REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDELHAMID IBN BADIS DE MOSTAGANEM

FACULTE DES SCIENCES EXACTES ET DES SCIENCES
DEPARTEMENT DE MATHEMATIQUES

MÉMOIRE

Pour obtenir le diplôme de
Magister **en Mathématiques**

Option : Analyse des Systèmes-Contrôle
et Optimisation Numériques

Présenté

BELGACEM RACHID

Sur Quelques Méthodes de Résolution pour le TSP

"*Travelling Salesman Problem* "

Devant les membres du jury :

S.M. BAHRI	**Maitre de conférences A**	**Président**	**UMAB**
D. BOUAGADA	**Maitre de Conférences A**	**Examinateur**	**UMAB**
M. ELOSMANI	**Maitre de Conférences A**	**Examinateur**	**ENSET** d'Oran
A. AMIR	**Maître de conférences A**	**Encadreur**	**UMAB**

Table des matières

Remerciements

Je remercie tout d'abord notre Dieu qui ma donné de la volonté et surtout de la patience pour réaliser ce travail.

Je tiens surtout à exprimer mes sincères remerciements à mon encadreur Monsieur A. Amir Maître de conférences à l'université de Mostaganem, et à lui exprimer toute ma reconnaissance et ma gratitude pour son assistance, ses conseils et sa compréhension, qui ont contribué à la réalisation de ce travail.

Je remercie vivement Monsieur S.M. Bahri Maître de conférence à l'université de Mostaganem, pour m'avoir fait l'honneur de présider le jury de ce mémoire. Je tiens à remercier tout particulièrement Monsieur Djillali Bouagada Maître de conférence à l'université de Mostaganem et Monsieur M.Elosmani Maître de conférence à ENSET Oran, pour l'honneur qu'ils m'ont fait en acceptant d'examiner ce travail.

Mes remerciements s'adressent aussi à mes Professeurs de post-graduation O.Belhamiti et D.Zoubir ainsi que mes collègues ELOSMANI Aissa, MARIR Saliha et ADJEL Nora .

Merci à mes chers et adorés parents pour toute présence effective.

Je ne saurais oublier dans mes remerciements tout mes enseignants, et mes amis ainsi que tous les membres de ma très chère famille

Au terme de ce travail, je voudrais adresser une pensée à tous ceux qui, de quelque manière que ce soit, par un conseil, une idée, un coup de main ou, tout simplement leur amitié, m'ont aidée à le réaliser.

Je vous souhaite à tous une agréable lecture !

Dédicace

Je dédie ce modeste travail à mon épouse
ainsi qu'à ma chère fille Hind.
Et à tous ceux qui ont participé de près
ou de loin a l'élaboration de ce travail.

Notations

$\mathbb{Z}$	:	L'ensemble des entiers relatives
$\mathbb{R}$	:	L'ensemble des nombres réels
$\mathbb{R}^+$	:	L'ensemble des nombres réels non-négatifs
$\mathbb{R}^m$	:	L'espace des vecteurs à m composantes réelles
$\mathbb{R}^m_*$	:	L'espace des vecteurs non nuls à m composantes réelles
$\mathbb{R}^{m\times n}$	:	L'espace des matrices réelles de dimension $m \times n$
A^T	:	Transposée de matrice
A^{-1}	:	Inverse de matrice
B^{adj}	:	L'adjoint de B
I_k		Matrice identité d'ordre k
0_n		Matrice nulle d'ordre n
$S_{(P)}$	:	L'ensemble des solutions réalisables de problème (P)
$v_{(P)}$	:	La valeure optimale de problème (P)
V	:	$V = \{1, ..., n\}$.L'ensemble des sommets
E	:	$E = \{e = (i, j) : i, j \in V, i \prec j\}$. L'ensemble des arêtes
c_e	:	Coût de l'arête e
$E(S)$	:	L'ensemble de tous les arêtes de E avec les deux extrémités dans S
		$E(S) = \{(i, j) \in E : i, j \in S\}$
$\delta(S)$	:	L'ensemble de toutes les arêtes avec une extrémité dans S et l'autre en $V \backslash S$
		$\delta(S) = \{(i, j) \in E : i \in S,\ j \notin S\ ou\ i \notin S,\ j \in S\}$
$\delta(i)$	:	L'ensemble des arêtes incidentes au nœud i
	:	$\delta(i) = \delta(\{i\}), i \in S$
$\cap, \cup$	:	Intersection (respetivement réunion) d'ensembles
$S \subset V$	:	S est strictiment contenu dans V.
$\|S\|$	:	Cardinale de S : nombre d'élements de l'ensemble S.
$c^T x$	:	Produit scalaire des vecteur c et x.
$x \in S$	:	x est élément de l'ensemble S
$\|.\|$	:	Norme euclidienne.
P^+	:	L'opérateur de projection $\mathbb{R}^m$
$\emptyset$	:	Ensemble vide
det	:	Déterminant d'une matrice.
RLP	:	Relaxation de LP
LB	:	Borne inférieure

Abréviations

LP	:	linear programming
ILP	:	integer linear programming
BIP	:	Binary integer program
TSP	:	Travelling Salesman Problem
AP	:	Assignment problem
$B\&B$	:	Branch and Bound

0.1 Introduction

En pratique, il arrive fréquemment que dans un problème d'optimisation certaines variables soient astreintes à être entières ou même binaire $x_j = 0$ ou 1, on parle alors de programme linéaire en nombre entiers (ILP) ou programme linéaire binaire (BIP).

Un des problèmes le plus étudié dans la classe des ILP est le problème du voyageur de commerce ("The Travelling Salesman Problem " noté " TSP"). Dans ce problème, un voyageur de commerce doit visiter plusieurs villes (ou clients) en passant une et une seule fois par chacune d'entre elles, et en minimisant la distance totale parcourue [25], l'objectif est de trouver un cycle hamiltonien [1].

En tan que problème d'optimisation, le TSP est un problème *NP-difficile* [18]. En effet, dans sa version symétrique[2], le nombre totale de solutions possibles est $\frac{(n-1)!}{2}$, où n est le nombre de villes. Avec une telle complexité factorielle, une résolution efficace du TSP nécessite donc le recours à des méthodes d'optimisation très performantes.

Les méthodes de résolution du TSP peuvent être réparties en deux groupes de nature différente :

– Le premier groupe, comprend les méthodes exactes qui garantissent la complétude de la résolution : c'est le cas de la méthode Séparation et Evaluation ("Branch and Bound" noté "$B\&B$") ; le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre. Citons aussi les méthodes qui s'appuies sur des techniques de relaxation, telle que la relaxation Lagrangienne.

– 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.

Le présent travail traite avec les deux approches, exactes et approchées.

Le mémoire est constitué de quatre chapitres :

Le premier chapitre est une introduction aux problèmes ILP, il aide aussi à mieux mesurer en quoi l'introduction de variable discrètes dans un modèle de programmation linéaire accroît considérablement le degré de difficulté du problème. L'accent sera mis sur le problème du TSP comme exemple de problèmes en variables binaires, cependant et pour des raisons techniques nous introduisons aussi le problème d'affectation ("Assignement Problem" noté "AP").

Au second chapitre, nous présentons les principes de base d'une méthode Branch and bound, nous donnons ensuite une variante adaptée au TSP, avec une relaxation AP et évaluation par la méthode Hongroise.

La dualité et relaxation Lagrangienne ferons l'objet du troisième chapitre ; l'application de la méthode sous-gradient à la fonction dual Lagrangienne avec un bon choix sur les contraintes dualisées donnera naissance à un algorithme exacte très performant.

Sans être exhaustif, nous proposons au quatrième chapitre une méthodes approchée ou métaheuristique appelée méthode de Recherche Tabou.

Nous concluons par clarifier quelques perspectives et les axes qui méritent réflexions.

[1] un cycle hamiltonien est un cycle passant une et une seule fois par tous les sommets du graphe,et de longueur minimale.

[2] c-àd dans le cas où le graphe associé n'est pas orienté.

Chapitre 1

Programmation linéaire en variables entières

1.1 Définition d'un ILP

1.1.1 Formulation d'un ILP

Considérons la formulation mathématique générale d'un problème (LP) de programmation linéaire en variables continues ([5],[35]):

$$(LP)\begin{cases} z = \min c^T x \\ x \in D \end{cases} \tag{1.1}$$

où D représente le polyèdre convexe des solutions admissibles dans $\mathbb{R}^n_+$

$$D = \{x \setminus Ax \leq b, x \geqslant 0\} \tag{1.2}$$

avec $A \in \mathbb{R}^{m\times n}$ et le second membre $b \in \mathbb{R}^{m\times 1}$.

Lorsque toutes les variables doivent être entières, le problème résultant noté (ILP) ("*integer linear programming*") est le problème générale de la programmation linéaire en variables entières ([19],[37],[28]) :

$$(ILP)\begin{cases} z = \min c^T x \\ x \in S = D \cap \mathbb{Z}^n_+ \end{cases} \tag{1.3}$$

Dans le cas particulier où les contraintes $x \in D \cap \mathbb{Z}^n_+$ sont remplacées par $x \in \{0,1\}^n$ on dit qu'on a un programme linéaire en $0-1$ où programme linéaire en variables binaires(" *Binary* integ*er program*") noté (BIP) :

$$(BIP)\begin{cases} z = \min c^T x \\ Ax \leq b \\ x \in \{0,1\}^n \end{cases} \tag{1.4}$$

La question de résoudre efficacement un ILP se pose , on sait résoudre efficacement un LP. En théorie, le problème est de complexité polynômiale (voir [35],[29],[1]) et en pratique, bien que exponentiel dans le pire des cas, le simplexe ([35],[1]) fonctionne bien. Peut-on obtenir les mêmes résultats pour le problème ILP ?

1.1.2 Relaxation linéaire continue

Le problème (LP) (donné par (1.1)) est appelé aussi la relaxation linéaire du problème (ILP), il s'agit d'un relachement de la contrainte $x \in D \cap \mathbb{Z}_+^n$ en $x \in D$ (D donné par la relation (1.2)). L'exemple suivant nous dit que la solution du (RLP) n'a parfois rien à voir avec la solution (ILP).

Exemple 1.1 *Considérons le programme linéaire en nombre entier suivant :*

$$\begin{cases} z = \max & 1.00x_1 + & 0.64x_2 \\ & 50x_1 + 31x_2 & \leq 250 \\ & 3x_1 - 2x_2 & \geq -4 \\ & x_1, x_2 & \textit{entières} \end{cases}$$

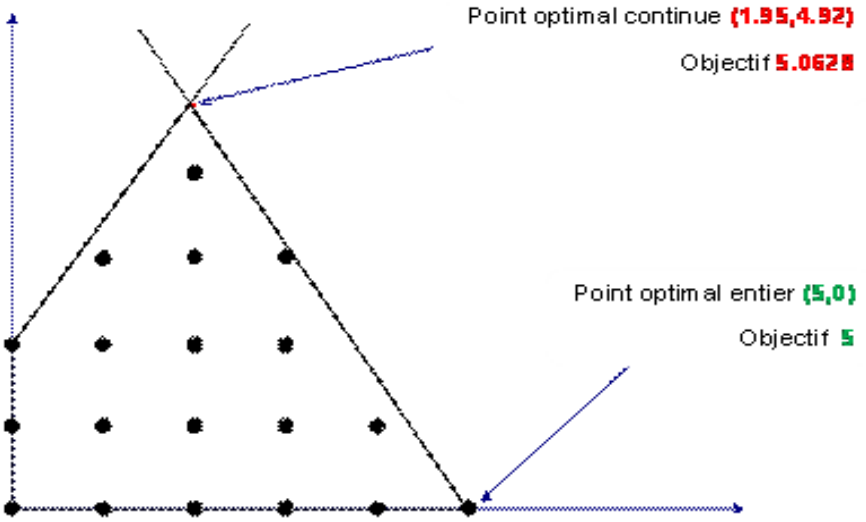


*Fig*1.1 : *Répresentation graphique*

En variables continues, la solution optimale est : $x_1 = 1.95$, $x_2 = 4.92$ *qui n'est pas entière (comme on le voit Fig* 1.1*).*

Un arrondi de cette solution serait $x_1 = 2$ *et* $x_2 = 5$ *. Ce n'est en aucun cas une solution du* (ILP) *pour deux raisons :*

- premièrement, cette solution ne satisfait pas les contraintes ;

- deuxièmement, ce n'est pas la solution entière du problème !

La solution entière du problème est $x_1 = 5$ *et* $x_2 = 0$ $(z = 5)$ *. On remarque même qu'elle est très éloignée de la solution optimale continue.*

Le seul lien exact qu'on peut établir (pour un problème de maximisation[1]) est que si $v_{(LP)}$ est la valeur de la fonction objective optimale du problème (LP), et $v_{(ILP)}$ est la valeur de la fonction objective du problème (ILP), alors $v_{(LP)} \geq v_{(ILP)}$.

[1]Pour un problème de minimisation, on aura $v_{(LP)} \leq v_{(ILP)}$.

1.2 Problèmes linéaires à solutions entières

Certains problèmes à coefficients entières ont naturellement des solutions entières, comme on va le voir pour le problème d'affectaion.

Définition 1.1 *Une matrice B carrée à coefficients entières est dite unimodulaire* (**UM**) *si le déterminant* $\det(B) = \mp 1$.

Définition 1.2 *Une matrice A $(m \times n)$ à coefficients entiers est dite totalement unimodulaire* (**TUM**) *si les déterminants de toutes ses sous-matrices carrées valent* $0, 1$ *ou* -1.

Si B est formée à partire d'un sous-ensemble de m colonnes linéairement indépendantes de A, elle détermine la solution de base (voir [35]) :

$$x = B^{-1}b = \frac{B^{adj}b}{\det(B)},$$

où B^{adj} est l'adjoint de B, et si B est **UM** et b est entier, on en déduit que le vecteur x est entiere.

Considérons le problème (LP) (donné par (1.1)) donné sous la forme standard, i.e.

$$D = \{x : Ax = b, \quad x \geq 0\}$$

Nous avons le théorème suivant :

Théorème 1.1 *Si A est TUM, alors tous les sommets de D sont entiers pour tout vecteur entier b.*

Donc, la résolution par le simplexe d'un ILP standard relaxé dont la matrice est **TUM** donne la solution entiere exacte de ILP.

Dans le cas de contraintes inégaliter (i.e D donné par la relation (1.2)). Nous avons le théorème suivant :

Théorème 1.2 *Si A est* **TUM** *, alors tous les sommets du D sont entiers pour tout vecteur entier b*

Preuve. Il revient donc à montrer que si A est **TUM**, alors (A/I) l'est. Nous ajoutons des variables d'écart et on applique le théorème (1.1). Soit C une sous-matrice carré inversible de (A/I). Les lignes de C peuvent être permutés afin de pouvoir écrire

$$C = \left(\begin{array}{c|c} B & 0 \\ \hline L & I_k \end{array}\right)$$

où I_k est une matrice identité de taille k et B est une sous-matrice carrée de A . On a $\det(C) = \det(B) = \pm 1$ parce que A est **TUM** et C est inversible. ■

Remarque 1.1 *La condition d'unimodularité est suffisante mais non nécessaire pour l'existence d'une solution optimale entière du problème* (LP) *. En pratique, de nombreux problèmes d'optimisation à données entières ont une solution entière bien que la matrice A ne soit pas unimodulaire, mais l'intégrité de la solution risque d'être perdue si des paramètres changent.*

Théorème 1.3 [29] *(Condition suffisante)*

Soit A une matrice contenant seulement les éléments 0, $+1$ *ou* -1 *et qui satisfait les deux conditions suivantes :*

1- Chaque colonne contient au plus deux éléments non-nuls.

