre chromatique du graphe sera supérieur ou égal à l'ordre de sa plus grande clique, que l'on note (G) . Autrement dit $\gamma\ (G) \geq \omega(G)$

Preuve :

Puisque, par définition, dans une clique d'ordre *k*, tous les sommets sont adjacents entre eux, il faudra *k* couleurs. Donc, il faudra au moins $\omega(G)$ couleurs pour colorier le graphe *G*.

Proposition: soit r le plus grand degré des sommets de *G*, alors $\gamma(G) \leq r$+1

Preuve : Soit un graphe et r le degré maximum de ses sommets. Donnons-nous une palette de (r+1) couleurs. Pour chaque sommet du graphe on peut tenir le raisonnement suivant : ce sommet est adjacent à r sommets au plus, et le nombre de couleurs déjà utilisées pour colorier ces sommets est donc inférieur ou égal à r . Il reste donc au moins une couleur non utilisée dans la palette, avec laquelle nous pouvons colorier notre sommet.

1-5-2 Coloration par liste

K-assignation de listes

Dans un graphe$G = (X, E)$, une k-assignation de listes est une fonction L qui à chaque sommet $x \in X$ associe une liste $L(x)$ de k entiers.

K-liste coloriable

Un graphe *G* est k-liste coloriable si pour toute assignation de liste L, G admet une coloration c telle que pour toute arête $xy \in E$, $c(x) \neq c(y)$ et pour tout sommet $x \in X$, $c(x) \in L(x)$.

Nombre chromatique par liste

Le plus petit k tel que *G* soit k-liste coloriable est le nombre chromatique par listes de *G*, noté $\mathrm{ch}(G)$.

1-6 Couplages

Soit *G* un graphe simple. Un **couplage** *C* de *G* est un sous-graphe partiel de *G*.

C'est aussi un ensemble d'arêtes deux à deux non-adjacentes.

Un sommet *v* est **saturé** par un couplage *C* si *v* est l'extrémité d'une arête de *C*. Dans le cas contraire, *v* est **insaturé**.

Un **couplage maximum** est un couplage contenant le plus grand nombre possible d'arêtes.

Un graphe peut posséder plusieurs couplages maximums

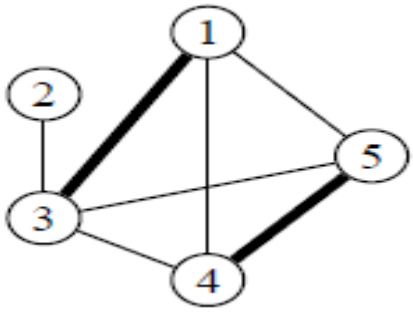
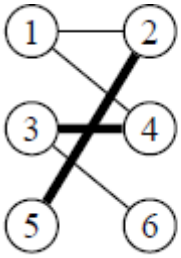


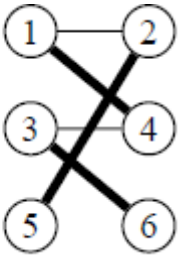
Figure1-13 : un couplage maximum de *G*

Les sommets 1, 3, 4 et 5 sont saturés

Un couplage parfait est un couplage où chaque sommet du graphe est saturé.



Un couplage.



Un couplage maximum et parfait.

Figure1-14

1-7 Programme linéaire :

Un Programme Linéaire (PL) est un problème d'optimisation consistant à maximiser(ou minimiser) une fonction objectif linéaire de variables soumises à un ensemble de contraintes exprimées sous forme d'équations ou d'inéquations linéaires.

La forme générale d'un programme linéaire est la suivante :

Un programme linéaire sous forme standard se présente comme suit :

$$(P_1)\begin{cases} \max z = c.x \\ sc \quad Ax = b \\ \quad\quad x \geq 0 \end{cases}$$

On peut toujours considérer que l'on a rang(A) = m. En effet, rang(A) < m signifie qu'une ou plusieurs lignes de la matrice A sont une combinaison linéaire des autres lignes. Ceci revient à dire que les contraintes correspondantes sont soit redondantes, soit incompatibles avec les autres (suivant la valeur des seconds membresb_i) ; dans le premier cas on peut éliminer ces contraintes, dans le second cas, le système A.x = b n'a pas de solution. Tout programme linéaire peut s'exprimer sous forme standard (P_1), en ajoutant éventuellement certaines variables appelées variables d'écart (par exemple, une inéquation. a.x ≤ b devient l'équation

a.x + s = b, s ≥0).

1-7-1 Programmation linéaire en nombres entiers

De nombreux problèmes d'optimisation, issus de domaines d'applications très divers, peuvent être modélisés comme des programmes linéaires dont les variables sont contraintes à être entières.

Les programmes linéaires ainsi obtenus sont appelés programmes linéaires en nombres entiers (PLNE). Ils sont souvent difficiles à résoudre, du fait notamment que l'espace de recherche est

discret. Toutefois, la programmation linéaire en nombres entiers est un outil utile de modélisation des problèmes et de nombreuses approches de résolution ont été développées.

La formulation générale d'un PLNE est la suivante :

$$(P_2)\begin{cases} \max z = c.x \\ sc \quad Ax \leq b \\ \quad\quad x \in N^n \end{cases}$$

En général, on suppose que A et b sont composés uniquement de valeurs entières

Un programme linéaire en nombres entiers peut, généralement, s'exprimer sous la forme d'un programme linéaire en variables booléennes (PL en 0−1).

Les méthodes classiques de programmation linéaire ne peuvent être utilisées systématiquement pour les PLNE puisqu'elles cherchent un sommet optimal du polyèdre des contraintes $\{x / A.x \leq b, x \geq 0\}$ qui, en général, n'a pas de coordonnées entières. De différentes approches ont été étudiées pour la recherche de solutions optimales entières des PLNE. Elles font appel essentiellement à des algorithmes de recherche arborescente par séparation et évaluation (branch and bound) et à des méthodes de coupes.

Introduction :

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Elle consiste à comprendre un problème réel et pouvoir le transformer en un modèle mathématique à fin d'extraire ses propriétés structurelles et le résoudre. Bien que les problèmes d'optimisation combinatoire sont souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent, donc, pas à ce jour de solution algorithmique efficace valable pour toutes les données. Les nombreuses méthodes de résolution développées en recherche opérationnelle (RO) peuvent être classées sommairement en deux grandes catégories : les méthodes exactes qui garantissent l'optimalité de la solution et les méthodes approchées qui perdent la complétude pour gagner en efficacité. Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent d'une manière implicite, l'ensemble des solutions de l'espace de recherche. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'année) telles les techniques de séparation et évaluation progressive (SEP). Ces méthodes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrès réalisés (notamment en programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille de problème, les méthodes exactes rencontrent, généralement, des difficultés face aux applications de taille importante.

Les méthodes approchées, quant à elles, utilisées pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. Elles sont utilisées depuis longtemps par de nombreux praticiens. Leur but est de trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir l'optimalité de la solution. Les méthodes approchées sont fondées principalement sur diverses heuristiques.

Résoudre un problème d'optimisation consiste à trouver une solution optimale, c'est-à-dire trouver une solution réalisable qui minimise ou maximise, selon le contexte, la fonction objectif. De nombreux problèmes d'optimisation combinatoire peuvent être formulés comme des PLNE particuliers ainsi, un algorithme permettant de résoudre un PLNE permet de résoudre beaucoup de problèmes d'optimisation combinatoire.

2-1 Optimisation combinatoire :

un problème d'optimisation combinatoire est un problème de recherche qui consiste a explorer un espace contenant l'ensemble de toutes les solutions potentielles réalisables, dans le but de trouver la solution optimale, sinon la plus proche possible de l'optimum permettant de minimiser ou maximiser une fonction dite fonction objectif .

C'est-à-dire :

Domaine qui étudie les problèmes de la forme :

$$Max_{x \in X} f(x)$$

où X est un ensemble fini.

Définition : Une instance I d'un problème est l'état où il peut se trouver. Celle-ci est caractérisée par la taille de ce problème ainsi que les relations liant ses éléments.

Remarque

La même définition peut se donner en utilisant le maximum sachant que

$$\underset{s \in S}{Max} f(s) = -\underset{s \in S}{Min} -f(s)$$

2-2 Formulation mathématique des problèmes d'optimisation

Les problèmes d'optimisation combinatoire peuvent être formulés comme suit:

2-2-1 Problèmes mono-objectifs

Un problème d'optimisation est généralement formulé comme un problème de minimisation ou de maximisation, et s'écrit sous la forme suivante:

$$\begin{cases} min\, f(x), & telque \\ g_i(x) \leq 0, & i = 1, \dots, m \\ h_j(x) = 0, & j = 1, \dots, p \\ x \in S \subset R^n \end{cases}$$

Où f est la fonction à minimiser, appelée fonction coût ou fonction objectif, x représente le vecteur des variables d'optimisation, $g_i$sont les contraintes d'inégalité et h_j les contraintes d'égalité, et S est l'espace des variables (appelé aussi espace de recherche). S indique quel type de variables sont considérées : réelles, entières, mixtes (réelles et entières dans un même problème), discrètes, bornées, etc. Un point x_A est appelé un point admissible si $x_A \in S$ et si les contraintes d'optimisation sont satisfaites :

$g_i(x_A) \leq 0$, i= 1,..., m et $h_j(x_A) = 0$, j = 1,..., p.

S : Espace de recherche, de configurations

2-2-2 Problèmes multi-objectifs

Un problème d'optimisation combinatoire peut avoir plusieurs fonctions objectif, il s'agit alors d'un problème d'optimisation multicritère ou multi-objectif.

