UNIVERSITE SAAD DAHLAB DE BLIDA 1
FACULTE DES SCIENCES
DEPARTEMENT DE MATHEMATIQUES

Mémoire

En vue de l'obtention du diplôme de

Master en Mathématiques

Option :

Recherche Opérationnelle

Thème

Sur le problème du voyageur de commerce

Présenté par :
M^lle^ Hamidi Meriem

Devant le jury :

Président :	**Tami Omar**	**MAA à Blida**
Examinateur :	**Frihi Redhouane**	**MAA à Blida**
Promoteur :	**Meddah Nacera**	**MCB à Blida**

2016-2017

Remerciements

Tout d'abord, je remercie le Dieu, notre créateur de m'avoir donné les forces, la volonté et le courage pour accomplir ce modeste travail

Je remercie Mme N. Meddah pour avoir acceptée de me diriger dans ce mémoire.

Je tiens également à remercier messieurs les membres de jury pour l'honneur qu'ils me font en acceptant d'évaluer ce travail. Je vous remercie pour l'intérêt que vous avez porté ce travail et pour vos précieux conseils et remarques.

En fin, mes vifs remerciements à l'ensemble des enseignants qui ont contribué à ma formation.

Résumé

Le problème de voyageur de commerce consiste à chercher un circuit hamiltonien dans un réseau dont le coût est le minimum possible. Il est bien connu que c'est un problème NP-difficile qui reste ouvert à la recherche.

Le but de ce travail est de présenter et d'implémenter l'algorithme de séparation et évaluation de Little et al. pour résoudre le problème de voyageur de commerce pour cas d'un graphe complet. Ainsi un logiciel est proposé dans ce sens.

Abstracts

The traveling salesman problem is to find a Hamiltonian circuit in a network whose cost is the minimum possible. It is well known that this is an NP-difficult problem that remains open to research.

The aim of this work is to present and implement the algorithm of brunch and bound of Little et al. to solve the traveling salesman problem when the graph is complete. So a program has been proposed in this direction.

الملخص

مشكلة المسافر التجاري هو العثور على دائرة هاملتون في شبكة تكلفتها الأدنى حد ممكن. ومن المعروف جيدا أن هذه مشكلة من مجموعة المشكلات الصعبة.

والهدف من هذا العمل هو تقديم وتنفيذ خوارزمية الفصل وتقييم ليتل وآخرون. من أجل حل مشكلة المسافر التجاري لحالة المخطط الكامل. لذلك فقد اقترحنا برنامج لحله.

Table de matière

Liste des figures

Liste des algorithmes

Introduction

Introduction

Dans la vie quotidienne, l'être humain est souvent confronté à des problèmes dans des domaines différents y compris le transport, la médecine, l'économie, l'industrie, l'énergie, l'éducation, la télécommunication…etc. La majorité de ces problèmes sont des problèmes d'optimisation non-linéaire, linéaire ou linéaire en nombre entier. L'étude de ces problèmes impose deux propriétés importantes l'optimalité et la rapidité. Donc pour dire que le problème a une solution il faut que l'algorithme (la méthode) trouvé(e) respecte ces deux critères.

Nous nous intéressons dans ce mémoire aux problèmes linéaires en nombres entiers. Ces problème sont NP-difficile et donc il n'existe pas d'algorithmes polynomiaux pour les résoudre. Vu l'importance de ces problèmes, les chercheurs ont fait de nombreuses études pour établir des algorithmes qui aident à la résolution de ces problèmes. Ces algorithmes sont regroupés en deux catégories :

- Les algorithmes qui donnent la solution optimale au problème mais en temps exponentiel.
- Les algorithmes qui donnent une solution approchée en un temps polynomial.

Parmi les problèmes d'optimisation en nombre entier, on trouve le problème du voyageur de commerce (PVC). Le PVC consiste à la recherche d'un trajet minimum permettant à un voyageur de visiter n villes séparées par des distances données en passant par chaque ville exactement une seule fois. Il commence par une ville quelconque et il termine en retournant à la ville de départ. Quel parcours faut-il choisir afin de minimiser la distance parcourue ?