2- Les lignes de A peuvent être partitionnées en deux sous-ensembles I_1 *et* I_2 *tels que pour chaque colonne contenant deux éléments non-nuls :*

-si les deux éléments non-nuls ont le même signe alors l'un est dans I_1 *et l'autre dans* I_2*.*

-si les deux éléments non-nuls sont de signes différents alors ils sont tous les deux dans I_1 *ou tous les deux dans* I_2*.*

Alors A est ***totalement unimodulaire****.*

Preuve. La preuve est par récurrence sur la taille des sous-matrices. Nous observons que toute sous-matrice d'un élément est TUM, c-à-d pour toute sous-matrice C de type (1×1) clairement det $C = 0, \pm 1$. Supposons maintenant, pour toute sous-matrice C de taille k . Si C a une colonne nulle, elle est singulière (det $C = 0$). Si une certaine colonne de C contient des éléments non nuls, le calcul du déterminant par rapport a cette colonne donne par hypothèse de récurrence le résultat voulu . Si chaque colonne de C contient exactement deux entrées non nulle, alors les conditions 1 et 2 du théorème (1.3) impliquent que

$$\sum_{i \in I_1} a_{ij} = \sum_{i \in I_2} a_{ij} \ \forall j$$

d'où une combinaison linéaire des lignes est nulle, par conséquent $det(C) = 0$. ■

1.3 Exemples

La programmation linéaire en nombres entiers est déjà très utile comme langage de modélisation, elle permet de décrire de façon concise et de communiquer de problèmes d'optimisation discrèts. Nous donnerons ici deux exemples, nous réferons le lecteur aux ouvrages ([27], [34], [29]) pour un grand nombre d'exemples pratiques de (ILP) .

1.3.1 Problème d'affectation

Le problème d'affectation ("*Assignment problem*", AP) est un problème particulier de (LP) en variables binaires, il consiste à affecter n ressources (*personnes,équipements, localisations ...*) à n activités *(travaux, activités, services ...)*. Il s'agit donc de définir une bijection de $\{1, ..., n\}$ sur $\{1, ..., n\}$ ou encor une permutation de n objets.

Désignons par $i = 1, ..., n$ les ressources, $j = 1, ..., n$ les activités , et introduisons les n^2 variables binaires

$$\begin{cases} x_{ij} = 1 & \text{si la ressource } i \text{ est affectée à l'activité } j \\ x_{ij} = 0 & \text{autrement} \end{cases}$$

Les contraintes du problème d'affectation s'écrivent donc simplement :

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

(la ressource i doit être affectée à une seule activité j)

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

(l'activité j ne peut être affectée qu'à une et une seule ressource i).

Nous ne traiterons ici que le problème linéaire d'affectation. L'affectation de la ressource i à l'activité j occasionne un coût d'affectation c_{ij}.

Le problème linéaire d'affectation s'écrit donc :

$$(AP) \begin{cases} \min & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \\ & \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \\ & \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \\ & x_{ij} = 1, 0 \end{cases}$$

La matrice des contraintes A est donnée par :

$$A = \begin{array}{c} \begin{array}{cccccccccc} x_{11} & \cdots & x_{1n} & x_{21} & \cdots & x_{2n} & \cdots & x_{n1} & \ldots & x_{nn} \end{array} \\ \begin{pmatrix} 1 & \cdots\cdots & 1 & & & & & & & \\ & & & 1 & \cdots\cdots & 1 & & & & \\ & & & & & & \ddots & & & \\ & & & & & & & 1 & \cdots\cdots & 1 \\ 1 & & & 1 & & & & 1 & & \\ & \ddots & & & \ddots & & & & \ddots & \\ & & 1 & & & 1 & & & & 1 \end{pmatrix} \end{array} \begin{array}{l} \left.\vphantom{\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}}\right\} I_1 \\ \left.\vphantom{\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}}\right\} I_2 \end{array}$$

d'après le théorème (1.3) , la matrice A des contraintes est **TUM**, les seconds membres b sont entiers (égaux à 1) ; d'aprés le théorème(1.2) les coordonnées des sommets du D des solutions admissibles, valent 0 ou 1.

Par conséquent le caractère binaire des variables peut être omis, il est possible de se referer au propriétés et techniques de (LP) en variable continue. Cependants, etant donnée que le rang de la matrice A est égale à $2n-1$, parmi les $2n$ contraintes il y en a $2n-1$ linéairement indépendantes et que dans chaque solution de base admissible, seuls n variables seront positives (égale à 1). Les solutions de bases sont toutes fortement dégénerées ($n-1$ variables de base sont nulles). L'utilisation du simplexe pour résoudre le problème d'affectation engendrere forcément des problèmes de cyclage voir [37].

Au paragraphe suivant nous présentons une méthode simple et efficace pour résoudre le problème d' AP .

Algorithme hongrois

L'Algorithme à été développé par Kuhn en 1955 ([22]) , et il est de complexité polynômiale. Nous donnons ces étapes dans l'exemple d'illustration suivant.

Exemple 1.2 *Soit le problème d'affectation linéaire avec* $n = 4$ *et* c *la matrice initiale des coûts*

$$c = \begin{pmatrix} 24 & 10 & 21 & 11 \\ 14 & 22 & 10 & 15 \\ 15 & 17 & 20 & 19 \\ 11 & 19 & 14 & 13 \end{pmatrix}$$

Les zéros supprimés par suite d'une affectation d'un autre zéro sur la même ligne ou colonne seront appelés les zeros barrés.

Les zéros affectés seront appelés les zéros encadrés.

Etape 1 : Réduction des lignes *:*

Pour chaque ligne, on prend la valeur minimum de la ligne que l'on soustrait à toutes les valeurs de la ligne.

$u_i = \min_{j=1,...,4} c_{ij}$,$i = 1, ..., 4$;

$c_{ij} \longrightarrow c_{ij} - u_i$

	1	2	3	4	U_i
1	24	10	21	11	10
2	14	22	10	15	10
3	15	17	20	19	15
4	11	19	14	13	11

Réduction des lignes $\longrightarrow$

	1	2	3	4	U_i
1	14	0	11	1	10
2	4	12	0	5	10
3	0	2	5	4	15
4	0	8	3	2	11

Etape 2 *:* Réduction des colonnes *:*

Créer une nouvelle matrice de coûts en choisissant le coût minimal dans chaque colonne et en le soustrayant de chaque coût dans la colonne

$v_j = \min_{i=1,...,4} c_{ij} - u_i$, $j = 1, ..., 4$

$c_{ij} \longrightarrow c_{ij} - u_i - v_j$,*la matrice réduit :* $\bar{c}_{ij} = c_{ij} - u_i - v_j$

	1	2	3	4	U_i
1	14	0	11	1	10
2	4	12	0	5	10
3	0	2	5	4	15
4	0	8	3	2	11
V_j	0	0	0	1	

Réduction des colonnes $\longrightarrow$

	1	2	3	4	U_i
1	14	0	11	0	10
2	4	12	0	4	10
3	0	2	5	3	15
4	0	8	3	1	11
V_j	0	0	**0**	1	**47**

Etape 3 :

-Si l'affectation n'est pas déjà admissible, détermine le nombre minimal de lignes nécessaires sur les lignes et les colonnes pour couvrir tous les zéros.

- Si ce nombre est égal au nombre de lignes (ou colonnes), la matrice est réduite ; aller à l'étape 5.

- Si ce nombre est inférieur au nombre de lignes (ou colonnes), aller à l'étape 4.

Dans ce cas, le nombre minimal de lignes est de 3. Donc, on va à l'étape 4.

	1	2	3	4
1	14	0	11	0
2	4	12	0	4
3	0	2	5	3
4	0	8	3	1

Etape 4 :

-Trouver la cellule de valeur minimum non-couverte par une ligne.

-Soustraire cette valeur de toutes les cellules non couvertes.

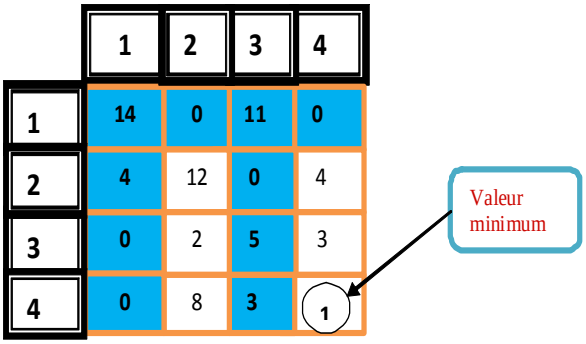
-Ajouter cette valeur aux cellules situées à l'intersection de deux lignes.

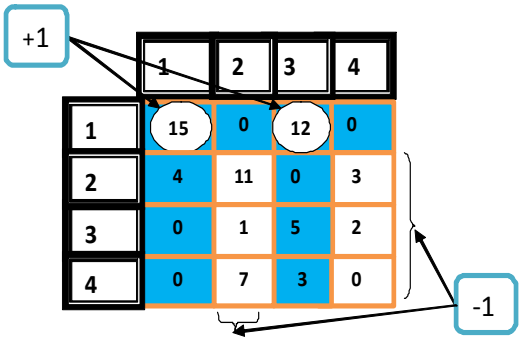
Retourner à l'étape 3.

ici, le plus petit élément parmi les élements découvrets est $\theta = 1$

-Soustraire 1 *de tous les élements qui ne soit pas couvrets*

- Ajouter 1 *à ceux qui sont entrées à l'intersection de deux lignes.*





Maintenant, le nombre minimal de lignes est de 4.

	1	2	3	4
1	15	0	12	0
2	4	11	0	3
3	0	1	5	2
4	0	7	3	0

Donc, on passe à l'étape 5.
Etape 5 *:* Déterminer la solution optimale.

	1	2	3	4
1	15	0	12	~~0~~
2	4	11	0	3
3	0	1	5	2
4	~~0~~	7	3	0

Elle contient à présent $n = 4$ affectations est donc une solution optimale :

$$\widetilde{x}_{12} = \widetilde{x}_{23} = \widetilde{x}_{31} = \widetilde{x}_{44} = 1$$

avec coût total :

$$\widetilde{z} = c_{12} + c_{23} + c_{31} + c_{44} = 10 + 10 + 15 + 13 = 48$$

Théorème 1.4 *(voire [18] page 250)*

La méthode Hongroise résoud correctement le problème AP en $O(n^3)$ opération arithmétiques.

1.3.2 le problème du TSP

Le problème du voyageur de commerce ("*Travelling Salesman Problem* " TSP) est l'un des problèmes le plus étudié dans l'optimisation discrète. bien que sa formulation est simple, il reste l'un des problèmes les plus difficiles à resoudre .

Soit $G = (V, A)$ un graphe où V est un ensemble de n sommets (n villes). A est un ensemble d'arcs ou arêtes (trajet $i - j$), et soit $C = (c_{ij})$ la matrice des coûts associée à A (distance entre la vile i et la vile j) .

Le TSP consiste à déterminer un circuit à distance minimale passant par chaque sommet une et une seule fois. Un tel circuit est connu comme un tour ou un circuit hamiltonien . Dans plusieurs autre applications, C peut aussi être interprété comme une matrice de coût de temps de voyage.

Formulations

Introduisant la variable x_{ij} qui vaut 1 si le tour contient le trajet $i-j$, et 0 autrement. c-à-d

$$x_{ij} = \begin{cases} 1 & \text{si le tour contient le trajet } i-j \\ 0 & \text{sinon} \end{cases}$$

Le problème TSP se formule ainssi :

$$\min \sum_{i \neq j} c_{ij} x_{ij} \tag{1.5}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, ..., n \tag{1.6}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, ..., n \tag{1.7}$$

$$\sum_{i \in S} \sum_{j \in \overline{S}} x_{ij} \geq 1 \ , S \subset V, 2 \leq |S| \leq n-2 \tag{1.8}$$

$$x_{ij} \in \{1, 0\} \qquad i, j = 1, ..., n \ \ , i \neq j \tag{1.9}$$

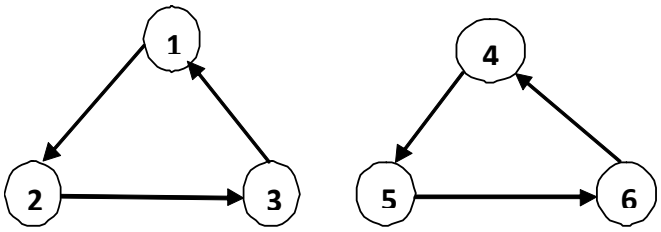
Dans cette formulation, la fonction objective décrit le coût de tournée optimale.

Les contraintes (1.8) sont les contraintes d'elimination de sous-tours .

Les contraintes (1.6) et (1.7) ne sont pas suffisantes pour définir les tours du moment qu'ils sont satisfaits aussi par les sous tours , ex :

$$x_{12} = x_{23} = x_{31} = x_{45} = x_{56} = x_{65} = 1$$

satisfaits pour (1.6) et (1.7) et (1.9) , mais ne correspond pas à un tour voir la figure ci-dessous,



Un moyen d'éléminer ces sous-tours est d'observer que dans un tour il doit y avoir un arc allant de $\{1, 2, 3\}$ vers $\{4, 5, 6\}$ et un arc allant de $\{4, 5, 6\}$ vers $\{1, 2, 3\}$.

En général pour tout $S \subset V$ avec $2 \leq |S| \leq n-2$, la contrainte (1.8) est satisfaite pour tous les tours. Un sous-tour est forcement violé par les contrainte (1.8) .

Une autre formulation de la contrainte (1.8) est donné par le relation suivante :

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad S \subset V, 2 \leq |S| \leq n-2 \tag{1.10}$$

Cette formulation contient $n(n-1)$ variables binaires, $2n$ contraintes de degré et $2^n - 2n - 2$ contraintes d'élimination de sous-tours. Ceci constitue un grand chalenge pour la résolution numérique du TSP .

1.4 Méthodes de résolution

Il existe trois grandes catégories de méthodes de résolution des problèmes ILP : les méthodes exactes , les méthodes heurestique et les méthodes métaheurestique.

Les méthodes exactes permettent d'obtenir une solution optimale à chaque fois, mais le temps de calcul peut être long si le problème est compliqué à résoudre.

Les méthodes heuristiques sont des méthodes specifique , permettent quant à elles d'obtenir rapidement une solution approchée, mais qui n'est donc pas toujours optimale .

Les méthodes métaheurestique, s'adaptent quasiment à tous les problèmes d'optimisation combinatoire . Les plus connues et les plus utilisées sont celles des " Recuit similé " et de la "recherche Tabou " et les " Algorithme génétique " . Leur principe est basé sur une recherche aléatoire guidée au sein d'un voisinage de la solution courante.

Pour le problème TSP, l'une des méthodes exactes les plus classiques et les plus performantes reste la Procédure par Séparation et Evaluation (Branch and Bound). Cette méthode est présentée aux chapitre suivant.

Chapitre 2

Méthode Branch and Bound

Ce chapitre est consacré à la méthode par Séparation et Evaluation (Branch-and-Bound [37]). Les procédures par séparation et évaluation (PSE) sont le moyen générique le plus utilisé pour la résolution exacte des problèmes d'optimisation combinatoire, et en particulier pour la résolution des ILP qui pratiquent une énumération intelligente de l'espace des solutions.

2.1 Condition d'optimalité par les bornnes

Soit le problème en variables discrètes

$$(P)\begin{cases} \min & f(x) \\ & \\ & x \in S \subset Z^n \end{cases}$$

où S contient un nombre fini (ou éventuellement un nombre infini dénombrable) de solutions, $f(x)$ fonction quelconque.