D'un point de vue mathématique, il est représenté, dans le cas où le vecteur F regroupe k fonctions objectif, de la façon suivante:

$$\begin{cases} min\, F(x), & telque \\ g_i(x) \leq 0, & i = 1, \dots, m \\ h_j(x) = 0, & j = 1, \dots, p \\ x \in S \subset R^n \end{cases}$$

De ce fait, résoudre un problème d'optimisation combinatoire nécessite l'étude de trois points suivants:

- La définition de l'ensemble des solutions réalisables,
- L'expression de l'objectif à optimiser,
- Le choix de la méthode d'optimisation (exacte ou approchée) à utiliser.

Les deux premiers points relèvent de la modélisation du problème, le troisième de sa résolution.

Voisinage

Soit *x* une solution, on dit que x^* est une solution voisine de *x*, si on peut obtenir x^* en modifiant légèrement *x*. Le voisinage V (*x*) de *x* est l'ensemble des solutions voisines de *x*.

Définition :

On dit que :

- x^* est un minimum local si $f(x^*) \leq f(x)\ \forall\, x \in V(x^*)$.
- x^* est un minimum global si $f(x^*) \leq f(x)\ \forall\, x \in D_f$
- x^* est un maximum local si $f(x^*) \geq f(x)\ \forall\, x \in V(x^*)$.
- x^* est un maximum global si $f(x^*) \geq f(x)\ \forall\, x \in D_f$.

2-3 La théorie de la complexité

Un algorithme

C'est une suite d'instructions ordonnées finies agissent sur des données en entrée qui à la fin donnent la solution du problème.

Un algorithme peut se caractérisé par le temps et l'espace qu'il utilise pour arriver à la solution. Ces deux notions de temps et d'espace sont des indicateurs sur l'efficacité ou non d'un algorithme, l'étude de cette caractéristique est connue par le nom de la complexité algorithmique.

La complexité algorithmique.

D'une manière générale, pour résoudre un problème, on est appelé à trouver l'algorithme le plus efficace. Cette notion d'efficacité induit normalement toutes les ressources de calcul nécessaire pour exécuter un algorithme. Or, le temps d'exécution est généralement le facteur dominant pour déterminer si un algorithme est assez efficace pour être utilisé dans la pratique, pour cela on se concentre principalement sur cette ressource.

On appelle complexité en temps d'un algorithme le nombre d'instructions élémentaires mises en œuvre dans cet algorithme afin de résoudre un problème donné. Une instruction élémentaire sera une affectation, une comparaison, une opération algébrique, la lecture l'écriture…..etc. Il existe aussi une mesure de la performance des algorithmes en termes d'espace mémoire dite complexité en espace. Le décompte précis de toutes les instructions d'un programme risque d'être assez pénible, et qu'entre deux exécutions de même algorithme avec un jeu de paramètre différent, le nombre d'instructions exécutées peut changer, on se contentera, en générale, d'apprécier un ordre de grandeur de ce nombre d'instructions. C'est ce qu'on désigne sous le nom de complexité de l'algorithme. Donc pour mesurer la complexité temporelle d'un algorithme, on s'intéresse plutôt aux opérations les plus coûteuses telles que la racine carrée, log, expo…etc.

La complexité d'un problème est la complexité du meilleur algorithme qui permet de le résoudre. Si cet algorithme est polynomial, le problème est dit facile, autrement le problème est difficile.

La classe P (polynomial time) :

Soit π un problème quelconque. Si pour toute instance I de π , une solution de I peut être déterminée algorithmiquement en un temps polynomial alors π est appelé problème

polynomial et l'algorithme qui le résout algorithme polynomial(ou un algorithme efficace). Les problèmes polynomiaux constituent la classe P.

La classe NP (Not Deterministe Plynomial Time):

Soit un problème quelconque π et I ses instances pour lesquelles la réponse est oui. Le problème π appartient à la classe NP s'il existe un algorithme polynomial qui permet de vérifier que cette réponse est bien oui pour ces instances I .Notons que NP ne veut en aucun cas dire Non Polynomial mais Not Determinist Ploynomial.

Remarque : problème de décision

Un problème de décision est un problème où la résolution se limite à la réponse par « oui » ou « non » à la question de savoir s'il existe une solution au problème**.**

Réduction polynomiale

On dit qu'un problème P_1 se réduit polynomialement en un problèmeP_2, s'il existe un algorithme polynomial A construisant, à partir d'une donnée D_1de P_1, une donnée D_2 de P_2 telle que la réponse pour D_1 soit oui si et seulement si la réponse pour D_2 est oui.

Problème NP-Complet

Un problème de décision π est NP-complet s'il satisfait les deux conditions suivantes :

$\pi \in$NP, et tout problème NP se réduit à π en temps polynomial.

Les problèmes NP-Complet sont liés par une relation d'équivalence dans le sens où s'il existe un algorithme polynomial pour un des problèmes de cette classe alors tous les problèmes NP-Complet pourront être résolus en un temps polynomial.

Cette classe englobe les problèmes les plus étudiés de l'optimisation combinatoire.

Problème NP-difficile

Un problème de NP est dit NP-difficile, si et seulement si il existe un problème NP-Complet qu'est réductible à lui en temps polynomial.

De cette définition, on conclut que pour montrer qu'un problème d'optimisation est NP-difficile, il suffit de montrer que le problème de décision associé à lui est NP-complet.

Ceci explique pourquoi, lors de l'étude d'un nouveau problème, on commence par chercher à classer ce problème. Si l'on parvient à montrer qu'il est polynomial, le problème sera résolu. Si par contre, on parvient à montrer qu'il est NP-complet, la recherche d'un algorithme exact pour résoudre un tel problème ne sera pas de première priorité, et il sera approprié de se concentrer sur des méthodes heuristiques que la plupart des spécialistes de l'optimisation combinatoire ont orienté leurs recherches pour les développer. Une méthode heuristique est souvent définie comme une procédure exploitant au mieux la structure du problème considéré,

dans le but de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible.

2-4 Les grandes classes des problèmes d'optimisation combinatoire :

Les problèmes en optimisation combinatoire sont classés en fonction de leur modélisation. C'est-à-dire de la façon dont ils sont exprimés en termes mathématiques. Donc des problèmes différents dans leur forme peuvent appartenir à une même classe.

De ce faite, tout problème d'optimisation combinatoire appartient à une classe des classes suivantes :

- Les problèmes de sac à dos.
- Les problèmes de tournées.
- Les problèmes d'affectation.
- Les problèmes d'ordonnancement.
- Les problèmes de flot.

Le problème du sac-à-dos

Considérons *n* objets, notés *i = 1,..., n* apportant chacun une utilité u mais possédant un poids p_i. On veut ranger ces objets dans un « sac » de capacité *c*. Le problème de sac-à-dos (knapsack) consiste à choisir les objets à prendre parmi les *n* objets de manière à avoir une utilité maximale et respecter la contrainte de la capacité, *c*, à ne pas dépasser.

La formulation PLNE du problème de sac-à-dos est très simple. On utilise pour chaque objet $i \in \{1, \dots, n\}$, une variable binaire x_i correspond à 1 si l'objet *i* est pris 0 sinon. Le problème du sac-à-dos est modélisé comme suit :

$$\begin{cases} Max \sum_{i=1}^{n} u_i x_i \\ p_i x_i \leq c \\ x_i \in \{0,1\} \; \forall \; i = \{1, \dots, n\} \end{cases}$$

Le problème du voyageur de commerce (PVC)

Un voyageur de commerce ayant *n* villes à visiter souhaite établir une tournée qui lui permette de passer une et une seule fois dans chaque ville pour finalement revenir à son point de départ, ceci en minimisant la longueur du chemin parcouru. Etant donné un graphe *G=(X, U)* dans lequel l'ensemble *X* des sommets représentent les villes à visiter, ainsi que la ville de départ de la tournée et *U*, l'ensemble des arcs de *G*, représentent les parcours possibles entre les villes. A tout arc $(i, j) \in U$, on associe la distance de parcours $d_{i,j}$ de la

ville *i* à la ville *j*. la longueur d'un chemin dans *G* est la somme de distances associées aux arcs de ce chemin. Le PVC se ramène alors à la recherche d'un circuit hamiltonien de longueur minimale dans *G*.

Le PVC peut être modélisé comme suit :

En associant à chaque couple *(i, j)* de villes à visiter *(i=1,..,n ; j=1,...,n* et $i \neq j$) une distance $\delta_{i,j}$ égale à $d_{i,j}$ s'il existe un moyen d'aller directement de *i* à *j* (c'est à dire , *(i, j)* $\in U$, fixé à ∞ sinon et une variable de succession,$x_{i,j}$, binaire, qui prend la valeur 1 si la ville *j* est visitée immédiatement après la ville *i* dans la tournée et qui prend la valeur 0 sinon. Le PVC est alors modélisé par :

$$Min \sum_{i=1}^{n}\sum_{j=1}^{n} \delta_{i,j}\, x_{i,j}$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \quad \forall i = 1, \dots, n \qquad (1)$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \quad \forall j = 1, \dots, n \qquad (2)$$

$$\sum_{i \in S, j \notin S} x_{i,j} \geq 2 \quad \forall\ S \subset X, S \neq \emptyset \qquad (3)$$

$$x_{i,j} \in \{0,1\} \quad \forall i = 1 \dots n, \forall j = 1 \dots n \qquad (4)$$

La contrainte (1)-(2) traduisent le fait que chaque ville doit être visitée exactement une fois, la contrainte (3) interdit les solutions composées des sous-tours disjointes, elle est généralement appelée contrainte d'élimination des sous-tours.