Une tâche essentielle dans la conception de n'importe quel algorithme est de tirer une condition optimale ou un critère d'arrêt à la fin de l'algorithme, c'est-à-dire, pour juger si la solution courante est optimale pour (P) ou de conclure qu'il n'y a pas de solution réalisable pour (P). à l'exception de très rares cas particuliers. les condition d'optimalité du problème (ILP) peut être vérifiées grâce à la convergence d'une suite de bornes supérieures et une suite de bornes inférieures de la fonction objective.

Soit f^* la valeur optimale de (P). Supposons qu'un algorithme génère une suite décroissante des bornes supérieures :

$$\bar{f}_1 \geq \bar{f}_2 \geq \cdots \geq \bar{f}_k \geq \cdots \geq f^*$$

et une suite croissante des bornes inférieures :

$$\underline{f}_1 \leq \underline{f}_2 \leq \cdots \leq \underline{f}_k \leq \cdots \leq f^*$$

où $\underline{f}_k$ et $\bar{f}_k$ sont les bornes inférieures et supérieure de f^* générés à l'itération k, respectivement.

Si $\bar{f}_k - \underset{-k}{f} \leq \varepsilon$ est vérifié pour un certains $\varepsilon \geq 0$ à l'itération k, on a :

$$f^* - \varepsilon \leq \underset{-k}{f} \leq f^*$$

Remarquons qu'une borne supérieure de f^* est souvent associée à une solution réalisable x^k de (P), puisque $f(x^k) \geq f^*$. Une borne inférieure de f^* est généralement obtenue en résolvant un problème de relaxation de (P) .

Une solution réalisable x^k est appelé une solution $\epsilon - approchée$ de (P) lorsque $f(x^k) = \bar{f}_k$ et $\bar{f}_k - \underset{-k}{f} \leq \varepsilon > 0$.

Nous avons le théorème suivant :

Théorème 2.1 *Supposer que* $\left\{ \bar{f}_k \right\}$ *et* $\left\{ \underset{-k}{f} \right\}$ *sont les suites des bornnes supérieures et des bornnes inférieures de* f^**, respectivement.*

Si le $\bar{f}_k - \underset{-k}{f} = 0$ *pour un certain* k *et* x^k *est solution réalisable à* (P) *avec,* $f(x^k) = \bar{f}_k$*, alors* x^k *est une solution optimale de* (P)*.*

2.2 Présentation de l'algorithme "Branch and Bound"

A chaque étape la méthode Branch and Bound $(B\&B)$ subdivise l'ensemble S en un nombre fini de sous - ensembles $S^{(i)}$ en veillant à ce que :

$$\bigcup_{i=1}^{p} S^{(i)} = S$$

De cette façon , le problème se réduit à l'étude des p problèmes $P^{(i)}$ plus restreints :

$$\left(P^{(i)}\right) \begin{cases} \min \quad f(x) \\ \\ \qquad x \in S^{(i)} \end{cases}$$

Ces subdivisions successives sont représentées à l'aide d'une arborescence (voire figure 2.1)

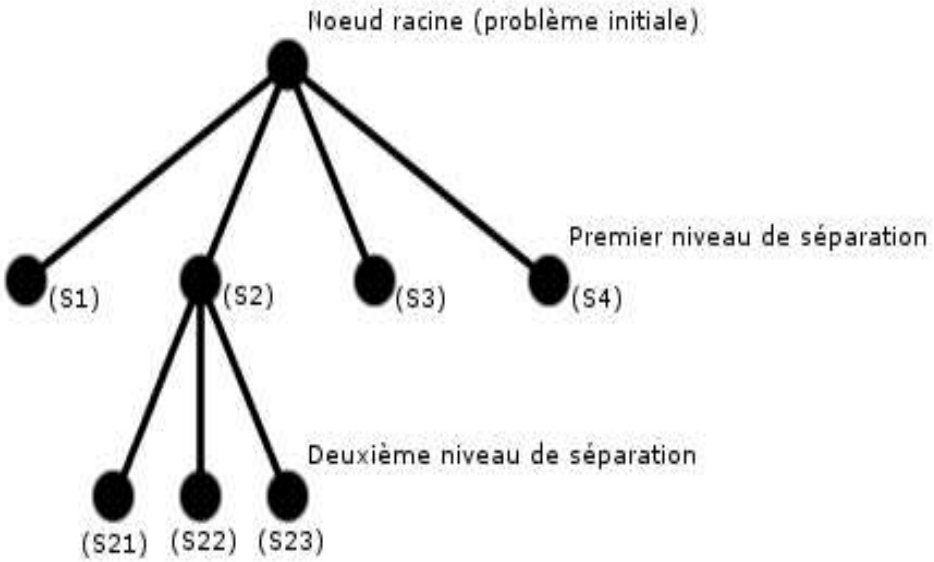


Fig 2.1 – *Arbre généré par décomposition du sous – problème initial*

L'algorithme consiste à séparer de manière récursive le problème en sous-problèmes de cardinalité inférieure. Le cardinal de l'ensemble à explorer est réduit en imposant à cet ensemble des contraintes supplémentaires (réduction du domaine). Une série de tests, appliquée à tous les sous-problèmes permet de supprimer de l'espace de recherche les sous-problèmes qui ne peuvent pas engendrer de solution optimale.

Cette recherche par décomposition de l'ensemble des solutions peut être repréesentée graphiquement par un arbre (figure 2.1). C'est de cette représentation que vient le nom de "méthode de recherche arborescente" .

– Chaque sous-problème créé au cours de l'exploration est symbolisé par un noeud de l'arbre (ou sommet), le noeud racine représentant le problème initial.

– Les branches de l'arbre symbolisent le processus de séparation, ils représentent la relation entre les noeuds.

– Les noeuds non séparés, appelés noeuds pendants ou sondé(par exemple, $(S_1), (S_3)$ *et* (S_4),de la figure(2.1)).

Plus précisément, le branch-and-bound est basé sur trois principes :

- le principe de séparation.
- le principe d'évaluation.
- la stratégie de parcours ou procédure de cheminement.

2.2.1 Le principe de séparation :

Le principe de séparation permet d'établir l'ensemble des règles qui vont régir la séparation d'un ensemble en sous-ensembles. Afin d'assurer l'optimalité de la solution fournie par le branch and bound, certaines règles doivent être respectées.

– **Règle 1** : Aucune solution optimale ne doit être écartée lors d'une séparation. On utilise la règle suivante qui garantit la règle 1 :

Règle 1' : La réunion des sous-ensembles obtenus lors d'une séparation doit être égale à l'ensemble séparé, si $S^{(i)}$ est séparé en $S^{(i1)}, ..., S^{(ip)}$ alors

$$\bigcup_{k=1}^{p} S^{(ik)} = S^{(i)}$$

avec de préférence $S^{(ik)} \cap S^{(il)} = \emptyset$, pour $k \neq l$,où $S^{(ik)}$,$k = 1, ..., p$ représentent les sous -ensembles du sous -ensemble parent $S^{(i)}$.

– **Règle 2** : Le cardinal d'un sous-ensemble doit être inférieur à celui de son père.

– **Règle 3** : Un sous-ensemble qui ne peut être séparé doit être sondable.

2.2.2 Principe d'évaluation :

Définition 2.1 *Une fonction d'évaluation est une fonction* $v : S^{(i)} \longrightarrow v_i$ *qui associe une valeur* v_i *à chaque sous-ensemble* $S^{(i)}$.

Le principe d'évaluation permet de diminuer l'espace de recherche. L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence.

Le branch-and-bound utilise, à des fins d'élagage (´elimination de branches dans l'arborescence de recherche), deux fonctions :

– une borne inférieure de la fonction d'utilité du problème initial, qui résulte d'une fonction d'évaluation.

– une borne supérieure de la fonction d'utilité des solutions d'un sous-ensemble.

La connaissance d'une borne inférieure du problème et d'une borne supérieure de la fonction d'utilité de chaque sous-problème permet de stopper l'exploration d'un sous-ensemble de solutions ne pouvant pas contenir de solutions candidates à l'optimalité . Si pour un sous-problème la borne supérieure est plus petite que la borne inférieure du problème, l'exploration du sous ensemble correspondant est inutile. D'autre part, lorsque le sous-ensemble est suffisamment « petit », on procède à une évaluation dite exacte : on résout alors le sous-problème correspondant.

La valeur v_i doit être une borne inférieure de la valeure optimale de la fonction économique du problème $P^{(i)}$:

$$\begin{cases} v_i \leq \widetilde{f_i} = \min f(x) \\ x \in S^{(i)} \end{cases}$$

En cas de maximisation :

$$\begin{cases} v_i \geq \widetilde{f_i} = \max f(x) \\ x \in S^{(i)} \end{cases}$$

Remarque 2.1 *Soient* $S^{(ik)}, k = 1, ..., p$ *les sous- ensembles obtenus par séparation du sous ensemble* $S^{(i)}$ *.Etant donné* $S^{(ik)} \subset S^{(i)}$*, il vient*

$$\widetilde{f_i} \leq \widetilde{f_{ik}} \qquad \forall k$$

et par conséquent

$$v_i \leq \widetilde{f}_{ik} \qquad \forall k$$

Ainsi la valeur de la fonction d'évaluation au nœud parent, constitue déjà une borne inférieure de la valeur optimale de la fonction économique pour tous les sous- nœuds qui en sont issue. Dès lors

$$v_i \leq v_{ik} \qquad \forall k$$

C'est-à-dire que la fonction d'évaluation est une fonction non décroissante.

Détermination d'une fonction d'évaluation

Pour obtenir la borne inférieure v_i de $\widetilde{f}_i$, la technique la plus utilisée est de procéder à une relaxation du problème $P^{(i)}$. Une telle relaxation consiste à élargir l'ensemble $S^{(i)}$ en un ensemble $R^{(i)}$ tel que $S^{(i)} \subset R^{(i)}$, et résoudre le problème relaxé $\left(RP^{(i)}\right)$:

$$\left(RP^{(i)}\right) \begin{cases} \widetilde{f}_i^R = \min f(x) \\ \qquad\quad x \in R^{(i)} \end{cases}$$

de sort que :

$$v_i = \widetilde{f}_i^R \leq \widetilde{f}_i .$$

Bien évidemment , il faut définir l'ensemble $R^{(i)}$ de façon à ce que qu'il soit possible - et de préférence assez simplement - de résoudre le problème relaxé $\left(RP^{(i)}\right)$

Les techniques de relaxation sont souvent utilisées dans le calcul d'une borne inférieure. Appliquées à un problème de minimisation, elles fournissent une évaluation par défaut de l'optimum, en relâchant les contraintes les plus difficiles à satisfaire, c'est-à-dire en les supprimant, ou en les prenant partiellement en compte .

Voici dans ce qui suit, les principales techniques de relaxation :

a)Relaxation linéaire :

déjà introduite au pragraphe (1.1.2)

b)Relaxation lagrangienne :

La relaxation lagrangienne s'articule sur l'idée de relâcher les contraintes difficiles, non pas en les supprimant totalement, mais en les prenant en compte dans la fonction objectif de sorte qu'elles pénalisent la valeur des solutions qui les violent [voire paragraphe (3.1.1)].

c)Relaxation des contraintes :

Une technique simple de relaxation consiste à ignorer certaines contraintes du problème. On obtient alors un problème dont la solution optimale est plus facile à calculer. Par exemple, soit un polyèdre $D^{(k)}$ défini par deux ensembles de containtes :

$$D^{(k)} = \{x \backslash A_1 x \leq b_1, A_2 x \leq b_2\}$$

Il peut s'avérer que la suppression d'un des deux ensembles de contraintes par exemple le second permet d'obtenir un polyèdre convexe

$$D'^{(k)} = \{x \backslash A_1 x \leq b_1\}$$

tel que le problème relaxé $\left(RP^{(i)}\right)$ défini par $R^{(i)} = D'^{(k)} \cap \mathbb{Z}_+^n$ est un problème classique ,évident ou simple à résoudre . Le probléme relaxé $\left(RP^{(i)}\right)$ est alors appelé sous problème du problème $\left(P^{(i)}\right)$ et la fonction d'évalution est obtenue par résolution du sous-problème .

Utilisatoin de la fonction d'évaluation : sondage d'un ensemble

Supposons connaitre des solutions admissibles du problème (P). ces solutions peuvent être :

- soit des solutions admissibles évidentes ;
- soit des solutions déterminées par application d'une heuristique ;
- soit des solutions générées en cours de procédure, lors de la résoltion de certains problèmes relaxés.

Notons $\widehat{x}$ la meilleure d'entre elles et $\widehat{f} = f(\widehat{x})$ qui vérifie donc :

$$\widehat{f} \leq v_i$$

et puisque $v_i \leq \widetilde{f_i}$ il vient :

$$\widehat{f} \leq \widetilde{f_i}$$

Donc si $\widehat{f} \leq v_i$, on peut affirmer qu'il n'y a dans $S^{(i)}$ aucune solution admissible meilleure que $\widehat{x}$ .Le nœud i est donc sondé.

Conditions ou tests de sondage :

Plus généralement un sous-ensemble $S^{(i)}$ sera sondé si une des trios conditions suivantes est réalisée.

α) $R^{(i)} = \emptyset$: dans ce cas , a fotiori , $S^{(i)} = \emptyset$

β) La solution optimale du problème relaxé $\left(RP^{(i)}\right)$ appartienne à $S^{(i)}$:

- Il s'agit de la solution optimale du problème $P^{(i)}$ $\left(\text{et } v_i = \widetilde{f_i}\right)$.
- si de plus $\widetilde{f_i} \leq \widehat{f}$, il convient d'actualiser $\widehat{x}$ et $\widehat{f}$.

γ) $\widehat{f} \leq v_i$: dans ce cas $f(\widehat{x}) \leq f(x) \quad \forall x \in S^{(i)}$

Remarque 2.2 *ces trois conditions appelées parfois respectivement conditions d'optimalité et de dominance permettent d'élaguer les branches de l'arbrescence issues du nœud i . il ne doit donc plus être considéré puisque toute l'information utile en été extraite.*

2.2.3 La stratégie de parcours (Procédure de cheminement)

La stratégie de parcours est la règle suivante dans laquelle est choisi le sommet qui doit être séparé parmi tous les sommets pendants de l'arborescence. Parmi les stratégies de parcours les plus connues, on peut citer :

– **La profondeur d'abord** ([**33**]) **:** par fois dénommée "procédure par séparation et évaluation séquentielle"

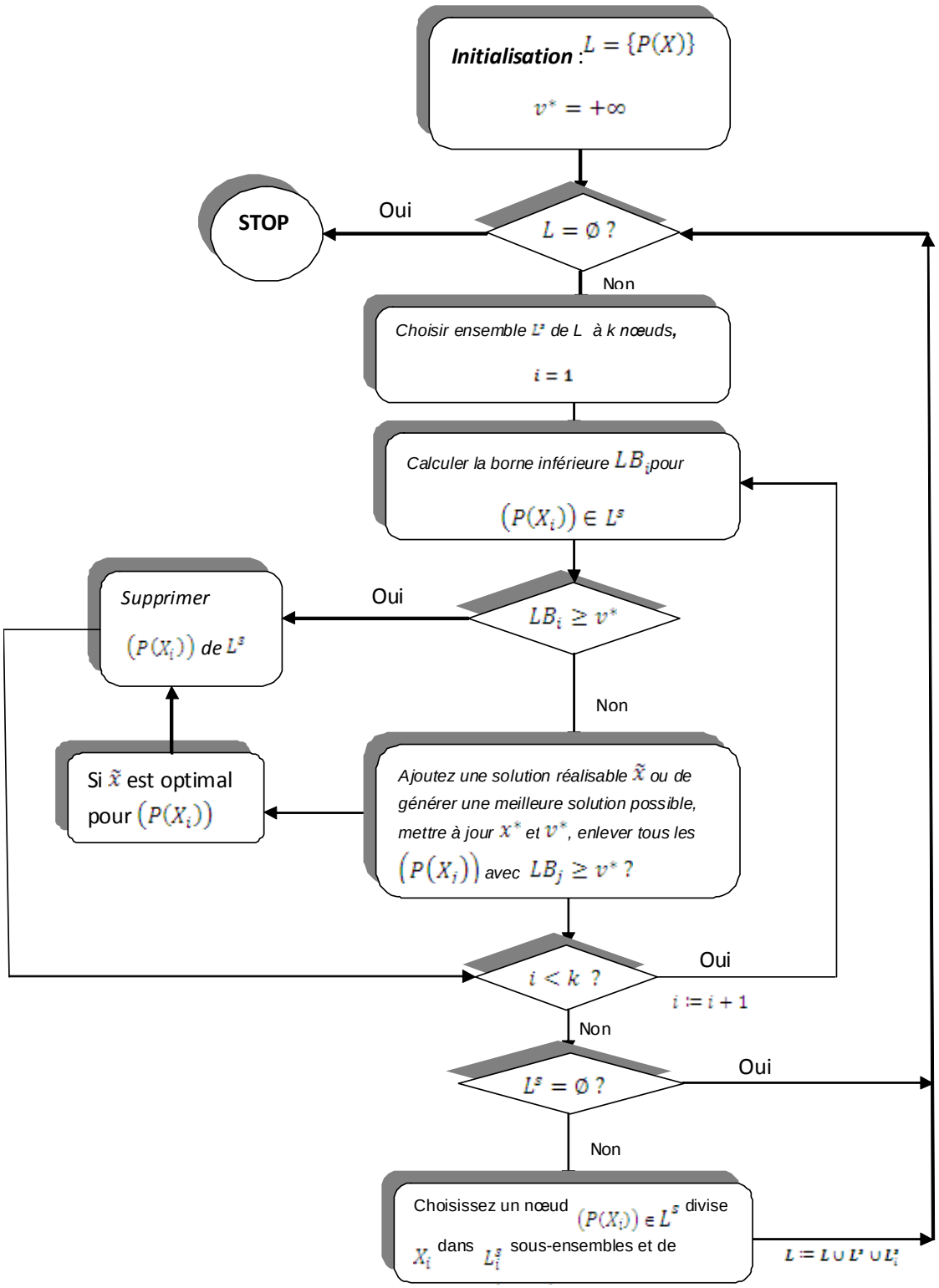
L'exploration privilégie les sous-problèmes obtenus par le plus grand nombre de séparations appliquées au problème de départ, c'est-à-dire aux sommets les plus éloignés de la racine (de profondeur la plus élevée). L'obtention rapide d'une solution optimale et le peu de place mémoire nécessaire en sont les avantages. L'inconvénient est l'exploration de sous-problèmes peu prometteurs à l'obtention d'une solution optimale.

– **La largeur d'abord** **:** Cette stratégie favorise les sous-problèmes obtenus par le moins de séparations du problème de départ, c'est à dire les sommets les plus proches de la racine. Il est à noter que cette stratégie est peu utilisée car elle présente une efficacité plus faible que les deux autres stratégies présentées.

– **Le meilleur d'abord** ([**33**]) **:** par fois dénommée "procédure par séparation et évaluation progressive"

Cette stratégie favorise l'exploration des sous-problèmes possédant la plus grande borne supérieure. Elle dirige la recherche là où la probabilité de trouver une meilleure solution est la plus grande. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possédent une évaluation inférieure à la valeur optimale.

2.2.4 Branch and bound :organigramme



2.3 Branch and bound pour le TSP, avec Relaxation AP et évaluation avec la méthode hongroise

– Si on retire les contraintes d'élimination de sous-tours, on obtient le problème d'affectation.

– Cette relaxation a une solution entière qui peut être obtenue par exemple avec la méthode hongroise.

– Le branchement est effectué de manière à éliminer les sous-tours.

– La valeur de la solution optimale du problème d'affectation AP est une borne inférieure sur la valeur de la solution optimale du TSP.

– Le coût d'un tour fournit une borne supérieure sur la valeur de la solution optimale.

– Si la solution optimale de AP est un tour (i.e. sans sous-tour), elle est également la solution optimale du TSP.

– Si un sous-tour apparaît :

$$x_{i_1 i_2} = x_{i_2 i_3} = x_{i_3 i_4} = \cdots = x_{i_{k-1} i_k} = x_{i_k i_1} = 1$$

– Dans une solution admissible, un de ces arêtes doit être absent, donc :

$$\begin{cases} x_{i_1 i_2} = 0 \\ \text{ou} \\ x_{i_2 i_3} = 0 \\ \text{ou} \\ \vdots \\ \text{ou} \\ x_{i_{k-1} i_k} = 0 \\ \text{ou} \\ x_{i_k i_1} = 0 \end{cases} \tag{2.1}$$

– Chacune de ces conditions va correspondre à une branche de l'arbre de branch-and-bound.

Remarque 2.3 *L'étape de branchement est illustré en (figure 2.4).*

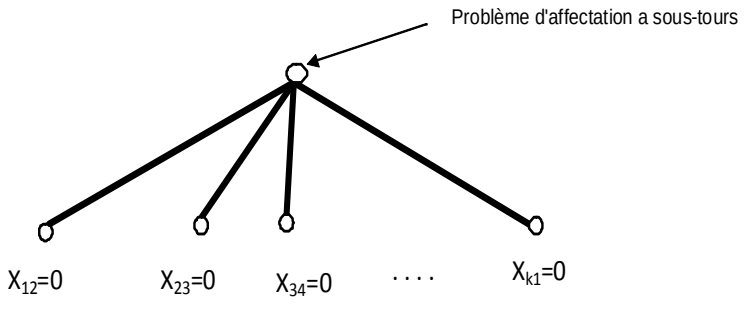


fig 2.4 : *Étape branchement*

Algorithme Branch and Bound pour le TSP

0.Initialisation :Construire une tournée initiale ($x_{\min}$) réalisable
par certains heuristiques,
ou l'utilisation $1 \to 2 \to \cdots \to n \to 1$
et ponson :$v_{(TSP)} = c_{12} + \cdots + c_{n1}$.

1.**Evaluation** : Résoudre le problème d'affectation (AP) (relaxation)
avec la méthode hongroise.
Si la solution optimale de AP est un tour
(i.e. cycle hamiltonien) , arrêt :
elle est également la solution optimale du TSP.

2.Choisissez une sous-tour avec le plus petit nombre de nœuds,
$x_{i_1 i_2} = x_{i_2 i_3} = x_{i_3 i_4} = \cdots = x_{i_{k-1} i_k} = x_{i_k i_1} = 1$, $k < n$

3.Branchement : Définir les k sous-problèmes en supprimant
chaque arête du sous-tour à leur tour,
en utilisons (2.1)

4. Choisir un sous-problème à résoudre.
Résoudre le problème d'affectation associée en plaçant :
$c_{ij} = \infty$ pour chaque arête supprimé. Dénoter la solution par $\mathbf{x}$, $\mathbf{v}$.

5. (i) **Si** $\mathbf{v} < v_{(TSP)}$, mais les sous-tours existent, aller à **2**.
(ii) (**Evaluation**)**Si** $\mathbf{v} > v_{(TSP)}$, et la solution n'a pas sous-tours,
mettre à jour la solution
$\mathbf{x} := x_{\min}$, $\mathbf{v} := v_{(TSP)}$. aller à **6.**
(iii) **Si** $\mathbf{v} > v_{(TSP)}$, aller à **6.**

6.Si les sous-problèmes non résolus existent, aller à **4** .
sinon arrêt : la solution optimale est $x_{\min}$, $v_{(TSP)}$

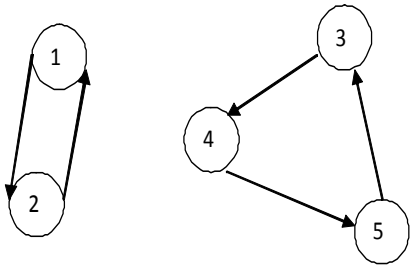
Nous allons illustrer cette méthode par un exemple TSP (asymétrique) avec 5 villes

Exemple 2.1

$$C = \begin{bmatrix} - & 2 & 0 & 6 & 1 \\ 1 & - & 4 & 4 & 2 \\ 5 & 3 & - & 1 & 5 \\ 4 & 7 & 2 & - & 1 \\ 2 & 6 & 3 & 6 & - \end{bmatrix}$$

Initialisation : Le tour $1 \to 2 \to 3 \to 4 \to 5 \to 1$ *donne* $z_{\min} = 10$
Pour la méthode hongroise, nous avons d'abord fixé $c_{ii} = \infty$ *pour forcer* $x_{ii} = 0$
la Solution du problème d'affectation associée à la méthode hongroise est :
$x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = 1$,

avec des sous tours 1→2→1 et 3→4→5→3 , $z = 8 < z_{\min}$.



Branchement :

Branchement sur le sous-tours première génère sous-problèmes

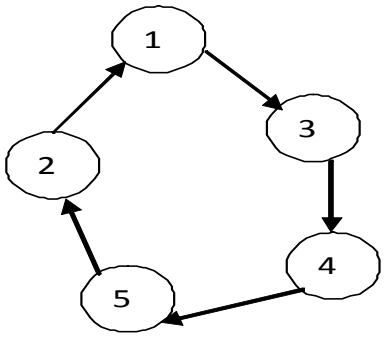
(1) $x_{12} = 0 \Longrightarrow c_{12} = \infty$

(2) $x_{21} = 0 \Longrightarrow c_{21} = \infty$

la Solution de sous-problème d'affectation (1) *associée à la méthode hongroise est :*

$x_{13} = x_{34} = x_{45} = x_{52} = x_{21} = 1$, *qui donne le tour :*

$1 \to 3 \to 4 \to 5 \to 2 \to 1$, $z = 9 < z_{\min}$

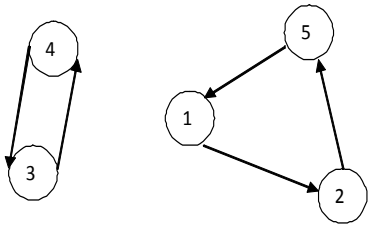


Définir $z_{\min} := 9$ *et enregistrez la solution* $x_{\min}$.

la Solution de sous-problème d'affectation (2) *associée à la méthode hongroise est :*

$x_{12} = x_{25} = x_{34} = x_{43} = x_{51} = 1$ *avec des sous tours :*

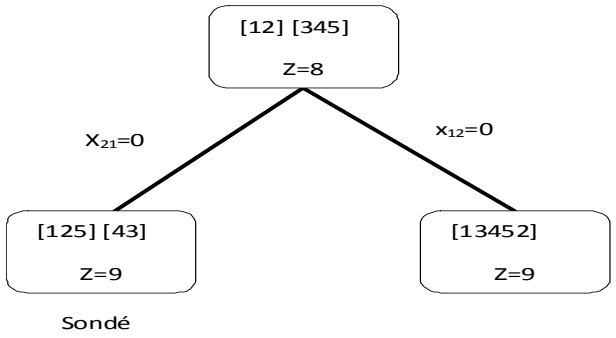
$1 \to 2 \to 5 \to 1$ *et* $3 \to 4 \to 3$, $z = 9$.



Cette borne inférieure n'est pas meilleure que la solution en place.

Tous les sous-problèmes sont sondés.

la Solution optimale est la tournée $1 \to 3 \to 4 \to 5 \to 2 \to 1$, *de longueur* $z = 9$.



Le Branch and Bound avec la relaxation AP est simple mais par fois inéfficace, en raison de sa technique de délimitation ; l'arbre du sous-problèmes peuvent devenir irréalisable. Une meilleure borne inférieure peut être calculée en utilisant d'autres relaxations.

Chapitre 3

Une approche dual-sousgradient

Quelques notions de théorie des graphes sont nécessaires pour introduire la relaxation 1-arbre. Au second paragraphe les propriétés de la dualité Lagrangienne sont adaptées au cadre de la programmation en nombres entiers. Le troisième paragraphe est consacré à l'étude de la méthode sous-gradient de base, outil fondamental pour la résolution numérique de la fonction duale Lagrangienne. En injectent les contraintes de degré dans la fonction objectif du problème TSP, la relaxation Lagrangienne serait exprimée comme une relaxation 1-arbre, l'application de la méthode sous-gradient engendrera un algorithme exacte pour la résolution du TSP.

3.1 Relaxation 1-arbre

Considérons le problème TSP spécifié par le graphe $G = (V, E)$ et c_e le poids d'une arrête $e \in E$.

Le problème est donc de trouver un tour à poids minimal

$$z^{TSP} = \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ est un tour} \right\}$$

Une relaxation intéressante est obtenue en remarquent :

a) Chaque tour est constitué de deux arrêtes adjacentes à 1 sommet et un chemin[1] reliant les sommets $\{2, ..., n\}$.

b) Un chemin est arbre[2] spécial.

Définition 3.1 *Un 1-arbre est un sous graphe de deux arêtes adjacentes au sommet* 1,

[1] Un chemin est une suite d'arcs, dont l'extrémité terminale de chacun est l'extrémité initiale du suivant sauf pour le dernier.

[2] Un arbre est un graphe connexe sans cycle.

aditionnées aux arêtes d'un arbre de sommets $\{2, ..., n\}$.

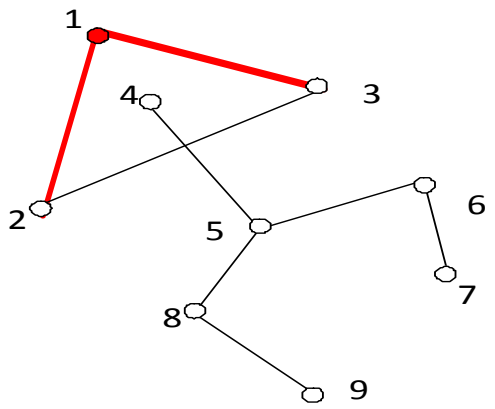


*Fig.*3.1 : *Exemple de* $1-arbre$

Remarque 3.1 *1) Il est clair que chaque tour est un 1-arbre, d'où*

$$z^{TSP} \geq z^{1-arbre} = \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ est un 1-arbre} \right\} \tag{3.1}$$

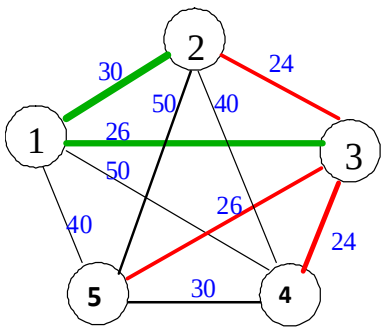
2) Un tour est un 1-arbre ayant deux arête incidentes de chaque sommet.
3) La relation (3.1)*, indique que le problème 1-arbre est une relaxation pour le* TSP.