Problème d'ordonnancement

Le problème d'ordonnancement consiste à ordonnancer un ensemble de tâches indépendantes sur *m* machines non reliées, le critère à optimiser étant la durée totale, t, de l'ordonnancement. Dans le contexte non relié, chaque tâche *i* a une durée d'exécution dépendant de la machine *j* sur laquelle elle est effectuée. Ainsi la durée de la tâche *i* sur la machine *j* est p_{ij} . Ce problème est modélisé par le programme linéaire suivant :

$$(PL) \begin{cases} Min\ t \qquad (1) \\ sc \sum_{j=1}^{m} x_{ij} = 1 \ \forall i \in \{1, \dots, n\} \qquad (2) \\ \sum_{i=1}^{n} p_{ij}\, x_{ij} \leq t \ \ \forall j \in \{1, \dots, m\} \qquad (3) \\ x_{ij} \in \{0,1\}\, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\} \qquad (4) \end{cases}$$

La contrainte(2) assure que chacune des *n* tâches est affectée à l'une des *m* machines.la contrainte (3) assure que pour chacune des machines *j* , la somme des durées d'exécution des

tâches affectées à *j* est inférieure à la durée de l'ordonnancement qu'est le critère à minimiser (1).en (4) , $x_{ij} = 1$ si et seulement si la tâche *i* est exécutée par la machine *j* .

Le problème d'affectation

Nous entendons par problème d'affectation le problème qui consiste à associer chaque élément d'un ensemble de *N* objets à un seul élément d'un autre ensemble de (avec $N \geq M$) *M* objets avec un coût minimal. Le problème d'affectation se présente fréquemment en recherche opérationnelle. Il consiste à réaliser une bijection des éléments *i* d'un ensemble *I* sur des éléments, d'un ensemble *J*, de même cardinalité, de telle manière qu'une certaine fonction de coût, dépendant du choix des couples *(i, j)* soit minimale.

Lorsque cette fonction de coût est linéaire, c'est un problème classique et sa solution est donnée par un algorithme polynomial.)

Pourtant, il existe des problèmes appartenant à de nombreux domaines, aussi variés que l'électronique, économie, l'informatique, etc., pour lesquels la fonction de coût est quadratique. Le problème d'affectation s'appelle alors « affectation quadratique ». C'est un problème NP- Complet et donc beaucoup plus difficile à résoudre que l'affectation linéaire.

Considérons les exemples suivants d'affectation :

Problème d'emploi du temps

Dans les établissements scolaires, chaque année l'administration est face à un problème, elle essaye de planifier un certain emploi de temps en respectant certaines contraintes telles que :

- Chaque enseignant doit être affecté à une seule salle pour une certaine durée (exemple : une heure) et le même enseignant ne peut être affecté à deux salles en même temps
- Chaque salle est occupée par un seul enseignant (deux enseignants ne peuvent pas occuper la même salle en même temps)

Le problème à résoudre consiste à concilier toutes ces contraintes pour proposer un emploi du temps sur une certaine durée (un semestre).

Problème d'affectation de fréquences dans les réseaux radio-mobiles :

Le problème d'affectation de fréquences dans les réseaux radio-mobiles est un exemple réel de problème de type problème d'affectation sous contraintes (PASC). L'affectation de fréquences est une extension du problème de coloration. L'objectif de base consiste à affecter des valeurs de fréquences aux stations d'un réseau radio-mobile tout en respectant un ensemble de contraintes pour minimiser les interférences et éventuellement le nombre de

fréquences utilisées. Les contraintes imposent que les fréquences affectées à deux stations adjacentes soient éloignées d'une distance prédéterminée.

En général, on caractérise deux types d'affectation : affectation simple et affectation généralisée.

Affectation simple

Soit un ensemble de *m* opérations qui doivent être exécutées par *n* ressources, $n \geq m$. Chaque couple (opération *i*, ressource *j*) (*i = 1* à *m*, *j = 1* à *n*) a un coût associé c_{ij} , qui représente la dépense associée à la réalisation de l'opération *i* par la ressource *j*. En supposant que chaque opération doit être exécutée une seule fois et que chaque ressource est utilisée au plus par une seule opération, le problème peut être modélisé de la manière suivante :

$$\begin{cases} \text{Minimiser } z=\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\,x_{ij} \quad (1) \\ \text{Sous les contraintes :} \\ \sum_{j=1}^{n} x_{ij} \leq 1, \quad i=\{1,..,m\} \quad (2) \\ \sum_{i=1}^{m} x_{ij} = 1, \quad j=\{1,\dots,n\} \quad (3) \\ x_{ij} \in \{0,1\} \quad \forall i \in \{1,\dots,n\}, \forall j \in \{1,\dots,m\} \quad (4) \end{cases}$$

où x_{ij} est une variable binaire associée à chaque paire (opération *i*, ressource *j*) et qui vaut 1 si l'opération i est effectuée par la ressource j et 0 sinon. Si l'on considère que X_1 représente l'ensemble des opérations ($|X_1| = m$) et X_2 l'ensemble des ressources ($|X_2| = n$) du problème (1)-(4), avec $n \leq m$ alors chaque arête entre un sommet *i* de X_1 et un sommet *j* de X_2 signifie que l'opération i peut être effectuée par la ressource *j*. On associe à chacune de ces arêtes un coût c_{ij} . Comme le coût total d'un couplage est donné par la somme des coûts de chaque arête, le problème d'affectation revient à trouver un couplage de cardinalité *m* et de coût minimum dans un graphe biparti.

Affectation généralisée

Le modèle ci-dessus peut être généralisé lorsque des paramètres peuvent intervenir. Dans ce cas, la contrainte (2) n'est plus suffisante pour décrire l'utilisation des ressources, on l'appellera contrainte de capacité telle qu'une ressource *j* est caractérisée par sa capacité b_i et chaque opération *i* nécessite la quantité a_{ij} si elle utilise la ressource *j* .Ainsi, le modèle devient :

$$\text{Minimiser } z=\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\, x_{ij}$$

Sous les contraintes

$$\sum_{j=1}^{n} a_{ij}\, x_{ij} \leq bi, i = \{1,..,m\}$$

$$\sum_{i=1}^{m} x_{ij} = 1, j = \{1,\dots,n\}$$

$$x_{ij} \in \{0,1\}$$

Avec $a_{ij} \geq 0$ et $b_i \geq 0$

2-5 Formulation des problèmes d'affectation :

Il existe plusieurs formulations du problème d'affectation parmi lesquelles nous en retenons trois comme suit :

2-5-1 Formulation par la programmation linéaire :

$$\text{Min } \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij}$$

Sous contraintes

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,\dots,n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1,..,n$$

$$x_{ij} \geq 0 \quad i,j = 1,\dots,n$$

Une solution réalisable pour le problème d'affectation dans ce cas est une matrice carrée de n^2 éléments, où :

$$x_{ij} = \begin{cases} 1 \; si \; i \; est \; associ\acute{e} \; \grave{a} \; j \\ 0 \qquad\qquad sinon \\ \forall \qquad i \, , j = 1 \dots . n \end{cases}$$

Cette formulation est très intéressante du fait qu'elle montre une particularité bien connue du problème d'affectation.

2-5-2 Formulation comme problème du couplage parfait dans un graphe biparti complet.

Un couplage dans un graphe non orienté est un ensemble d'arêtes qui n'ont pas de sommets en commun. Un couplage est dit parfait si chaque sommet du graphe est une extrémité d'une des arêtes du couplage. Le problème d'affectation peut donc être considéré comme le problème du couplage parfait de coût minimum dans un graphe biparti complet $G = (X_1 \cup X_2, V)$ avec $|X_1| = |X_2| = n$. Chaque arête $(i, j) \in E$ est munie de la valuation C_{ij} . La figure 2-1 représente une instance du problème d'affectation formulée comme un problème de couplage de coût minimum.

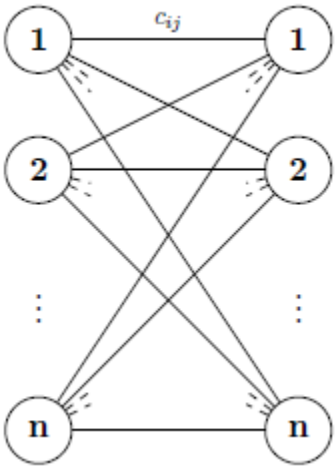


Figure 2-1

Une solution réalisable pour cette formulation peut être représentée par l'ensemble des n arêtes qui forment le couplage.

2-5-3- Formulation en problème de flot maximum à coût minimum

La formulation en problème de flot maximum à coût minimum est un cas général du problème d'affectation. Une grande partie des approches de résolution et des résultats du problème d'affectation sont à l'origine développées pour les problèmes de flot.

Ce problème consiste à faire circuler dans un réseau de la matière, à partir d'un sommet source s vers un sommet puits t, à travers les arcs d'un graphe orienté munis à la fois de capacités q_{ij} et de coûts unitaires c_{ij} . Un flot réalisable respecte les contraintes de capacité sur les arcs et la loi de conservation de la matière, à savoir pour chaque sommet du graphe différent de la source s et du puits t, la somme des flux des arcs entrants est égale à la somme des flux sur les arcs sortants du sommet. Le problème d'affectation peut être formulé comme

un problème de flot maximum à coût minimum comme le montre la figure2-2. Les capacités des arcs sortants de s et ceux entrants en t sont d'une unité et les coûts qui leur sont associés sont nuls. Les autres arcs sont non contraints et sont munis des coûts c_{ij} du problème d'affectation.

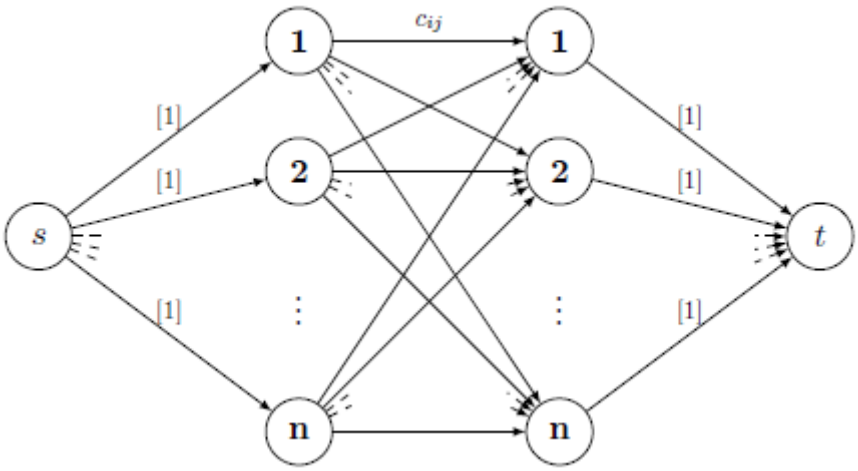


Figure 2-2

Il est évident que les arcs non adjacents à s ou à t et dont le flux vaut 1 correspondent aux arêtes d'une affectation ou d'un couplage.

Introduction

La résolution des problèmes d'optimisation combinatoire 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.

D'une manière très générale, ces dernières suivent deux approches différentes pour la recherche d'une solution : l'approche de construction et l'approche de voisinage.

Ces méthodes font partie de deux groupes de nature différente. Le premier groupe comprend les méthodes exactes d'arborescence qui garantissent la complétude de la résolution : c'est le cas de SEP, A*, hongrois…etc. Le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre (dans le pire des cas). Pour améliorer l'efficacité de la recherche, on utilise des techniques variées pour calculer des bornes permettant d'élaguer le plus tôt possible des branches conduisant à un échec. Le second groupe comprend les méthodes approchées dont le but est de trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue. Les méthodes approchées sont fondées principalement sur diverses heuristiques, souvent spécifiques à un type de problème.

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

3-1 Les méthodes de résolution en optimisation combinatoire :

Elles sont schématisées comme suit :

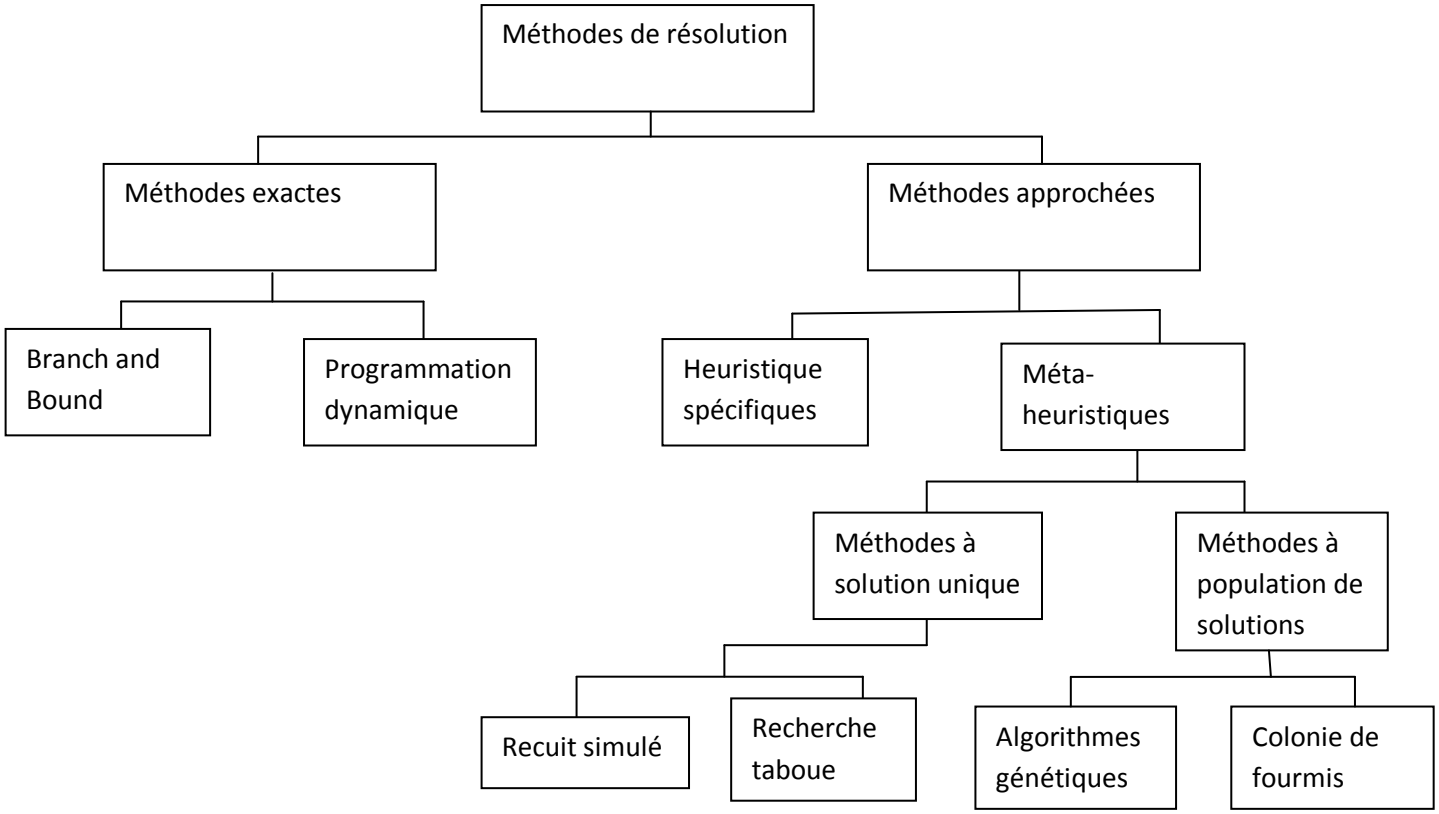


Figure 3-1 : un schéma montrant les différentes méthodes de résolution en optimisation combinatoire

3-1-1 Les méthodes exactes :

Les méthodes exactes telles que la programmation linéaire en nombres entiers, les procédures de recherche arborescente ou la programmation dynamique, garantissent de trouver la solution optimale et de prouver son optimalité pour toutes les instances d'un problème d'optimisation combinatoire (POC). Elles permettent aussi de fournir des informations sur les bornes inférieures / supérieures de la solution optimale d'un problème d'optimisation combinatoire dans le cas où l'algorithme est arrêté avant la fin. Par contre, le temps nécessaire pour trouver la solution optimale d'un problème d'optimisation combinatoire augmente généralement avec la taille du problème. De façon pratique, seuls les problèmes de petite ou moyenne taille peuvent être résolus de façon optimale par des

algorithmes exacts. De plus, pour certains problèmes, la consommation de mémoire de ces algorithmes peut être très grande et peut parfois entraîner l'arrêt prématuré de l'application. La programmation linéaire peut être utilisée pour résoudre de façon exacte les POC. En effet, lorsqu'un POC peut être modélisé en programme linéaire, il peut alors être résolu, par exemple à l'aide de l'algorithme de simplexe. Lorsqu'on a la contrainte supplémentaire voulant que les valeurs des variables de décision soient entières, on a un programme linéaire en nombre entier (PLNE).

On peut résoudre de façon exacte certaines instances de PLNE à l'aide de l'algorithme de B&B. Le B&B utilise le branchement pour deviser l'ensemble des solutions en sous ensembles et l'évaluation pour mettre des bornes supérieures ou inférieures sur les solutions. La première étape de l'algorithme consiste à résoudre la relaxation du PLNE en éliminant la contrainte d'intégralité. Si la solution du programme relaxé est entière, alors elle est aussi une solution optimale du programme non relaxé. Sinon, une opération de branchement est exécutée en choisissant une variable de décision *x* non entière dans la solution optimale précédemment obtenue. On sépare en deux l'ensemble de solutions en ayant, d'un côté, les solutions dont *x* prend une valeur supérieure à celle dont la solution optimale précédente, et de l'autre côté, les solutions où sa valeur est inférieure. Les nouveaux nœuds sont alors évalués et on coupe les branches qui sont jugées non intéressantes .Les branches sont coupées lorsque le programme linéaire n'est pas réalisable, lorsque la solution est de moindre qualité que la meilleure solution trouvée. Le B&B s'arrête quand il n'y a plus de nœuds à évaluer. La meilleure solution entière trouvée est la solution optimale du problème .Il est à noter que le choix du prochain nœud à évaluer ainsi que celui de la variable de branchement influence grandement la performance du B&B.

3-1-1-1 Bip Match

Le Bip Match est un algorithme rapide utilisant les chaînes alternées améliorantes pour trouver un couplage maximum dans un graphe biparti. Pour un graphe G et un couplage C, une chaîne est alternée si elle emprunte alternativement des arêtes dans C et hors C. Pour un couplage, une chaîne alternée est améliorante (ou augmentante) si ses deux extrémités sont insaturées par le couplage.

Le fonctionnement de l'algorithme est basé sur les deux propriétés suivantes dues à Berge

Propriété 1 : Soient G un graphe, C un couplage de G et une chaîne alternée améliorante μ pour C. Un transfert sur μ donne un nouveau couplage C0 avec |C0 | = |C| + 1.