Trouver une solution 1-arbre optimale associée à un prblème TSP est simple, comme le montre l'exemple suivant :

Exemple 3.1 *Considérons un problème* TSP *dont la matrice des coûts est*

$$C_e = \begin{bmatrix} - & 30 & 26 & 50 & 40 \\ - & - & 24 & 40 & 50 \\ - & - & - & 24 & 26 \\ - & - & - & - & 30 \\ - & - & - & - & - \end{bmatrix}$$

La solution optimale 1-arbre est calculée en prenant les deux arêtes à coût minimal sortant du sommet 1, i.e ; l'arête $(1, 2)$ *et* $(1, 3)$*, plus les arêtes d' un arbre optimal des sommets* $\{2, 3, 4, 5\}$*, qui est donné par les arêtes* $(3, 4), (3, 5)$ *et* $(2, 3)$.



3.2 Dualité Lagrangienne

Soit le programme linéaire en nombre entier suivant :

$$(ILP)\begin{cases} z_{ILP} = \min\ c^T x & \\ \quad s.c & Ax \leq b \quad (p\ contraintes) \\ & Dx \leq d \quad (m\ contraintes) \\ & x \in \mathbb{Z}_+^n \end{cases}$$

où $A \in \mathbb{R}^{p\times n}, D \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^p, d \in \mathbb{R}^m$

l'utilisation d'une procédure de type Branch-and-Bound, nécessite la connaissance d'une borne sur la valeur de (ILP). La relaxation la plus simple a été introduite au paragraphe (1.1.2) . Nous présentons ici une des relaxation la plus utilisée en (ILP).

3.2.1 Relaxation Lagrangienne

La relaxation Lagrangienne est appliquée en générale lorsqu'on reconnaît dans la matrice des contraintes, des contraintes difficilles " $Dx \leq d$ " dont la relaxation engendrera des problèmes plus faciles à resoudre, ou bien pour lesquels on dispose d'outils efficaces de résolution.

les contraintes[3] relâchées sont réinjectées dans la fonction objectif, pondérées par les coefficients $\lambda = (\lambda_1, ..., \lambda_m)^\top \in \mathbb{R}_+^m$ appellés multiplicateurs.

Considérons le problème (ILP) et notons $X = \{x \in \mathbb{Z}_+^n \mid Ax \leq b\}$,on définit la fonction Lagrangienne comme suit :

$$L(x, \lambda) = c^T x + \lambda^T (Dx - d)$$

On définit la fonction dual :

$$(L_\lambda) \qquad \begin{cases} d(\lambda) = \min L(x, \lambda) & = \min(c^T x + \lambda^T (Dx - d)) \\ \quad x \in X \end{cases} \tag{3.2}$$

Proposition 3.1

$(i)\ \forall \lambda \in \mathbb{R}_+^m, S_{(ILP)} \subset S_{(L_\lambda)}.$

$(ii)\ v_{(L_\lambda)} \leq v_{(ILP)}$ (3.3)

Preuve. *(i) est évident.*

Si $x \in S_{(ILP)}$, $\lambda \geq 0$ et $Dx \leq d$ on aura $c^T x + \lambda^T (Dx - d) \leq c^T x$, d'où (ii). ■

[3]Lorsque les m contraintes qui sont dualisée sont des contraintes d'égalité de la forme "$Dx = b$", les multiplicateurs de Lagrange correspondant sont de signe queleconque ($\lambda \in \mathbb{R}^m$) .

3.2.2 Problème dual Lagrangien

On définit le problème dual Lagrangien comme suit :

$$(D) \quad \begin{cases} \max & d(\lambda) \\ \lambda \in \mathbb{R}^m_+ \end{cases}$$

Proposition 3.2 *Soit* $\lambda \in \mathbb{R}^m_+$ *si*
$(i)\, x_\lambda$ *est une solution optimale de* (L_λ),
$(ii)\, Dx_\lambda \leq d$ *(réalisabilité) et*
$(iii)\, (D\ x_\lambda)_i = (d)_i$ *,si* $\lambda_i > 0$ *(complémentarité)*
Alors, x_λ *est une solution optimale de* (ILP).

Preuve. par (i) $v_{(D)} \geq v_{(L_\lambda)} = c^T x_\lambda + \lambda^T (Dx_\lambda - d)$,
par (iii) $c^T x_\lambda + \lambda^T (Dx_\lambda - d) = c^T x_\lambda$,
par (ii) x_λ est réalisable pour (ILP) et par conséquent $c^T x_\lambda \geq v_{(ILP)}$

Par la relation (3.3) on aura :

$$v_{(ILP)} \ \geq v_{(D)} \geq v_{(L_\lambda)} = c^T x_\lambda + \lambda(Dx_\lambda - d) = c^T x_\lambda \geq v_{(ILP)}$$

Alors , $v_{(D)} = v_{(ILP)} = c^T x_\lambda$ et x_λ est optimal pour (ILP) ■

Théorème 3.1 *La fonction duale* $d(\lambda)$ *est une fonction concave linéaire par morceaux*

Preuve. Par définition, pour tout $\lambda \in \mathbb{R}^m_+$

$$d(\lambda) = \min_{x \in X} (c^T x + \lambda^\top (Dx - d))$$

puisque X est fini , d est le minimum d'un nombre fini de fonctions linéaires de λ.
Donc, $d(\lambda)$ est fonction concave linéaire par morceaux.([31], [32]) ■

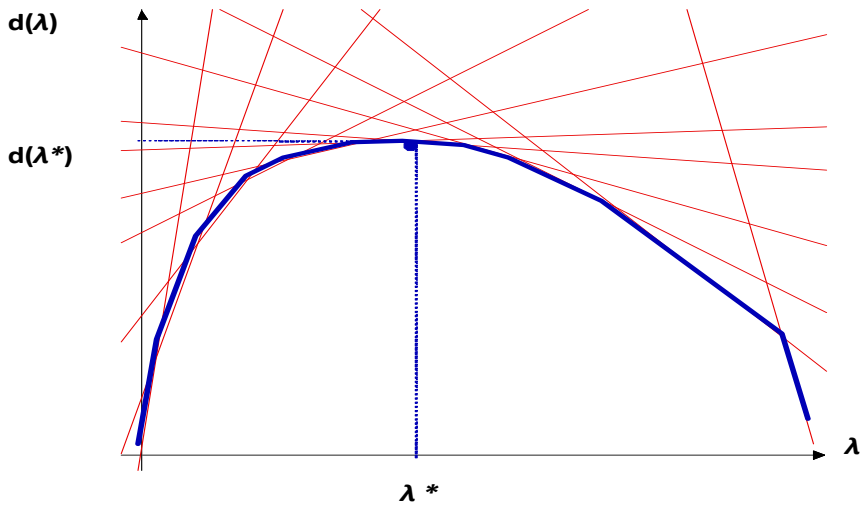


*Fig.*3.2 : $d(\lambda)$ *est concave, linéaire par morceaux*

3.2.3 Relaxation Lagrangienne pour le TSPs :

Considérons le problème TSP :

$$z = \min \sum_{e\in E} c_e x_e \tag{3.4}$$

$$s.c : \sum_{e\in\delta(i)} x_e = 2 \qquad \text{pour tout } i \in V \tag{3.5}$$

$$\sum_{e\in E(S)} x_e \leq |S| - 1 \quad 2 \leq |S| \leq |V| - 1 \tag{3.6}$$

$$x_e \in \{0, 1\} \qquad \text{pour tout } e \in E \tag{3.7}$$

La moitié des sous-tours introduit dans les contraintes (3.6) sont redondants. on a :

$$|S| - \sum_{e\in E(s)} x_e = \frac{1}{2}\sum_{i\in S}\sum_{e\in\delta(i)} x_e - \sum_{e\in E(s)} x_e = \frac{1}{2}\sum_{e\in\delta\left(S,\overline{S}\right)} x_e$$

où $\delta\left(S, \overline{S}\right)$ est l'ensemble des arrêts avec un point final dans S et autre dans $\overline{S} = V/S$, et donc comme $\delta\left(S, \overline{S}\right) = \delta\left(\overline{S}, S\right)$, $|S| - \sum_{e\in E(s)} x_e = \left|\overline{S}\right| - \sum_{e\in E(\overline{S})} x_e$ donc ,

$\sum_{e\in E(s)} x_e \leq |S| - 1$ si seulement si $\sum_{e\in E(\overline{s})} x_e \leq \left|\overline{S}\right| - 1$

en additionnant les contraintes (3.5) et divisant par 2 , nous obtenons :

$$\sum_{e\in E} x_e = n$$

Donc ,suprimons toutes les contraintes sous-tour avec $1 \in S$.Nous obtenons une relaxation Lagrangienne , on dualisons toutes les contraintes de degré sur les sommets sauf sommet 1.

$$(L_\lambda)\begin{cases} d(\lambda) = \min \sum_{e\in E}(c_e - \lambda_i - \lambda_j)x_e + 2\sum_{i\in V}\lambda_i \\ \sum_{e\in\delta(1)} x_e = 2 \\ \sum_{e\in E(s)} x_e \leq |S| - 1 \qquad 2 \leq |S| \leq |V| - 1, \quad 1 \notin S \\ \sum_{e\in E} x_e = n \\ x \in \{0, 1\}^n \end{cases}$$

les solutions réalisables de (L_λ) sont précisément les 1-arbre.

Exemple 3.2 *Reprenant l'exemple cité ci-dessus*

$$(c_e) = \begin{pmatrix} - & 30 & 26 & 50 & 40 \\ - & - & 24 & 40 & 50 \\ - & - & - & 24 & 26 \\ - & - & - & - & 30 \\ - & - & - & - & - \end{pmatrix}$$

Soit la valeur dual $\lambda = (0, 0, -15, 0, 0, 0)$,
la matrice $\overline{c}_e$ *des coûts réduit est*

$$\overline{c}_e = c_e - \lambda_i - \lambda_j = \begin{pmatrix} - & 30 & 41 & 50 & 40 \\ - & - & 39 & 40 & 50 \\ - & - & - & 39 & 41 \\ - & - & - & - & 30 \\ - & - & - & - & - \end{pmatrix}$$

La solution optimale 1-arbre est calculée en prenant les deux arrêtes à coût minimal sortant du sommet 1, i.e; l'arrête $(1, 2)$ *et* $(1, 5)$*, plus les arrêtes d' un arbre optimal des sommets* $\{2, 3, 4, 5\}$*, qui est donné par les arrêtes* $(4, 5)$*,* $(2, 3)$ *et* $(3, 4)$*.*

Il se trouve que la solution optimale 1-arbre $1-2-3-4-5-1$*, est aussi un tour, donc une solution optimale d'après* (3.1)*.*

$d(\lambda) = (30 + 39 + 39 + 30 + 40) - 2\sum \lambda_i = 178 - 30 = 148$

$d(\lambda) = 148 \leq z$ *(*z *donné par* (3.4) *) est un tour) alors est une solution optimale*

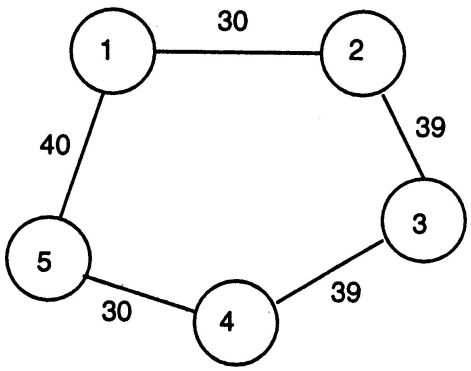


*Fig.*3.3 : *toure optimale de TSP*

3.3 Méthode sous-gradient

Définition 3.2 *(sous-gradient) :Soit* X *un convex de* $\mathbb{R}^m$ *et* $f : \mathbb{R}^m \to \mathbb{R}$ *une fonction concave*[4] *.* $\xi \in \mathbb{R}^m$ *est appelé sous-gradient de* f *au point* $x \in X$ *si*

$$f(y) \leq f(x) + \xi^\top (y - x) \qquad \forall y \in X \tag{3.8}$$

Définition 3.3 *(sous-différentiel)L'ensemble de touts les sous-gradient de f au point x notée* $\partial f(x)$

$$\partial f(x) = \left\{ \xi \ / \ f(y) \leq f(x) + \xi^\top (y - x) \quad , \forall y \in X \right\}$$

est appelé le sous-differentiel de f *au point* x *.*

Pour une étude exhauctive au calcul sous-differentiel nous renvoyons le lecteur aux ouvrages classiques ([31], [32])

[4]Pour une fonction convexe l'inégilaté dans la relation (3.8) est inversée

Théorème 3.2 *Soit x_λ une solution optimale du problème (L_λ) . Alors : $\xi = Dx_\lambda - d$ est un sous-gradient de d au point λ.*

Preuve. On a :

$$d(\lambda) = c^T x_\lambda + \lambda^\top (Dx_\lambda - d)$$

pour $\mu \in \mathbb{R}^m_+$,nous avons :

$$\begin{aligned} d(\mu) &= \min_{x\in X}(c^T x + \mu^\top (Dx - d) \\ &\leq c^T x_\lambda + \mu^\top (Dx_\lambda - d) \\ &= c^T x_\lambda + \lambda^\top (Dx_\lambda - d) + \underbrace{(\mathbf{Dx_\lambda - d})^\top}(\mu - \lambda) \\ &= \quad d(\lambda) + \xi^\top (\mu - \lambda) \end{aligned}$$

donc $\xi = Dx_\lambda - d$ est un sous-gradient de la fonction dual ■

3.3.1 Méthode sous-gradient

Le schéma standard de la méthode sous-gradient de base pour résoudre (D) est le suivant :

$$\lambda^{k+1} = P^+\left(\lambda^k + s_k \frac{\xi^k}{\|\xi^k\|}\right) \tag{3.9}$$

où ξ^k est le sous gradient de d au point λ^k , s_k est le pas , et P^+ est l'opérateur projection, de $\mathbb{R}^m$ sur $\mathbb{R}^m_+$ c-à-d :

$$P^+(\lambda) = \max(0, \lambda) = (\max(0, \lambda_1), ..., \max(0, \lambda_m))^\top$$

3.4 Algorithme sous-gradient pour le dual Lagrangien

Algorithme sous-gradient pour le dual Lagrangien

Etape 0 :(Initialisation)
choisir nimport quel $\lambda^1 \geq 0$, posons $v^1 = -\infty$, $k = 1$.

Etape 1 :
Résoudre le problème Lagrangien

$$(L_{\lambda^k}) \qquad d(\lambda^k) = \min_{x\in X} L(x, \lambda^k)$$

obtenir une solution optimale x^k ,
calculon $\xi^k = D(x^k) - d$ et $v^{k+1} = \max(v^k, d(\lambda^k))$,
si $\xi^k = 0$ arrête et λ^k est la solution optimal de(D).

Etape 2 :
calculons

$$\lambda^{k+1} = P^+\left(\lambda^k + s_k \frac{\xi^k}{\|\xi^k\|}\right)$$

avec $s_k > 0$, le pas .

Etape3 :
ponson $k := k + 1$, et aller en l'étape 1

Nous avons le lemme de base suivante de la procédure ci-dessus.