Propriété 2 : Un couplage est maximal si et seulement s'il n'admet aucune chaîne alternée améliorante.

Le **Bip Match** part d'un couplage initial non vide, obtenu en cherchant les chaînes alternées améliorantes réduites à une arête - on affecte chaque sommet de *X* à son premier successeur libre, c'est-à-dire n'étant pas déjà extrémité d'une arête du couplage.

A partir de cette affectation initiale, on cherche les chaînes améliorantes, par une exploration en largeur du graphe. Cette recherche vise à améliorer la solution trouvée par l'affectation initiale, puis la solution courante, par l'augmentation de la cardinalité du couplage. On part toujours d'un sommet insaturé de *X* et on s'arrête à la plus courte chaîne trouvée. On va explorer les sommets alternativement (un sommet de *X*, un sommet de *Y*), de manière que la chaîne ait une arête hors du couplage, puis une arête dans le couplage. L'algorithme s'arrête quand il n'arrive plus à trouver des chaînes améliorantes.

3-1-1-2 Algorithme hongrois

Un cas particulier du problème d'affectation, souvent traité dans la littérature est celui où le nombre des opérations à effectuer est égal au nombre des ressources utilisées.

En effet, tout problème d'affectation simple peut être mis sous cette forme par introduction d'opérations fictives à coût nul pour toutes les ressources dans le cas $m < n$, ou de ressources fictives à coût nul pour toutes les opérations dans le cas $m > n$. Une méthode de résolution du problème d'affectation qui prend en compte cette structure est l'algorithme hongrois.

L'algorithme hongrois est un algorithme qui utilise la modélisation sous forme de programme linéaire du problème d'affectation. Une schématisation simpliste de l'algorithme hongrois pourrait être la suivante :

Etape 0 On crée la matrice des coûts (c_{ij}) $i= 1,..., m, j = 1,..., n, (m = n)$ initiale, qui est considérée non négative.

Etape 1 Pour chaque ligne de la matrice, on trouve l'élément minimal et on le soustrait à tous les éléments de cette ligne. S'il reste des colonnes sans coefficients nuls, pour chacune de ces colonnes, on trouve l'élément minimal et on le soustrait à tous les éléments de cette colonne. On obtient alors un problème équivalent avec une matrice ayant un zéro par ligne et par colonne.

Etape 2 Sélectionner le maximum de zéros possible de façon à ce qu'il n'y ait qu'un zéro sélectionné par ligne et par colonne. Si l'on a sélectionné *n* zéros alors on a trouvé l'affectation optimale, on arrête l'algorithme sinon on continue. (Aller à l'étape 3)

Etape 3 Marquez chaque ligne n'ayant pas de zéro sélectionné. Marquez chaque colonne ayant un zéro sur une ligne marquée. Marquez chaque ligne ayant un zéro marqué dans une

colonne marquée. Répéter cette opération jusqu'à un état stable; sélectionnez alors la sous-matrice formée par les lignes marquées et par les colonnes non marquées; Cette étape permet de sélectionner la plus grande sous-matrice n'ayant aucun zéro.

Etape 4 Trouvez la valeur minimum de la sous-matrice trouvée à l'étape 3. Il faut alors soustraire cette valeur à toutes les lignes marquées, et l'ajouter à toutes les colonnes marquées. Retournez à l'étape 2.

3-1-1-3 Branch and bound (séparation et évaluation) :

La méthode Branch and Bound est une méthode de résolution de classe de problème d'optimisation globale en utilisant la stratégie de diviser un ensemble *X* donné en plusieurs sous ensembles de plus en plus petit et ceci en se basant sur deux concepts, le branchement qui consiste à partitionner ou diviser l'espace des solutions en sous problèmes, pour les optimiser chacun individuellement, et l'évaluation qui consiste à déterminer l'optimum global.

Les étapes de Branch and Bound

Séparation

Les méthodes de Branch and Bound reposent sur la réduction progressive de l'ensemble des solutions optimales. Du fait de la difficulté des problèmes d'optimisation combinatoire, il est souvent plus facile d'aborder leur résolution en décomposant de manière itérative l'ensemble admissible en plusieurs sous-ensembles et ainsi définir des sous-problèmes. À chaque itération d'un algorithme de Branch and Bound, un sous-ensemble est choisi dans l'ensemble des parties de l'ensemble réalisable. Généralement des contraintes sont ajoutées à la description de l'ensemble séparé. Par exemple, l'ensemble admissible d'un problème avec des variables binaires est divisé en imposant qu'une variable vaille 0 pour les solutions appartenant à un sous-ensemble et que la même variable soit égale à 1 pour les solutions d'un second sous-ensemble.

Evaluation

L'évaluation permet de calculer une borne inférieure (dans le cas de la minimisation) sur la valeur de la solution optimale des sous-problèmes issus de la séparation et d'éliminer les nœuds qui ne peuvent contenir la solution optimale du problème original.

Les méthodes de résolution exacte sont limitées à des problèmes de taille relativement petite, c'est la raison pour laquelle de nombreuses méthodes approchées ont été adaptées à la résolution de ces problèmes.

3-1-2 Les méthodes approchées

Contrairement aux méthodes exactes, les méthodes approchées ne procurent pas forcément une solution optimale, mais seulement une bonne solution (de qualité raisonnable) en un temps de calcul aussi faible que possible.

Une partie importante des méthodes approchées est désignée sous le terme de Méta-heuristiques.

Plusieurs classifications des méta-heuristiques ont été proposées ; la plupart distinguent globalement deux catégories : les méthodes à base de solution courante unique, qui travaille sur un seul point de l'espace de recherche à un instant donné, appelées méthodes à base de voisinage comme les méthodes de recherche locale (méthode de la descente), de recuit simulé et de recherche taboue, et les méthodes à base de population, qui travaillent sur un ensemble de points de l'espace de recherche, comme les algorithmes évolutionnaires et les algorithmes de colonies de fourmis.

Heuristiques

En optimisation combinatoire, une heuristique est un algorithme approché qui permet d'identifier en temps polynomial, au moins, une solution réalisable rapide, pas nécessairement 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 garantie 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.

Méta-heuristiques

Face aux difficultés rencontrées par les heuristiques pour avoir une solution réalisable de bonne qualité pour des problèmes d'optimisation difficiles, les méta-heuristiques ont fait leur apparition. Ces algorithmes permettent généralement d'obtenir une solution de très bonne qualité pour des problèmes issus des domaines de la recherche opérationnelle 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 méta-heuristique reste alors, dans la majorité des cas, très intéressant par rapport aux différents types d'approches de résolution.

Une méta-heuristique peut être adaptée pour différents types de problèmes, tandis qu'une heuristique est utilisée à un problème donné (i.e. : une méta-heuristique combine plusieurs approches heuristiques).

Les méthodes approchées utilisent, durant sa recherche de l'optimum, un certain processus appelé le voisinage pour cela on donne quelques définitions le concernant :

Méthodes de voisinage

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

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

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

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

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

A chaque arc <*si*, *sj*> ∈ *E* du graphe, on peut également associer une valeur $C_{ij} = f(sj) - f(si)$ qui correspond à la variation de coût entre *si* et *sj*.

Avec cette notion de graphe, une méthode de voisinage peut être vue comme un processus qui parcourt un chemin du graphe. A chaque nœud, le mécanisme de parcours choisit l'arc à parcourir essentiellement en fonction des valeurs des arcs partant du nœud courant.

L'efficacité de la méthode dépend donc de deux choses : la structure du graphe déterminée par le voisinage et la façon de le parcourir. La recherche locale est un exemple simple de cette classe de méthodes.

3-1-2-1 Recherche locale

La recherche locale, appelée aussi la descente ou l'amélioration itérative, représente une classe de méthodes heuristiques très anciennes.

Traditionnellement, la recherche locale constitue une arme redoutable pour attaquer des problèmes réputés très difficiles tels que le problème de voyageur de commerce et la partition de graphes. Contrairement à l'approche de construction, la recherche locale manipule des configurations complètes durant la recherche.

Une méthode de recherche locale est un processus itératif fondé sur deux éléments essentiels : un voisinage $N: X \rightarrow 2^X$ et une procédure exploitant le voisinage. Plus précisément, elle consiste à débuter avec une configuration quelconque *s* de *X*, et choisir un voisin *s'* de *s* tel que $f(s') < f(s)$ et remplacer *s* par *s'* et à répéter jusqu'à ce que pour tout voisin *s'* de *s*, $f(s') \geq f(s)$. Cette procédure fait intervenir à chaque itération le choix d'un voisin qui améliore la configuration courante. Plusieurs possibilités peuvent être envisagées pour effectuer ce choix. Il est possible d'énumérer les voisins jusqu'à ce qu'on en découvre un qui améliore strictement (première amélioration). Comme l'espace des solutions *X* est fini, cette procédure de descente s'arrête toujours, et la dernière configuration trouvée ne possède pas de voisin strictement meilleur qu'elle-même. Autrement dit, la recherche locale retourne toujours un optimum local.

L'avantage principal de cette méthode réside dans sa grande simplicité et sa rapidité. Mais les solutions produites sont souvent de qualité médiocre et de coût très supérieur au coût optimal. Pour remédier à ce problème, une solution consiste à accepter des voisins de même performance que la configuration courante. Cette approche permet à la recherche de se déplacer sur les plateaux, mais n'est pas suffisante pour ressortir de tous les optima locaux.

Notons enfin qu'on trouve également l'idée de recherche locale dans le célèbre algorithme du simplexe pour la programmation linéaire.

L'inconvénient majeur de la méthode de descente est son arrêt au premier minimum local rencontré. Pour améliorer les résultats, on peut lancer plusieurs fois l'algorithme en partant d'un jeu de solutions initiales différentes.

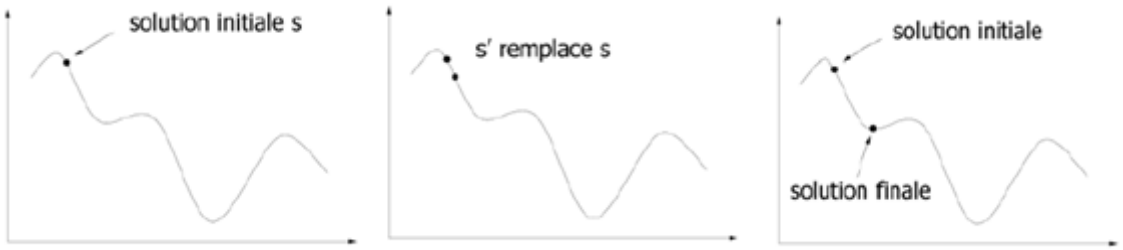


Figure3-2 : Evolution d'une solution dans la méthode de recherche locale.

Algorithme de la recherche locale :

Pour un problème de minimisation d'une fonction f , la méthode descente peut être décrite comme suit :

Solution initiale s ;

Répéter

Choisir s'dans un voisinage $N(s)$ de s ;

Si $f(s') < f(s)$ alors $s := s'$;

jusqu'à ce que $f(s') \geq f(s), \forall s' \in N(s)$.

Caractéristiques des heuristiques :

Les heuristiques présentent de nombreux avantages mais aussi un certain nombre de limitations. Nous résumons ci-dessous les différentes caractéristiques de ces méthodes :

- généralité et application possible à une large classe de problèmes,
- efficacité pour de nombreux problèmes,
- possibilité de compromis entre qualité des solutions et temps de calcul,
- optimum non garanti,
- nécessité d'adaptation de la méthode au problème traité,
- difficulté de prévoir la performance.

Classification des algorithmes heuristiques :

Les algorithmes heuristiques appartiennent essentiellement à trois approches de résolution :

- Construction séquentielle : méthodes très rapides mais pas particulièrement efficaces,
- Recherches locales,

Recherche Tabou,

Recuit simulé,

Recherche à voisinage variable ou espace de recherche variable,

- Méthodes à base de populations ou distribuées

Algorithmes hybrides évolutionnistes,

Algorithme de colonies de fourmis,

Algorithme hybride distribué.

3-1-2-2 Recuit simulé :

La méthode du recuit simulé a été introduite en 1983. Cette méthode tire son origine de la physique. Son principe de fonctionnement repose sur une imitation du phénomène de recuit en science des matériaux. Ce processus utilisé en métallurgie pour améliorer la qualité d'un solide, cherche un état d'énergie minimale qui correspond à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire régulière parfaite. En partant d'une haute température où le métal serait liquide, on refroidit le métal progressivement en tentant de trouver le meilleur équilibre thermodynamique. Chaque niveau de température est maintenu jusqu'à obtention d'un équilibre.

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

A chaque étape, l'acceptation ou non d'un nouvel état dont l'énergie est supérieure à celle de l'état courant est déterminée de manière probabiliste : un réel $0 \leq \theta \leq 1$ est tiré aléatoirement et ensuite comparé avec P(ΔE,T). Si P(ΔE,T)$\geq \theta$ alors le nouvel état est accepté pour remplacer l'état courant, sinon, l'état courant est maintenu.

Après un grand nombre de perturbations, un tel processus fait évoluer le système vers un état d'équilibre thermodynamique selon la distribution de Boltzmann qui est définie par la

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

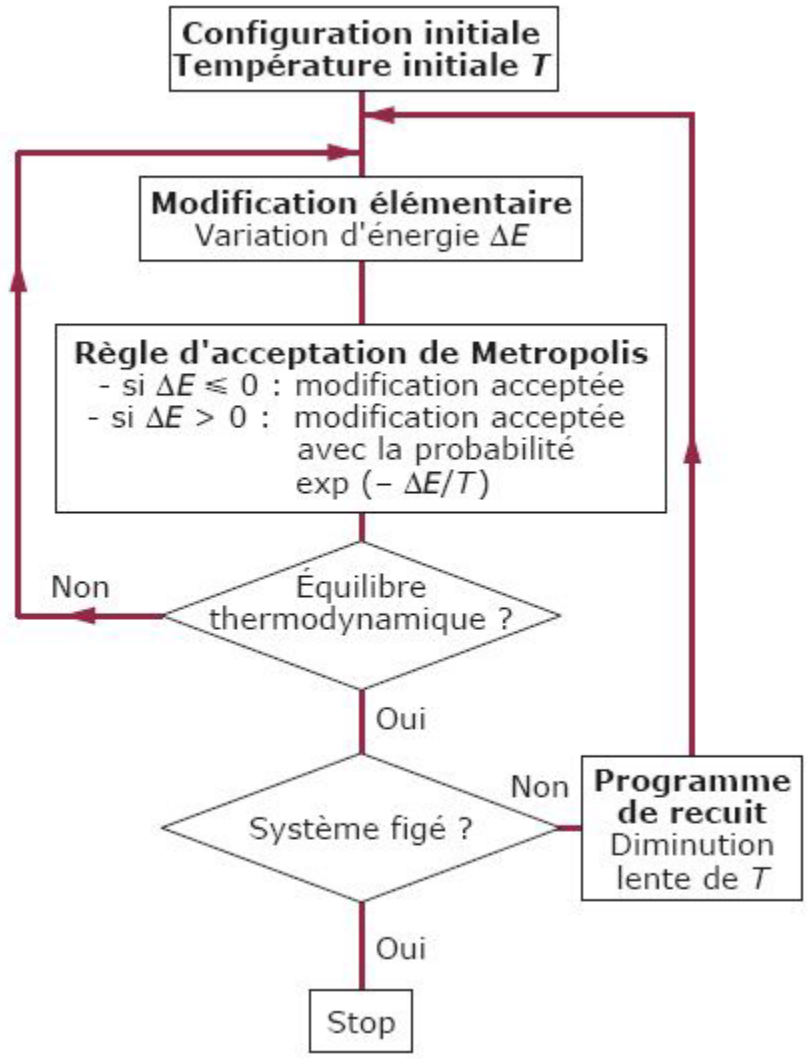


Figure 3-3: Fonctionnement de l'algorithme de recuit simulé

3-1-2-3 Recherche taboue :

La recherche taboue (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. 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 œuvre une liste tabou (tabu list) T de longueur k contenant les k dernières solutions visitées, ce qui ne donne pas la possibilité à 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 taboue.

Définition de base de recherche taboue

Définition :

Développée dans un cadre particulier par Glover en 1986, c'est une méthode heuristique de recherche locale utilisée pour résoudre des problèmes complexes et de très grande taille (souvent NP-difficile).

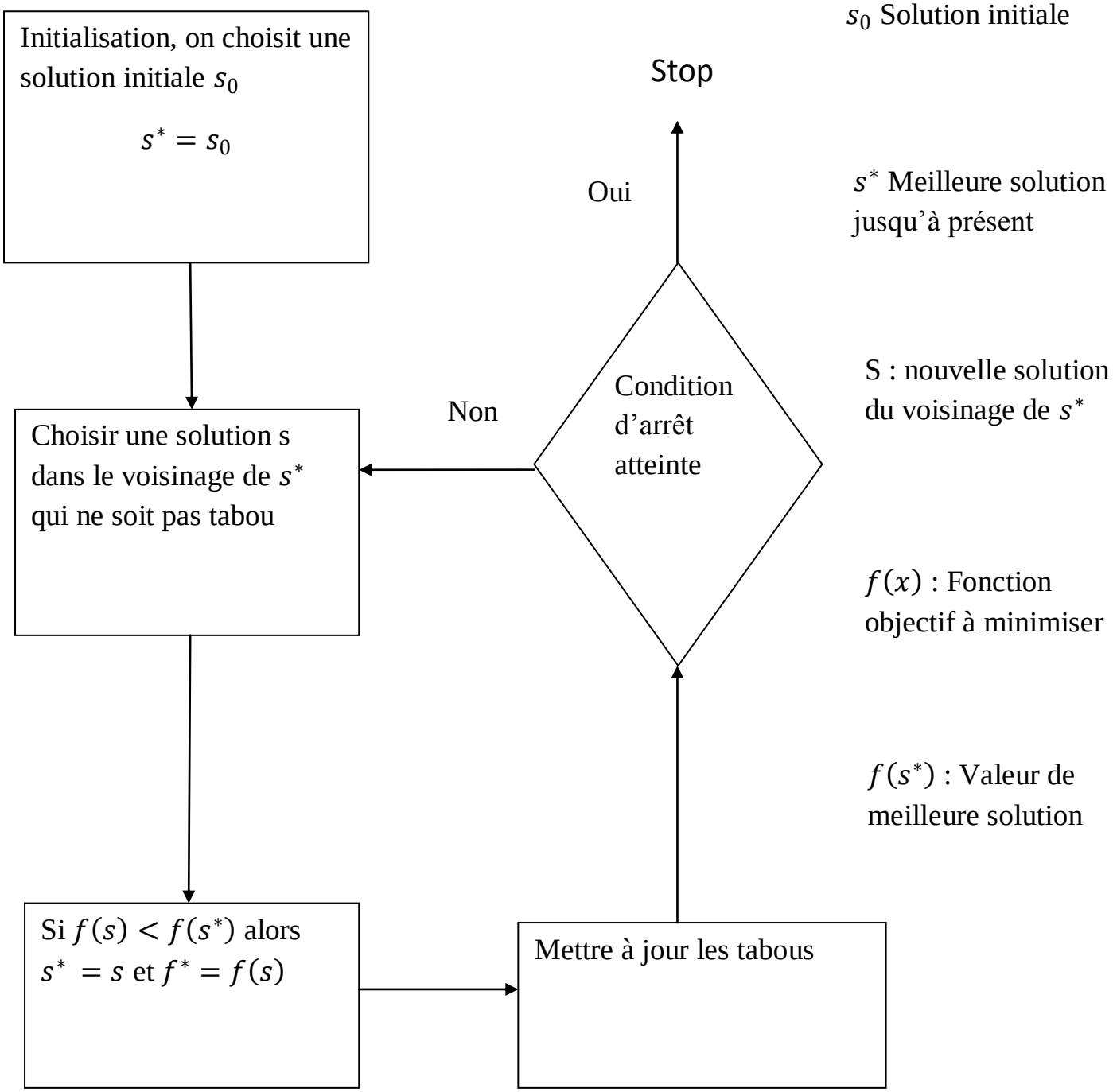
Principe de base

Poursuivre la recherche de solutions même lorsqu'un optimum locale est rencontré et ce :

- En permettant des déplacements qui n'améliorent pas la solution ;
- En utilisant le principe de mémoire pour éviter les retour en arrière (mouvement cyclique).