Lemme 3.1 *soit $\lambda^* \geq 0$ une solution optimal pour (D), alors pour tout k ,on a*

$$d(\lambda^*) - v^k \leq \frac{\left\|\lambda^1 - \lambda^* \right\|^2 + \sum_{i=1}^{k} s_i^2}{2\sum_{i=1}^{k} \left(\frac{s_i}{\| \xi^i \|}\right)} \tag{3.10}$$

Preuve. puisque ξ^i est un sous gradient de d à λ^i nous avons :

$$d(\lambda^*) \leq d(\lambda^i) + (\xi^i)^\top (\lambda^* - \lambda^i)$$

Ainsi,

$$\begin{aligned} \left\|\lambda^{i+1} - \lambda^* \right\|^2 &= \left\| P^+\left(\lambda^i + s_i \tfrac{\xi^i}{\|\xi^i\|}\right) - P^+(\lambda^*) \right\|^2 \\ &\leq \left\| \lambda^i + s_i \tfrac{\xi^i}{\|\xi^i\|} - \lambda^* \right\|^2 \\ &= \left\|\lambda^i - \lambda^* \right\|^2 + 2\left(\tfrac{s_i}{\| \xi^i \|}\right)(\xi^i)^\top (\lambda^i - \lambda^*) + s_i^2 \\ &\leq \left\|\lambda^i - \lambda^* \right\|^2 + 2\left(\tfrac{s_i}{\| \xi^i \|}\right)\left(d(\lambda^i) - d(\lambda^*)\right) + s_i^2 \end{aligned} \tag{3.11}$$

additionnons (3.11) pour $i = 1, ..., k$, on obtient :

$$\begin{aligned} 0 &\leq \left\|\lambda^{k+1} - \lambda^* \right\|^2 \\ &\leq \left\|\lambda^1 - \lambda^* \right\|^2 + 2\sum_{i=1}^{k} \left(\frac{s_i}{\| \xi^i \|}\right)\left[d(\lambda^i) - d(\lambda^*)\right] + \sum_{i=1}^{k} s_i^2 \end{aligned}$$

donc

$$\begin{aligned} d(\lambda^*) - v^k &= d(\lambda^*) - \max_{i=1,...,k} d(\lambda^i) \\ &\leq \frac{\sum_{i=1}^{k} \left(\frac{s_i}{\| \xi^i \|}\right)\left[d(\lambda^*) - d(\lambda^i)\right]}{\sum_{i=1}^{k} \left(\frac{s_i}{\| \xi^i \|}\right)} \\ &\leq \frac{\left\|\lambda^1 - \lambda^* \right\|^2 + \sum_{i=1}^{k} s_i^2}{2\sum_{i=1}^{k} \left(\frac{s_i}{\| \xi^i \|}\right)} \end{aligned}$$

■

3.4.1 choix des pas s_k :

Plusieurs règles différentes ont été proposées pour choisir le pas s_k pour que la méthode du sous-gradient converge.Dans la suite, nous discuterons trois règles de base ([20], [21], [30]).

Règle 1 :pour un pas $s_k = \varepsilon$ où $\varepsilon > 0$ (pas constante indépendant de k).

Règle 2 :Une règle classique garantissant la convergence théorique de l'algorithme consiste à choisir une série divergente de carré sommable pour les longueurs du pas satisfait :

$$\sum_{k=1}^{+\infty} {s_k}^2 < \infty \quad et \sum_{k=1}^{+\infty} s_k = \infty$$

Règle 3 :dans [30], polyak demontre que la suite $\left\{d(\lambda^k)\right\}$ engendrée par la relation (3.9) converge vers la valeur minimum $d(\lambda^*)$ recherchée si la suite des pas$\{s_k\}$ satisfait les conditions suivantes :

$$s_k \longrightarrow 0, k \longrightarrow \infty, et \sum_{k=1}^{+\infty} s_k = +\infty$$

Noter qu'il existe $M > 0$ tel que $\left\|\xi^k\right\| = \left\|D(x^k) - d\right\| \leq M$ pour tout k puisque $x^k \in X$ et X est un ensemble fini de nombre entier .

Théorème 3.3 (i) *Si la règle* (1) *est utilisée dans l'algorithme* (2.4) *alors*

$$\lim_{k \longrightarrow +\infty} \inf \ v^k \geq d(\lambda^*) - \frac{1}{2}\varepsilon M \tag{3.12}$$

(ii) *Si la règle* (2) *ou la règle* (3) *pour pas de choix est utilisé dans l'algorithme* (2.4) *alors*

$$\lim_{k \longrightarrow +\infty} v^k \geq d(\lambda^*) \tag{3.13}$$

Preuve. (i) En note que $\left\|\xi^i\right\| \leq M$ pour tout i , par la relation (3.10) nous avons :

$$d(\lambda^*) - v^k \leq \frac{\left\|\lambda^1 - \lambda^* \right\|^2 + \varepsilon^2 k}{2\frac{\varepsilon k}{M}} \longrightarrow \frac{1}{2}\varepsilon M \quad , (k \longrightarrow +\infty)$$

d'où (3.12)

(ii) si le pas (2) est utilisée , alors par le lemme (3.1) nous avons :

$$0 \leq d(\lambda^*) - v^k \leq \frac{\left\|\lambda^1 - \lambda^* \right\|^2 + \sum_{i=1}^{k} s_i^2}{2\sum_{i=1}^{k} \left(\frac{s_i}{M}\right)} \longrightarrow 0 \quad , (k \longrightarrow +\infty) \tag{3.14}$$

Donc (3.14) est vérifiée .

si le pas (3) est utilisée, nous réclamons que le côté droit de (3.14) converge à 0.Sinon, il doit exister $\eta > 0$ tels que :

$$\frac{\left\|\lambda^1 - \lambda^* \right\|^2 + \sum_{i=1}^{k} s_i^2}{2\sum_{i=1}^{k} \left(\frac{s_i}{M}\right)} \geq \eta \quad , \forall k,$$

ou

$$\sum_{i=1}^{k} s_i^2 - 2\left(\frac{\eta}{M}\right)\sum_{i=1}^{k} s_i \geq -\left\|\lambda^1 - \lambda^*\right\|^2 \ , \forall k. \tag{3.15}$$

Comme $s_i \to 0$ $(i \to \infty)$, il existe N_1 telle que $s_i \leq \frac{\eta}{M}$ où $i > N_1$. Donc :

$$\begin{aligned}
& \sum_{i=1}^{k} s_i^2 - 2\left(\frac{\eta}{M}\right)\sum_{i=1}^{k} s_i \\
= \ & \left(\sum_{i=1}^{N_1} s_i^2 + \sum_{i=N_1+1}^{k} s_i^2\right) - \left(\frac{\eta}{M}\right)\left(\sum_{i=1}^{k} s_i + \sum_{i=1}^{k} s_i\right) \\
\leq \ & \sum_{i=1}^{N_1} s_i^2 + \left(\frac{\eta}{M}\right)\sum_{i=N_1+1}^{k} s_i - \left(\frac{\eta}{M}\right)\left(\sum_{i=1}^{k} s_i + \sum_{i=N_1+1}^{k} s_i\right) \\
= \ & \sum_{i=1}^{N_1} s_i^2 - \left(\frac{\eta}{M}\right)\sum_{i=1}^{k} s_i \to -\infty \quad (k \to \infty)\,.
\end{aligned}$$

Ceci contredit 3.15. ■

Evidemment, un algorithme de sous-gradient basé sur la régle 3 n'offre pas un grand intérêt en pratique, car la convergence ver un point optimale est généralement excessivement lente.

C'est pourquoi nous étudierons d'autres procédés de choix des paramètres de déplacement s_k qui permettent, sous certaines conditions, d'obtenir de meilleures vitesses de convergence, et, plus précisément, la regle suivante :

Méthode de relaxation :

dans [16],Held, Wolfe et Crowder proposent d'utiliser la suite de pas définie comme suit :

$$s_k = \rho_k \frac{\overline{w} - d(\lambda^k)}{\left\| \xi^k \right\|} \ , \ 0 < \varepsilon_1 \leq \rho_k \leq 2 - \varepsilon_2 < 2 \ \ \forall k \tag{3.16}$$

où

$\overline{w}$ est une estimation[5] de la valeur optimale $v(D)$, ρ un coefficient de relaxation, $\overline{w} \geq d(\lambda^k)$ et $\xi^k \neq 0 \ \forall k$.

[5]En pratique, comme on ne connaît pas $v(D)$ (c'est précisément la valeur cherché)on devra se contenter d'une estimation $\overline{w}$ de $v(D)$.

3.4.2 Application aux TSP

Algorithme de sous gradient pour le dual Lagrangienne au problème TSP

Algorithme de sous gradient pour le dual Lagrangienne au problème TSP
Etape 0 :(Initialisation) choisir nimport quel $\lambda^1 \geq 0$, ponson $v^1 = -\infty$, $k = 1$.
Etape 1 : Résoudre le problème Lagrangien (L_{λ^k}) $\quad d(\lambda^k) = \min \sum_{e\in E}(c_e - \lambda_i^k - \lambda_j^k)x_e + 2\sum_{i\in V}\lambda_i^k$ et obtenir un x_e^k solution optimal , calculon $\xi^k = 2 - \sum_{e\in\delta(i)} x_e$ et $v^{k+1} = \max(v^k, d(\lambda^k))$, si $\xi^k = 0$ on' arrête et λ^k est la solution optimale de(D).
Etape 2 : calculons $\lambda^{k+1} = \lambda^k + s_k \frac{\xi^k}{\|\xi^k\|}$ $où\ s_k > 0\ le\ pas \qquad s_k = \rho\frac{\overline{w} - d(\lambda^k)}{\| \xi^k \|} \quad , 0 < \rho < 2$
Etape 3 : ponsons $k := k + 1$, et aller en l'étape 1.

Exemple 3.3

$$(c_e) = \begin{pmatrix} - & 30 & 26 & 50 & 40 \\ - & - & 24 & 40 & 50 \\ - & - & - & 24 & 26 \\ - & - & - & - & 30 \\ - & - & - & - & - \end{pmatrix}$$

Supposons que nous avons trouvé le tour $1-2-3-4-5-1$ *de coût* 148 .

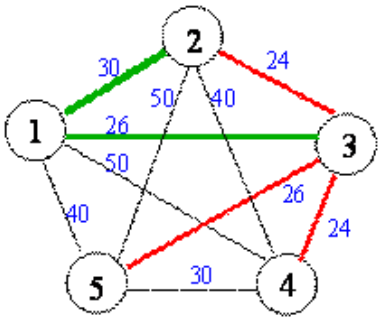
Nous utilisons $\rho = 1$ *et* $\overline{w} = 148$

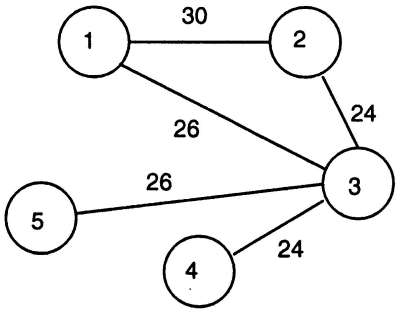
$$\begin{aligned}
\lambda^{k+1} &= \lambda^k + s_k \frac{\xi^k}{\|\xi^k\|} \\
&= \lambda^k + \rho\frac{\overline{w} - d(\lambda^k)}{\| \xi^k \|}\frac{\xi^k}{\|\xi^k\|} \\
&= \lambda^k + \frac{\overline{w} - d(\lambda^k)}{\| \xi^k \|^2}\xi^k \\
&= \lambda^k + \frac{148 - d(\lambda^k)}{\left\| \left(2 - \sum\limits_{e\in\delta(i)} x_e\right)\right\|^2}\left(2 - \sum_{e\in\delta(i)} x_e\right) \\
&= \lambda^k + \frac{148 - d(\lambda^k)}{\left\| \left(2 - \sum\limits_{e\in\delta(i)} x_e\right)\right\|^2}\left(2 - \sum_{e\in\delta(i)} x_e\right)
\end{aligned}$$

Itération 1 *:* $\lambda^1 = (0,0,0,0,0)$,

$$c_e - \lambda_i^1 - \lambda_j^1 = (c_e) \quad ,$$

$$d(\lambda^1) = 30 + 26 + 24 + 26 + 24 = 130$$





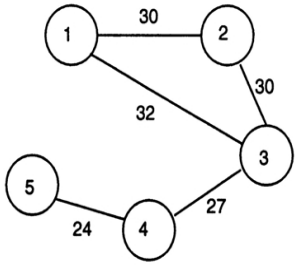
$$\left(2 - \sum_{e \in \delta(i)} x_e\right) = (0, 0, -2, 1, 1)$$

$$\begin{aligned}\lambda^2 &= \lambda^1 + \frac{148 - d(\lambda^1)}{\|(0,0,-2,1,1)\|^2}(0, 0, -2, 1, 1) \\ &= \lambda^1 + \frac{148-130}{6}(0, 0, -2, 1, 1) \\ &= (0, 0, -6, 3, 3)\end{aligned}$$

Itération 2 : $\lambda^2 = (0, 0, -6, 3, 3)$

$$\left(c_e - \lambda_i^2 - \lambda_j^2\right) = \begin{pmatrix} - & 30 & 32 & 47 & 37 \\ - & - & 30 & 37 & 47 \\ - & - & - & 27 & 29 \\ - & - & - & - & 24 \\ - & - & - & - & - \end{pmatrix}$$

$$d(\lambda^2) = 143 + \sum_{i=1}^{6} \lambda_i^2 = 143$$



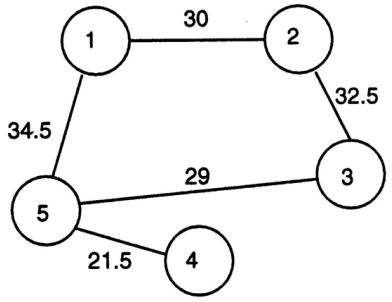
$$\left(2 - \sum_{e \in \delta(i)} x_e\right) = (0, 0, -1, 0, 1)$$

$$\begin{aligned}\lambda^3 &= \lambda^2 + \frac{148 - d(\lambda^2)}{\|(0,0,-1,0,1)\|^2}(0, 0, -1, 0, 1) \\ &= \lambda^2 + \frac{148-143}{2}(0, 0, 1, 0, -1) \\ &= (0, 0, -8.5, 3, 5.5)\end{aligned}$$

Itération 3 : $\lambda^3 = (0, 0, -8.5, 3, 5.5)$

$$\left(c_e - \lambda_i^2 - \lambda_j^2\right) = \begin{pmatrix} - & 30 & 43.5 & 47 & 34.5 \\ - & - & 32.5 & 37 & 44.5 \\ - & - & - & 29.5 & 29 \\ - & - & - & - & 21.5 \\ - & - & - & - & - \end{pmatrix}$$

$$d(\lambda^3) = 147.5 + \sum_{i=1}^{6} \lambda_i^3 = 147.5$$



$$\lceil d(\lambda^3) \rceil = \lceil 147.5 \rceil = 148$$

Chapitre 4

Recherche Tabou

4.1 Introduction

Définition 4.1 (Les mtaheuristiques) [2] :*Une métaheuristique est une stratégie générale, applicable à un grand nombre de problèmes, à partir de laquelle on peut dériver un algorithme heuristique*[1] *pour un problème particulier.*

Les métaheuristiques constituent une classe de méthodes qui fournissent des solutions de bonne qualité en temps raisonnable à des problèmes combinatoires réputés difficiles pour lesquels on ne connaît pas des méthodes calssiques plus efficaces.

Les métaheuristiques sont représentées essentiellement par les méthodes de recherche locale[3] comme le recuit simulé et la recherche tabou (voire section (4.2)), et les algorithmes évolutifs comme les algorithmes génétiques et les colonies de fourmis. On s'est limité uniquement à la recherche Tabou, pour les autres méthodes nous référons le lecteur aux ouverages et articles (([17], [14], [7], [27])).