Mémoire :

Elle est représentée par une liste taboue qui contient des mouvements ou des solutions qui sont temporairement interdits.



Organigramme de la recherche taboue en générale

Figure3-4

Critère d'arrêt :

- On peut arrêter la recherche à tout moment.
- Des critères d'arrêt possibles sont :

  -Si une solution prouvée optimale a été trouvée ;

  -Si une limite a été atteinte en ce qui concerne :

  - Le nombre d'itérations,
  - Le temps de calcul.

Comportement de l'algorithme tabou

- **Si la liste taboue est courte :**
  - Il y a moins d'interdictions (mouvements tabous),
  - L'algorithme tend à parcourir de moins grandes distances dans l'espace de recherche .Il explore moins l'espace de recherche,
  - Le risque de cycles est plus grand.

- **Si la liste taboue est longue :**
  - Il y a avantage d'interdictions,
  - La recherche risque de manquer de nombreux optima locaux sur son chemin,
  - L'algorithme tend à parcourir de plus grandes distances dans l'espace de recherche,
  - Il explore d'avantage l'espace de recherche,
  - Le risque de cycles est réduit.
- **Le comportement de l'algorithme dépend :**
  - De la longueur de la liste taboue.
  - La taille de la liste des configurations.

Remarque

Plus la liste de configurations est petite, moins la liste taboue a besoin d'être grande.

Diverses améliorations de la recherche taboue

La liste peut s'avérer trop contraignante lors de la recherche d'une solution. Le mécanisme d'aspiration permet de lever ponctuellement le statut « tabou » afin d'atteindre des solutions inédites.

L'intensification : est l'une de stratégies qui permet de mémoriser les meilleures solutions rencontrées (ou leurs configurations) et les utilise afin d'améliorer la recherche.

La diversification : cherche à utiliser des mouvements encore jamais réalisés afin d'explorer des régions nouvelles de l'espace de recherche en mémorisant bien sûr les solutions visitées.

Avantages et inconvénients

La recherche taboue 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 taboue exige une gestion de la mémoire de plus en plus lourde en mettant des stratégies de mémorisation. L'efficacité de la méthode taboue offre son utilisation dans plusieurs problèmes d'optimisation combinatoire classiques tels que le problème de voyageur de commerce, le problème d'ordonnancement…etc.

Domaines concernés par la recherche taboue

Problème de transport.

Planification et ordonnancement.

Optimisation des graphes.

Télécommunication…etc.

Algorithme tabou

1-Initialisation

s_0 Une solution initiale

$s \leftarrow s_0, s^* \leftarrow s_0\ , c^* \leftarrow f(s_0)$

$T = \emptyset$

2-Générer un sous-ensemble de solutions au voisinage de s

$s^{'} \in N(s) tel\ que\ \forall\ x \in N(s), f(x) \geq f(s^{'}) et\ s^{'} \notin T$

Si $f(s^{'}) < c^*$ alors $s^* \leftarrow s^{'}\ et\ c^* \leftarrow f(s^{'})$

Mise à jour de T

Jusqu'à la vérification de la condition d'arrêt.

Introduction :

Dans ce chapitre nous appliquons deux méthodes différentes : la méthode de descente et la recherche taboue pour une instance de problème de voyageur de commerce.

4-1 Présentation du problème :

Un voyageur de commerce souhaite visiter un certain nombre de villes, il doit visiter chaque ville une et une seule fois. En partant d'une ville initiale et y revenir.

Etant donné des distances entre chaque paire de villes, il doit minimiser la distance totale parcourue. On peut représenter ce problème par un graphe :

Chaque ville correspond à un sommet et chaque arête à une paire de villes pouvant être visitées l'une à la suite de l'autre.

Le problème correspond à trouver un tour complet (circuit Hamiltonien) dans ce graphe qui minimise la somme des distances (l'objectif).

Dans notre exemple, le nombre de ville à visiter est fixé à 7.

On applique deux heuristiques différentes la méthode descente et la recherche taboue utilisant le principe de voisinage.

On schématise notre problème par un graphe $G=(X, E)$ constitué comme suit :

$X = \{1, 2, 3, 4, 5, 6, 7\}$ sont les villes à visiter, par le voyageur de commerce.

E= la liaison des villes pouvant être visitées l'une après l'autre et une évaluation des arêtes représentant la distance entre ces villes.

Comme le montre la figure 4-1 :

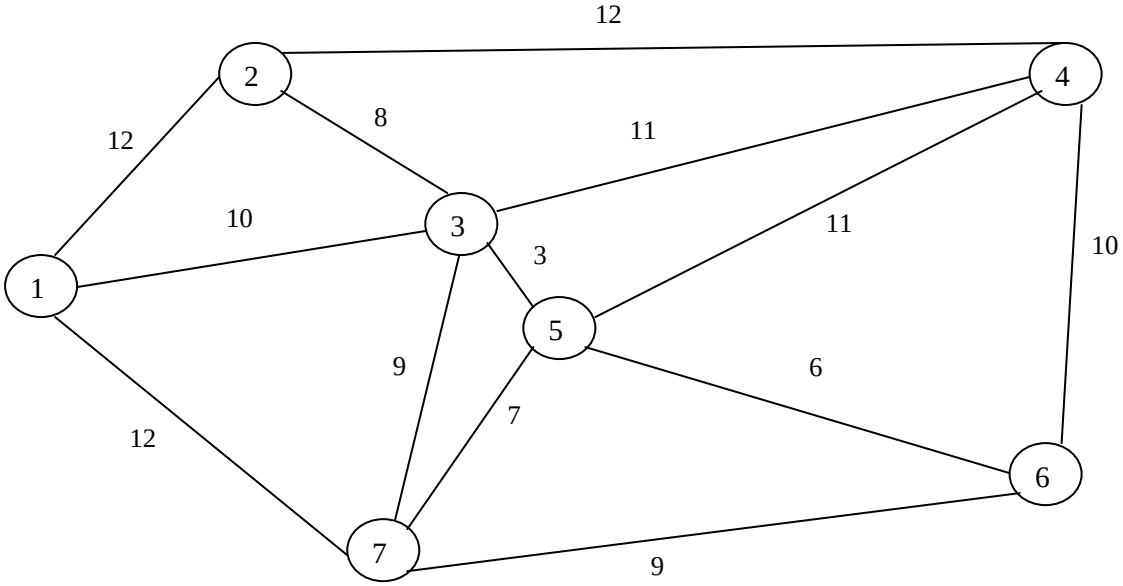


Figure : 4-1

Le principe utilisé pour la résolution du problème :

Inversion de sous-tours

Pour le problème du voyageur de commerce, on peut définir plusieurs types de modification Un type de modification possible consiste à inverser l'ordre de visite d'une sous-séquence de villes.

Exemple :

si la séquence de visite des villes est 1-2-3-4-5-6-7-1 et que nous choisissions d'inverser la sous-séquence 3-4, la solution 1-2-4-3-5-6-7-1 serait obtenue suite à cette modification.

Sur notre exemple, on peut vérifier que la distance totale passe de 69 à 65 suite à cette modification.