4.2 La recherche Tabou

La recherche Tabou est un méta-heuristique originalement développée par Glover,

1986[10]. Depuis cette date, la méthode est devenue très populaire, grâce aux succès qu'elle a remportés pour résoudre de nombreux problèmes. Cette méthode combine une procédure de recherche locale avec un certain nombre de règles et de mécanismes permettant à celle-ci de surmonter l'obstacle des optimaux locaux, tout en évitant de cycler. Son principe repose sur une méthode de déplacement sur l'espace des solutions, tout en cherchant constamment à améliorer la meilleure solution courante et en conservant en mémoire la liste des précédents déplacements et ainsi guider la recherche en dehors de zones précédemment parcourues. En général, on ne va pas garder tous les déplacements (trop coûteux en

[2] Méta- : du grec « qui dépasse, englobe »

[1] Étymologiquement, le mot heuristique signifie "trouver, découvrir" en grec ancien.

[3] On appelle reherche locale celle qui converge vers un minimum local.
Les méthodes de recherche locale, appelées aussi méthodes de recherche par voisinage ,partent d'une solution initiale et, par raffinements successives, construisent des suites de solutions de coûts décroissants pour un problème de minimisation.

mémoire), mais on va seulement empêcher l'accès à certaines solutions pendant un certain nombre d'itérations. La méthode consiste à se déplacer d'une solution vers une autre par observation du voisinage de la solution de départ et à définir des transformations tabous que l'on garde en mémoire. Une transformation Tabou est une transformation que l'on s'interdit d'appliquer à la solution courante.

4.2.1 Description de l'algorithme de la recherche Tabou

On résume la méthode Tabou comme suit :
Soient,
- x^0, la solution initiale.
- $x \rightarrow f(x)$, la fonction à minimiser.
- x^*, la meilleure solution jusqu'à présent.
- $F^* = f(x^*)$, la valeur de la fonction objectif de la meilleure solution trouvée.

Une fois ces définitions fixées, il suffit de suivre le schéma de cet algorithme :

Algorithme Tabou

(**a**) x^0 soit la solution de départ choisie.
$k \longleftarrow 0$.
$L \longleftarrow \emptyset$; (initialisation de la liste Tboue)
$x^* \longleftarrow x^0$
$S_t \longleftarrow S_0$

(**b**) **RÉPÉTER** (itération courante k)
Déterminer x^{k+1} tel que ;
$f\left(x^{k+1}\right) = \underset{x \in V_L\left(x^k\right)}{Min} \{f(x)\}$
($V_L\left(x^k\right)$ désigne l'ensemble des solutions voisines de x^k après élimination des solutions interdites telles que spécifiées par la liste Taboue L) ;
Si ($f(x^{k+1}) < F^*$) alors :
$x^* \longleftarrow x^{k+1}$;
Fin Si
Si ($|L| = T$) alors :
Éliminer de L l'information la plus ancienne.
Fin Si
Ajouter dans L l'nformation concernant la nouvelle solution x^{k+1}, ou la transition entre x^k et x^{k+1} ;
$k \longleftarrow k + 1$;
TANT QUE (condition d'arrêt non vérifiée) .

(**c**) La solution fournie par l'algorithme est x^*, de coût $f(x^*)$.

4.2.2 Critère d'arrêt

Des critères d'arrêt possibles sont :

- Si une solution prouvée optimale a été trouvée.
- Un nombre maximal prédéfini à été atteint ou un temps à ne pas dépasser.
- Si la recherche semble stagnée sans amélioration de la meilleure solution trouvée.

4.2.3 Liste Taboue

La liste Taboue est un concept important dans la recherche Tabou, elle mis à jour les nouvelles recherche tout évitant de garder tous les solutions précédentes. Sur un grand nombre d'itérations, cette liste Tabou pourrait économiser le temps de calcul et augmenter ainsi l'efficacité de la recherche de manière significative.

La liste Taboue est généralement gérée comme une liste "circulaire" : on élimine à chaque itération la solution Tabou la plus ancienne, en la remplaçant par la nouvelle solution retenue. Mais le codage d'une telle liste est encombrant, car il faudrait garder en mémoire tous les éléments qui définissent une solution. Pour palier à cette contrainte, on remplace la liste Taboue des solutions interdites par une liste de "transformations interdites", en interdisant la transformation inverse d'une transformation faite récemment.

Nous allons illustrer le concept de la liste Taboue par un exemple.

Exemple 4.1 *Supposons que nous voulons passer par chaque itinéraire (linéaire) une seule fois, sans répétition (voir Fig.*4.1*).*

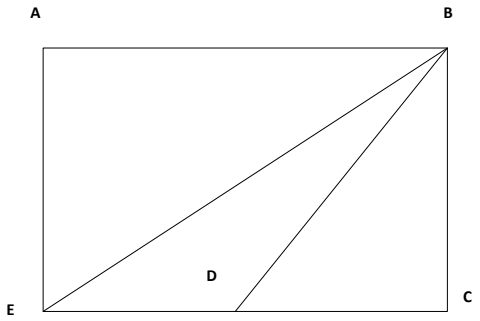


*fig.*4.1 : *Passant à travers chaque ligne une seule fois.*

Si nous commençons le déplacement à partir par exemple, du point E,nous allons d'abord aller à trvers EA et atteindre le point A. présent ,on met la route EA dans la liste Taboue (noté L) :

$$L = \{EA\}$$

La prochaine route ne peut être que AB, car EA est déjà dans L, revenir n'est pas autorisé .Nous actualisons la liste L :

$$L = \{EA, AB\}$$

Au point B, il ya quatre routes possibles mais seulement trois (BE, BD, BC) sont autorisées parce que la route AB est maintenant dans L. Nous voyons que la taille de L est

croissante . Supposons que nous choissions au hasard la route BD pour aller au point D. La liste Taboue maintenant devient :

$$L = \{EA, AB, BD\}$$

Maintenant du point D, on peut passer par DC ou par DE, mais nous ne pouvons pas revenir en arrière sur BD. Donc, nous choisissons aléatoirement DE pour atteindre le point E alors la liste Tabou devient :

$$L = \{EA, AB, BD, DE\}$$

Au point E, nous avons trois routes possibles, mais seulement EB est autorisée, les deux autres routes sont interdites. Nous avons donc à choisir la route EB pour revenir au point B, et la liste s'actualise est devient :

$$L = \{EA, AB, BD, DE, EB\}$$

Encore une fois au point B, il y a quatre possibilités, mais seulement BC est admissible , les trois autres routes sont dans L et donc ne sont pas admises. Nous devons donc aller à travers BC pour atteindre le point C. La liste L actualisée est :

$$L = \{EA, AB, BD, DE, EB, BC\}$$

Maintenant, la seule route autorisée est CD pour aller au point D. Ainsi nous complétons la tâche, et la liste L devient :

$$L = \{EA, AB, BD, DE, EB, BC, CD\}$$

Ici nous avons utilisé toute l'histoire de la liste Tabou.Nous pouvons utiliser totalement cette histoire si la mémoire de l'ordinateur ne pose pas de problème. Souvent, nous utilisons uniquement une partie de l'histoire comme une liste tabou permanente , habituellement avec une longueur fixe. Si nous utilisons une longueur fixe $n = 5$ dans notre exemple, puis les deux dernières étapes, la liste tabou doit être remplacé par :

$$L = \{AB, BD, DE, EB, BC\}$$

et

$$L = \{BD, DE, EB, BC, CD\}$$

L'avantage d'une taille fixe pour la liste Taboue est d'economiser la mémoire et faciliter l'implémentation, elle a l'inconvénient d'une information incomplète. L'utilisation effective de la mémoire à la recherche Taboue est toujours un domaine ouvert de recherches actives.

4.3 Une application de la recherche Tabou au problème TSP

Appliquons cette méthode au proplème TSP.Les étapes de base sont présentées ci-dessous.

4.3.1 Une représentation de la solution :

Une solution réalisable est représentée comme une séquence de nœuds, codé par un tableau des entiers qui représente le successeur et le prédécesseur de chaque ville.

6	2	4	1	5	8	3	7

Tab (4.1)

Ainsi,le tableau (*Tab* (4.1)) représente le circuit suivant :
le point de départ est la ville 6 qui est aussi la ville d'arrivée, en passant par les villes selon leur ordre d'apparition dans le tableau : $6 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow 8 \rightarrow 3 \rightarrow 7 \rightarrow 6$.

4.3.2 Solution initiale :

Une bonne solution possible mais non optimale pour le TSP peut être trouvée rapidement en utilisant une approche heuristique. En commençant par le premier nœud dans la tournée, et trouver le nœud le plus proche. Chaque fois trouver le nœud le plus proche non visité à partir du noeud courant jusqu'à ce que tous les nœuds soient visités.

4.3.3 Voisinage :

Un voisinage pour une solution donnée est définie comme toute autre solution qui est obtenue par un échange par paire de deux nœuds de la solution. Ceci garantit toujours que tout voisinage d'une solution réalisable est toujours une solution réalisable (c-à-d, ne forme pas de sous-tour). Si nous fixons le nœud 1 comme point de départ et la fin de nœud, pour un problème de N nœuds, il ya $C\,^{2}_{N-1}$ vosinages . A chaque itération, le voisinage est sélectionné par rapport à une fonction objective (distance minimale) . nous avons l'exemple ci-dessous :

Exemple 4.2

1	**2**	3	4	**5**	6	7	8

$\rightarrow$

1	**5**	3	4	**2**	6	7	8

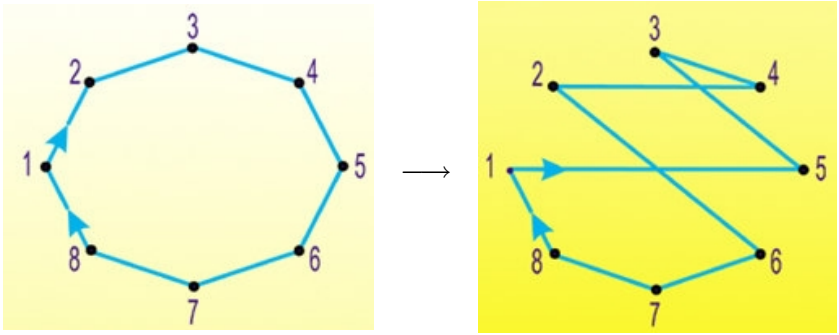


Figure (4.2) : *Solution de voisinage obtenue en échangeant les ordre de visite des villes* 2 *et* 5

4.3.4 Liste Taboue :

Pour empêcher le processus de cyclage dans un petit ensemble de solutions, certaines solutions récemment visitées sont stockées dans une liste Taboue, ce qui empêche leur survenue pendant une période limitée.

Pour notre problème, la solution utilisée est une paire de nœuds qui ont été échangé récemment. Une structure de Taboue stocke le nombre d'itérations pour lequel une paire de nœuds est interdite d'échange.

Taille de la liste Tabou :

La taille de la liste Taboue est à déterminer empiriquement, elle varie avec les problèmes, mais c'est une donnée primordiale. En effet une liste trop petite peut conduire à un cycle , alors qu'une liste trop grande peut interdire des transformations intéressantes.

La composante de la mémoire principale (mémoire à court terme) peut être stockée dans une matrice où l'échange des villes i et j est enregistré dans le $i^{ème}$ ligne et $j^{ème}$ colonne.

4.3.5 Critère d'aspiration

Afin de ne pas manquer de très bonnes solutions, il est courant d'introduire un critère dit d'aspiration. Le critère utilisé pour cela dans le problème actuel du TSP est la permission d'échange de nœuds, même si elle est Taboue, s'il en résulte une solution ayant une valeur objective meilleure que celle qui existe déjà.

4.3.6 Critère d'arrêt :

L'algorithme se termine, si pour un certain nombre prédéfini d'itérations il n y a aucune amélioration dans la solution.

Exemple 4.3 *Nous utilisons un exemple de cinq villes avec.*

$$C = \begin{pmatrix} - & 5.39 & 10 & 14.14 & 10 \\ 5.39 & - & 5.39 & 9.43 & 9.43 \\ 10 & 5.39 & - & 10 & 14.14 \\ 14.14 & 9.43 & 10 & - & 10 \\ 10 & 9.43 & 14.14 & 10 & - \end{pmatrix}$$

Solution initiale :

Supposons que nous partons d'un parcours aléatoire $2-5-3-1-4-2$

Voisinage :

Afin de permuter deux villes quelconques parmi ces cinq villes, nous utilisons les echanges d'indices suivantes pour permuter deux villes adjacentes :

$$MIE = \begin{pmatrix} 2 & 1 & 3 & 4 & 5 \\ 1 & 3 & 2 & 4 & 5 \\ 1 & 2 & 4 & 3 & 5 \\ 1 & 2 & 3 & 5 & 4 \\ 5 & 2 & 3 & 4 & 1 \end{pmatrix}$$

Où la première ligne, échange les deux premières villes et la cinquième ligne échange la première et la dernière (cinquiéme)villes.

Liste Taboue :

Afin d'éviter la répétition d'échanges déjà réalisés, nous utilisons une liste Taboue pour la matrice d"indice d'échange ci-dessus :

$$L = (0 \quad 0 \quad 0 \quad 0 \quad 0)$$

où le premier élément correspond à la ligne de la matrice d'echange MIE. Maintenant, nous utilisons également une mémoire de longueur fixe, par exemple, $m = 2$ ce qui signifie, nous enregistrons deux étapes dans le passé récent .

itération 1:

De la route initiale :

$$route = \boxed{2}\boxed{5}\boxed{3}\boxed{1}\boxed{4}$$

avec une distance initiale $d = 57.14$, essayons, la nouvelle route en permutant les deux premières villes ou $route = route[MIE(1)]$. On aura la route :

$$route = \boxed{5}\boxed{2}\boxed{3}\boxed{1}\boxed{4}$$

dont la distance totale est 48.96 *donc :*

$$\mathbf{d} = \min(\mathbf{57.14}, \mathbf{48.96}) = \mathbf{48.96}.$$

Nous mettons à jour la liste Taboue, mais il vaut mieux l'actualiser lorsque nous trouvons une meilleur solution après les cinq échanges dans la matrice MEI, la nouvelle route est prise en permutant la seconde paire $route = route[MEI(2)]$ ona :

$$route = \boxed{2}\boxed{3}\boxed{5}\boxed{1}\boxed{4}$$

dont la distance totale est 53.10 *qui est plus grande que d, cette dernière tentative est écartée. d vaut toujours $d = 48.96$. Le prochain échange nous donné :*

$$route = \boxed{2}\boxed{5}\boxed{1}\boxed{3}\boxed{4}$$

avec une distance 48.87 *qui est plus courte que $d = 48.96$,d est mise à jour avec $d = 48.87$, l'échenge suivant donne :*

$$route = \boxed{2}\boxed{5}\boxed{3}\boxed{4}\boxed{1}$$

avec une distance 53.10*, donc nous l'ecartons. Le cinquième échange donne :*

$$route = \boxed{4}\boxed{5}\boxed{3}\boxed{1}\boxed{2}$$

avec une distance 48.96 *qui est plus grande que $d = 48.87$, ainsi, nous l'ecartons également. Après avoir parcouru les cinq permutation possible , la meilleur solution est :*

$$route = \boxed{2}\boxed{5}\boxed{1}\boxed{3}\boxed{4}$$

avec $d = 48.87$.Il est actuellement nécessaire de mettre à jour la liste Taboue :

$$L = (0\ 0\ m\ 0\ 0) = (0\ 0\ 2\ 0\ 0)$$

ceci signifie que le troisième échange a été enregistré .

itération 2 :

A présent, nous répétons la même procédure c-à-d les cinq échange, mais en prenant comme route initiale la route :

$$route = \boxed{2}\boxed{5}\boxed{1}\boxed{3}\boxed{4}$$

Le premier échange mène à

$$route = \boxed{5}\boxed{2}\boxed{1}\boxed{3}\boxed{4}$$

avec une distance 44.82 *donc :*

$$\mathbf{d} = \min(\mathbf{48.87}, \mathbf{44.82}) = \mathbf{44.82}.$$

Pour le deuxième échange, nous avons donné :

$$route = \boxed{2}\boxed{1}\boxed{5}\boxed{3}\boxed{4}$$

avec une distance 48.96 *ainsi nous la rejetons. Comme la prochaine entrée dans la liste Taboue est* 2*, nous n'effectuerons pas cet échange. Le prochain échange mène à*

$$route = \boxed{2}\boxed{5}\boxed{1}\boxed{4}\boxed{3}$$

avec une distance 48.96 *que nous l'écartons. Enfin, le cinquième échangne donne :*

$$route = \boxed{4}\boxed{5}\boxed{1}\boxed{3}\boxed{2}$$

avec une distance 44.82 *.qui est la même que le premier échange, donc il n'est pas nécessaire de mettre à jour d. Après les quatre échange permis, la meilleur solution actuelle est :*

$$route = \boxed{5}\boxed{2}\boxed{1}\boxed{3}\boxed{4}$$

et la nouvelle liste Taboue est $L = (2\ 0\ 1\ 0\ 0)$*, où le troisième élément est réduit de* 2 *à* 1*, ce qui signifie que nous pouvons utiliser le troisième échange plus tard lorsque cette entrée devient nulle.*

itération 3 :

Commençant encore une fois à partir de la meilleure solution actuelle :

$$route = \boxed{5}\boxed{2}\boxed{1}\boxed{3}\boxed{4}$$

après deux générations d'échange, le premier échange n'est pas autorisé parce que la première entrée dans la liste Taboue est de 2*, donc nous commençons par le deuxième échange qui mène à la route :*

$$route = \boxed{5}\boxed{1}\boxed{2}\boxed{3}\boxed{4}$$

qui a une distance totale $d = 40.78$*, ce qui est la route optimale globale pour ces cinq villes. Le prochain échange mène à la route*

$$route = \boxed{5}\boxed{2}\boxed{1}\boxed{4}\boxed{3}$$

avec une distance de 53.10.