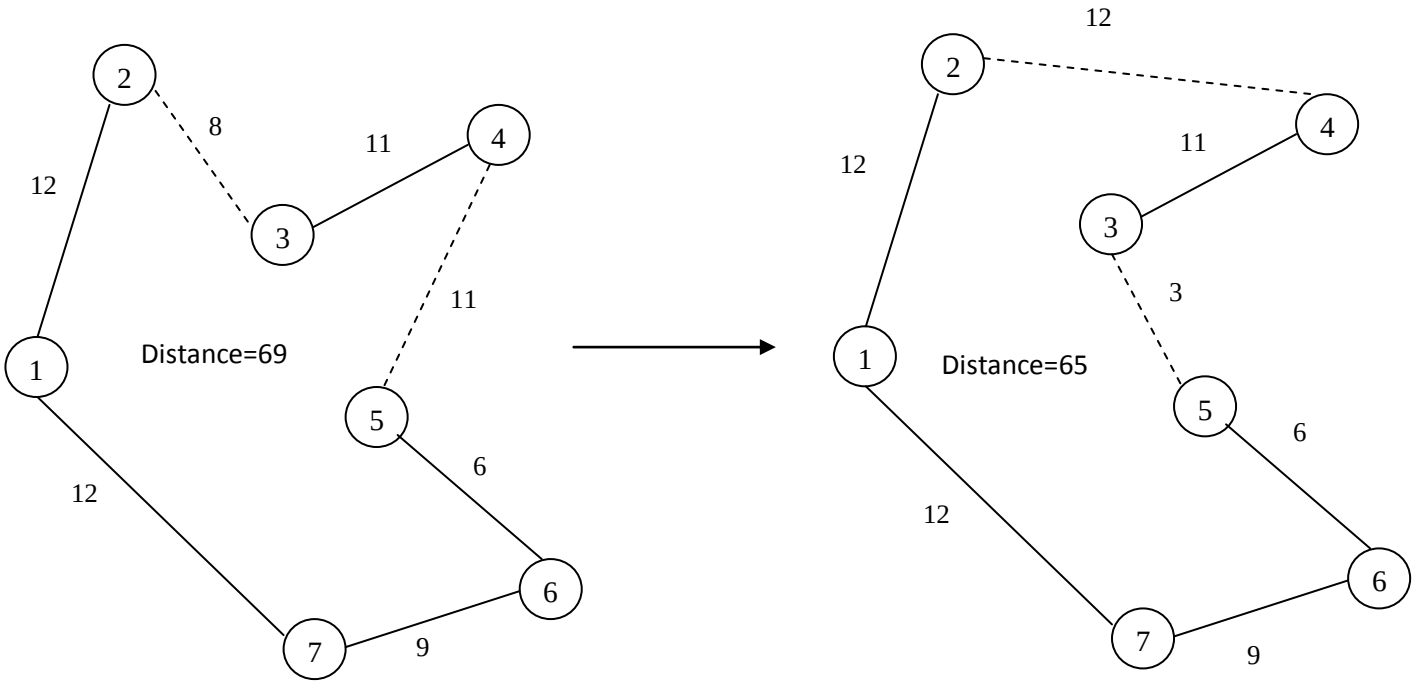


Figure 4-2

4-2 Application de la méthode de descente et la recherche taboue

4-2-1 La méthode de descente

Par inversion de sous-tours :

- Identifier une solution réalisable initiale.
- Considérer toutes les inversions de sous-tours et choisir celle qui améliore le plus la distance totale parcourue par le nouveau tour ainsi obtenu.
- Arrêter s'il n'y a aucune inversion de sous-tours qui permette d'améliorer la distance totale parcourue

Application

On choisit aléatoirement une solution initiale et on essaye de l'améliorer par inversion de sous-tours.

- Solution initiale : 1-2-3-4-5-6-7-1 (la distance totale est 69)
- Inversions de sous-tours (les autres possibilités ne mènent pas à une solution réalisable) :
  - 1-3-2-4-5-6-7-1 : 68
  - 1-2-4-3-5-6-7-1 : 65
  - 1-2-3-5-4-6-7-1 : 65
  - 1-2-3-4-6-5-7-1 : 66

Il y a deux modifications qui diminuent le plus la distance : on choisit la première.

- Solution courante : 1-2-4-3-5-6-7-1 (65)
- Inversions de sous-tours :
  - 1-2-3-4-5-6-7-1 : 69 (solution précédente)
  - 1-2-4-6-5-3-7-1 : 64

Il n'y a qu'une seule modification qui diminue la distance totale : on la choisit.

- Inversions à partir de 1-2-4-6-5-3-7-1 :
  - 1-2-4-3-5-6-7-1 : 65 (solution précédente)
  - 1-2-4-6-5-7-3-1 : 66

Aucune modification n'améliore la valeur de l'objectif, on arrête.

La meilleure solution obtenue par descente 1-2-4-6-5-3-7-1 d'une distance 64.

Plusieurs méta-heuristiques utilisent la même approche (voisinage) que la descente, mais tentent d'éviter de demeurer piégées dans un minimum local (la recherche avec taboue).

4-2-2 La recherche taboue :

L'idée de cette méthode est de permettre des modifications qui n'améliorent pas la valeur de l'objectif. Toutefois, on choisira toujours la meilleure modification possible mais nous venons de voir qu'une des modifications possibles nous ramène à la solution précédente. Il faut donc changer la définition de l'ensemble des modifications possibles pour interdire celles qui nous ramènent à la solution précédente.

A cette fin, on conservera une liste des dernières modifications effectuées en rendant taboue (en interdisant) la modification inverse. Cette liste taboue peut être vue comme une mémoire à court terme permettant de guider la recherche. A chaque itération, on choisit la meilleure modification possible (excluant celles qui sont taboues), puis on met à jour cette liste en ajoutant la modification inverse de celle effectuée.

Contrairement à la méthode de descente, il n'y a pas de critère d'arrêt simple.

Typiquement, on utilise une combinaison des critères suivants :

- Nombre maximum d'itérations ;
- Temps limité ;
- Nombre d'itérations successives sans amélioration ;
- Il n'y a plus de modification possible.

Adaptation de cette méta-heuristique pour résoudre un problème particulier : structure de voisinage + implantation de la liste taboue.

Voisinage : inversion de sous-tours, ce qui implique l'ajout de deux liens et l'élimination de deux liens.

Liste taboue : les deux liens ajoutés sont insérés dans la liste taboue; une modification est taboue si les deux liens à éliminer sont dans la liste taboue.

La taille de la liste taboue

On ne conservera dans la liste taboue que les liens ajoutés lors des deux dernières itérations : on dit que la longueur de la liste taboue est 4.

Test d'arrêt : On arrête l'algorithme après 4 itérations.

Tabou effectue les mêmes modifications que la descente, mais gère également la liste taboue

Reprenons notre exemple :

Solution initiale : 1-2-3-4-5-6-7-1 (69).

Itération 1 :

- Inversion de la sous-séquence 3-4

  → ajout des liens 2-4 et 3-5.

  → Liste taboue = 2-4, 3-5.

- Nouvelle solution : 1-2-4-3-5-6-7-1 (65)

Itération 2 :

- Inversion de la sous-séquence 3-5-6

  →ajout des liens 4-6 et 3-7.

  → Liste taboue = 2-4, 3-5, 4-6, 3-7.

- Nouvelle solution : 1-2-4-6-5-3-7-1 (64).

Itération 3

A partir de la solution courante 1-2-4-6-5-3-7-1, il y a deux modifications :

- Inversion de la sous-séquence 6-5-3

  →élimination des liens 4-6 et 3-7, mais les deux sont dans la liste taboue : modification taboue

- Inversion de la sous-séquence 3-7

  → ajout des liens 5-7 et 3-1, élimination des liens 5-3 et 7 -1

  → Liste taboue = 4-6,3-7, 5-7, 3-1

Nouvelle solution : 1-2-4-6-5-7-3-1 (66 > 64)

Itération 4

A partir de la solution courante 1-2-4-6-5-7-3-1

- Inversion de la séquence 5-7

  →ajout des liens 6-7 et 5-3, élimination des liens 6-5 et 7-3.

  →Liste tabou : 5-7,3-1,6-7,5-3.

- Nouvelle solution 1-2-4-6-7-5-3-1 (63)

Condition d'arrêt atteinte (4 itérations)

La meilleure solution obtenue par taboue est 1-2-4-6-7-5-3-1 minimisant la distance parcourue jusqu'à 63.

Conclusion :

En comparant les deux méthodes, on constate que la méthode de descente est intéressante, mais elle ne garantit pas de trouver la meilleur solution locale. Quant à taboue, elle est plus performante.

En générale, la recherche Taboue (RT) nous donne des résultats meilleurs que la méthode de descente.

Dans notre travail, nous avons étudié quelques méthodes et algorithmes de résolution de certains problèmes d'optimisation combinatoire, ces méthodes sont généralement classées en deux grandes catégories : les méthodes exactes et les méthodes approchées.

Si les méthodes de résolution exactes permettent d'obtenir une solution 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 méta-heuristiques, 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 engendrée 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.

En fin nous avons appliqué deux des méthodes présentées sur un exemple du problème du voyageur de commerce.

Bibliographie

[1] C. BERGE: graphe et hypergraphes, DUNOD, paris,1958.

[2] M. MINOUX: programmation mathématique, théorie et algorithme, DUNOD 1983.

[3] Jin-Kao Hao, Philippe Galinier, Michel Habib.Méthaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. Revue d'Intelligence ArtificielleVol : No. 1999

[4] M. SAKAROVITCH: graphes et programmation linéaire, HERMANN 1984.

[5] M. SAKAROVITCH: programmation discrete, HERMANN 1984.

Résumé :

L'optimisation peut être définie comme une branche mathématique orientée vers la recherche de la meilleure façon d'opérer des choix en vue d'aboutir au résultat visé ou au meilleur résultat possible. Elle fait partie de la science d'aides à la décision dans la mesure où elle propose des modèles conceptuels en vue d'analyser et de maîtriser des situations complexes pour permettre aux décideurs de comprendre et d'évaluer les enjeux et d'attribuer ou de faire les choix les plus efficaces.

La résolution des problèmes combinatoires est assez délicate puisque le nombre fini et ou dénombrable et ou infini 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). Ces approches de résolution peuvent être classées en deux catégories : les méthodes exactes et les méthodes approchées.

Les méthodes exactes gagnent en l'optimalité des solutions et perdent en temps d'exécution, ce qui est l'inverse pour les méthodes approchées qui ne garantissent pas de trouver une solution exacte, mais seulement une approximation en des temps raisonnables de calculs.

Mots clés : Optimisation combinatoire, heuristique, méta-heuristique, recherche taboue, méthode de descente.