La poursuite de la procédure ne donne pas d'améliorations dans la qualité de la solution. Comme nous ne savons pas si la solution actuelle est optimal globale ou non, nous allons continuer la recherche jusqu'à un nombre prescrit d'itérations ou après un certain nombre d'itérations sans amélioration de la solution. Dans le cas présent, la solution optimale globale est la route :

$$route = \begin{array}{|c|c|c|c|c|} \hline 5 & 1 & 2 & 3 & 4 \\ \hline \end{array}$$

Nous avons vu que la recherche Taboue prend seulement environ 10 marches pour atteindre la route optimale globale, et cela est beaucoup plus efficace en comparaison avec les 120 étapes par une enumération complète . En outre, la liste Taboue a évité au moins 3 répétitions possibles, et économise environ $\frac{1}{3}$ des échanges possibles. En général, pour des problèmes avec grand nombre n, le temps de calcul sauvegardé en utilisant la recherche Tabou sera plus important.

Chapitre 5

Conclusion et perspectives

Ceci n'est qu'un échantillon de méthodes abordées en littératures pour résoudre le TSP, dans le cadre des méthodes méta-heuristiques, les algorithmes génétiques ont la part du lion, nous avons omis d'en parler dans ce mémoire vue leurs spécificités et ses techniques appropriées, c'est un domaine vaste qui mériterait à lui seul un ouvrage. L'étude réalisée nous permet d'envisager deux axes de recherche :

5.1 Problème de tourné de véhicule

C'est du TSP généralisé. les véhicules sont situés dans un dépôt et ils sont requis pour visiter des clients dispersés afin de satisfaire une clientèle connue. Le problème apparaît dans un grand nombre de situations pratiques et il est connu sous le nom du problème de tournées de véhicules (VRP). Les clients sont indexés par $i = 2, ..., n$ et $i = 1$ se réfère au dépôt. Les véhicules sont indexés par $k = 1, ..., m$. Un client i a une demande q_i, Les frais de déplacement entre le client i et j est c_{ij}. La capacité du véhicule k est Q_k. On suppose que tous les clients et les véhicules sont classés dans l'ordre décroissant de q_i et Q_k respectivement. Le VRP de base est d'acheminer les véhicules (une voie par véhicule, le départ et l'arrivé se passe au dépôt), de sorte que tous les clients sont fournis avec leurs exigences et le coût total des déplacements est réduit au minimum. La Figure ci-dessous montre la forme d'une solution à un VRP.

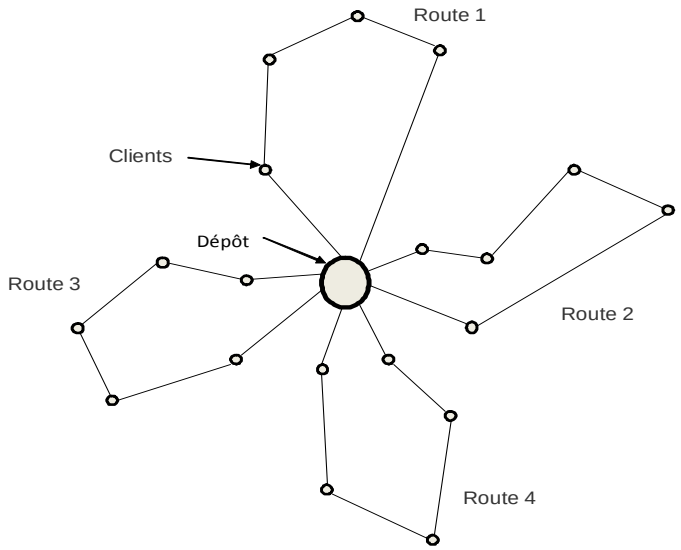


Fig.(5.1) : *forme de solution au VRP de base*

Formulation

$$x_{ijk} = \begin{cases} 1 \text{ Si le véhicule } k \text{ visite le client } j \text{ immédiatement après le client } i \\ 0 \text{ autrement} \end{cases}$$

$$y_{ik} = \begin{cases} 1 \text{ Si le client } i \text{ est visité par le véhcule } k \\ 0 \text{ autrement} \end{cases}$$

Le VRP de base est donc de minimiser

$$\sum_{ij} c_{ij} \sum_{ijk} x_{ijk} \tag{5.1}$$

par rapport aux contraintes

$$\sum_{k} y_{ik} = \begin{cases} 1, & i = 2, ..., n \\ m, & i = 1 \end{cases} \tag{5.2}$$

$$\sum_{i} q_i y_{ij} \leq Q_k \quad k = 1, ..., m \tag{5.3}$$

$$\sum_{j} x_{ijk} = \sum_{j} x_{jik} = \sum y_{ik} \quad \text{pour } i = 1, ..., n \text{ et } k = 1, ..., m \tag{5.4}$$

$$\sum_{i,j \in S} x_{ijk} \leq |S| - 1, \quad \text{pour tout } S \subseteq \{2, ..., n\}, \; k = 1, ..., m \tag{5.5}$$

$$y_{ik} \in \{0, 1\}, \; i = 1, ..., n \text{ et } k = 1, ..., m \tag{5.6}$$

$$x_{ijk} \in \{0, 1\}, \; i, j = 1, ..., n \text{ et } k = 1, ..., m \tag{5.7}$$

Les contraintes (5.2) assurent que chaque client est attribué à un certains véhicule (sauf pour le dépôt qui est visité par tous les véhicules), les contraintes (5.3) sont les contraintes de capacité des véhicules, les contraintes (5.4) assurent que le véhicule qui se rend à un client quitte également ce client, et les contraintes (5.5) sont les habituelles contraintes d'élimination de sous-tours pour le TSP.

5.2 Méthodes sous-gradient à valeur objectif variable

L'algorithme présenté au chapitre III, s'appui sur la méthode sous-gradient, depuis les tests numérique réaliser par Held, Wolfe et Crowder ([16]), il s'est avéré que la règle 2 et 3, données au paragraphe (3.4.1), ne sont pas nécessaires pour assurer la convergence de l'algorithme sous-gradient. Seul un choix du pas du type

$$s_k = \rho \frac{\overline{w} - d\left(\lambda^k\right)}{\left\|\xi^k\right\|} \quad \text{où } 0 < \rho < 2$$

peut assurer, sous certaine hypothèse une convergence même géométrique. Cependant, les travaux réaliser par Polyak montrent que généralement la suite générée par la méthode converge en valeur vers l'estimation $\overline{w}$. Ces dernières années, des nouvelles techniques ont été utilisées pour éviter ce problème, ces techniques s'appuient essentiellement sur l'idée de remplacer l'estimation $\overline{w}$ par une suite w_k, qui sera mise à jour au fur et à mesure le déroulement de l'algorithme ; tout en assurant la convergence de la suite $(w_k)_k$ vers la valeur optimale $d(\lambda^*)$. Parmi les approches étudiées pour la construction de la suite $(w_k)_k$, notons ici les travaux réalisés par Kim et al ([21]), U. Bra¨nnlund ([3]), J.L. Goffin et K.C. Kiwiel ([13]) et CHURLZU LIM et HANIF D. SHERALI ([4]). Aussi, Fumero ([9]) a déjà appliqué ces techniques pour le TSP.

Bibliographie

[1] **Alexander Shrijver** :Theory of Linear and Integer, 1986. Programming.Wiley-interscience series in discrete mathematics and optitimization

[2] **Mokhtar .S. Bazaraa, Hanif D.Sherali, C.M.Shetty** :Nonlinear programming. Theory and algorithms. Third Edition. A Wiley-lnterscience Publication.

[3] **U. Bra¨nnlund**, "On relaxation methods for nonsmooth convex optimization," Ph.D Dissertation, Depart-ment of Mathematics, Royal Institute of Technology, Stockholm, Sweden, 1993.

[4] **CHURLZU LIM and, HANIF D. SHERALI**, Convergence and Computational Analyses for Some Variable Target Value and Subgradient Deflection Methods, Computational Optimization and Applications, 34, 409–428, 2005.

[5] **George B.Dantzig et Mukund N. Thapa** :Linear Programming,1 : Introduction.Springer Series in operations Research.

[6] **Duan Li et Xiaoling Sun** : Nonlinear integer programming . 2006 by Springer Science+Business Media, LLC

[7] **Edmund K. Burke, Graham Kendall** : Search Methodologies,Introductory Tutorials in Optimization and Decision Support Techniques.2005 by Springer Science-i-Business Media, LLC.

[8] **Edwin. K.P. Chong,Stanislaw H. Żak** : An Introduction to Optimization,Second Edition.A Wiley-lnterscience Publication.

[9] **F. Fumero**, "A modified subgradient algorithm for Lagrangean relaxation," Computers & Operations Research, vol. 28, no. 1, pp. 33–52, 2001.

[10] **F. Glover**, Future paths for integer programming and links to articial intelligence, Comput. Oper. Res. 13, 533-549, 1986

[11] **Glover, F**. (1989) "Tabu Search — Part I," ORSA Journal on Computing, Vol. 1, No. 3, pp. 190-206. First comprehensive description of tabu search)

[12] **Glover, F**. (1990) "Tabu Search — Part II," ORSA Journal on Computing, Vol. 2, No. 1, pp. 4-32. The second part of this comprehensive description of tabu search introduces additional mechanisms such as the reverse elimination method

[13] **J.L. Goffin and K.C. Kiwiel**, "Convergence of a simple subgradient level method," Mathematical Programming, vol. 85, no. 1, pp. 207–211, 1999.

[14] **D. E. Goldberg**. Genetic algorithms in search, optimization and machine learning. Addison Wesley,1989

[15] **P.L. Hammer, E.L.Johnson, B.H.Korte and G.L.Nemhauser** : Annals of Discrete Mathematics 1 :Studies in Integer Programming. North-holland publishing company- 1977

[16] **Held M., Wolfe P., Crouwder H.P.** (1974), Validation of Subgradient Optimization' Mathematical Programming 6, 62-88.

[17] **J. H. Holland et. Al., "Induction** : Processes of Inference, Learning, and Discovery", MIT Press, 1989.

[18] **Garey M.R. et Johnson D.S.** : Computer and intractabiliy : a guide to the theory of NP-completeness. Freeman (1989).

[19] **M. J¨unger,T. Liebling ,D. Naddef , G. Nemhauser ,W. Pulleyblank,G. Reinelt, G.Rinaldi, L.Wolsey** : 50 Years of Integer Programming 1958–2008.Springer-Verlag Berlin Heidelberg 2010.

[20] **S. Kim and H. Ahn.** Convergence of a generalized subgradient method for nondifferentiable convex optimization. Mathematical Programming, 50 :75-80, 1991.

[21] **S. Kim, H. Ahn, and S-C Cho**. Variable target value subgradient method. Mathematical Programming, 49 :359-369, 1991.

[22] **H.H.Kuhn**,the Hungarian Methode for the Assignment Problem,Naval Research Logistics Quarterly 2 (955) 83-97.

[23] **A. H. Land et A. G. Doig,** "An automatic method for solving discrete programming problems", Econometrica, vol. 28, 1960, pages 497-520.

[24] **Gilbert Laporte** : « The traveling salesman problem : an overview of exact and approximate algorithms ». European Journal of Operational Research vol 59 (1992).

[25] **Lawler E.L., Lenstra J.K., Rinnooy kan A.H.G. et Shmoys D.B(Eds)** : The travelling salesman problem :a guided tour of combinatorial optimization .Wiley (1985).

[26] **Claude Lemarechal et Robert Mifflin :** Nonsmooth Optimization. Proceedings of a IIASA Workshop, March 28 - April 8, 1977.Ilasa proceedings series.Volume 3

[27] **Michel Minoux** :Programmation mathématique ,théoie et algorithmes tome 2(2 édition)

[28] **Garfinkel R.S. et Nemhauser G.L.** : integer programing .wiley(1972).

[29] **C.H. Papadimitriou et K. Steiglitz** :Combinatorial Optimization : Algorithms and Complexity ,Dover publucation, INC. Mineola. New york.

[30] **k. MinB. T Polyak**, optimization of nonsmooth functions. USSR Computational Mathematics and Mathematical Physics, 9 :14—29, 1969.

[31] **R.T.Rockafellar** (1970), Convex analysis, Princeton Univesity Press, Princeton.

[32] **R.T.Rockafellar** (1981), The Theory of Subgradients and its Applications to Problem of Optimization. Convex and Nonconvex Function, Helderman Verlag, Berlin.

[33] **Roy B.** :Algèbre moderne et théorie des graphes orientées vers les sciences économiques et sociales ,tomes 1 et 2 , Dunod ,Paris (1969-1970).

[34] **M.Sakarovitch**, 1984, Optimisation Combinatoire, Méthodes Mathématiques et Algorithmiques, Hermann, Paris.

[35] **Jacques Teghem** : Programmation linéaire. Éditions de l'Université de Bruxelles (2003).

[36] **Robert J. Vanderbei** : Linear programming, Foundations and Extensions.Third Edition. Springer

[37] **G.L. Nemhauser et L.A. Wolsey**. Integer and Combinatorial Optimization.Wiley, New York, 1998.

[38] **Xin-She Yang** ,Introduction to Mathematical Optimization -From Linear Programming to Metaheuristics ,Cambridge International Science Publishing (2008)

[39] **Yurii Nesterov** : Introductory lecteures on convex optimization. ABasic Course.Springer