La notion de distance peut-être remplacée par d'autres notions comme le temps qu'il met ou l'argent qu'il dépense. En général, on cherche à minimiser un coût.

Différents chercheurs ont étudié le problème du voyageur de commerce et ont donné des algorithmes (exacts et approchés) pour le résoudre. Dans ce mémoire, on s'intéresse à l'algorithme élaboré par J.D.C. Little, K.G. Murty, D.W. Sweeney et C. Karel en 1963 qui est un algorithme exact de type séparation et évaluation.

Le manuscrit est réparti en quatre chapitres :

Dans le premier chapitre, on rappelle les définitions des notions de base sur les graphes nécessaires à la compréhension de ce manuscrit. Ainsi nous donnons la représentation matricielle d'un graphe d'une manière générale. Puis on donne quelques notions sur la complexité algorithmique.

Dans le chapitre 2, on expose le problème du voyageur de commerce : définitions, historiques, variantes, utilisation et à la fin on donne la modélisation de ce problème sous forme d'un problème linéaire en nombre entier.

Au chapitre 3, on expose les méthodes de résolution des problèmes NP-difficiles classées en trois catégories, la méthode de séparation et évaluation comme un exemple, les heuristiques et en fin les métaheuristiques avec la méthode de colonie de fourmis à titre d'exemple.

Le chapitre quatre est consacré à la définition de l'algorithme de Little en donnant son principe et en exprimant les procédures de séparation et évaluation ainsi que le pseudo code de l'algorithme avec un exemple d'application. Puis on expose l'implémentation de l'algorithme avec une discussion.

Chapitre 1

Chapitre 1

Concepts fondamentaux des graphes et complexité algorithmique

Dans ce chapitre, nous donnons quelques définitions et notations de base de théorie des graphes qui sont nécessaires pour introduire le problème de voyageur de commerce. Pour plus de détails concernant la théorie des graphes, nous invitons le lecteur à consulter les ouvrages de C. Berge [1] et de Chartrand et Lesniak [4] Ainsi que de M. Gondran et M. Minoux [7].

1.1. Définitions et notations de base

Graphe

Un *graphe* $G = (X, U)$ est déterminé par deux ensembles :

- $X = \{x_1, x_2, \dots, x_n\}$ dont les éléments sont appelés *sommets* ou parfois des *nœuds,* et
- $U = \{u_1, u_2, \dots, u_m\}$ du produit cartésien $X \times X$ dont les éléments sont des couples de sommets de type (x_i, x_j) avec $i, j \in \{1, \dots, n\}$.

Un graphe peut être orienté ou non

- Dans *un graphe orienté*, les couples $(x_i, x_j) \in U$ sont orientés c'est-à-dire que (x_i, x_j) est un couple ordonné, où x_i est *l'extrémité initiale* et x_j est *l'extrémité terminale*. Un couple (x_i, x_j) est appelé un *arc*, et est représenté graphiquement par $x_i \rightarrow x_j$.
- Dans *un graphe non orienté*, les couples $(x_i, x_j) \in U$ ne sont pas orientés, c'est-à-dire que (x_i, x_j) est équivalent à (x_j, x_i). Un couple (x_i, x_j) est appelé une *arête*, et est représentée graphiquement par $x_i - x_j$.

Une *boucle* est une arête (un arc) dont les extrémités se coïncident.

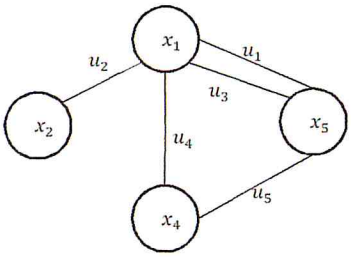
Ordre et taille d'un graphe

Le nombre de sommets d'un graphe G est appelé *ordre* de G, et est noté par $n = |V|$, et le nombre d'arêtes est appelé *la taille* de G, et est noté par $m = |E(G)|$. Un graphe est dit fini ou infini suivant son ordre. Deux sommets qui sont reliés par une arête (arc) sont dits *adjacents*.

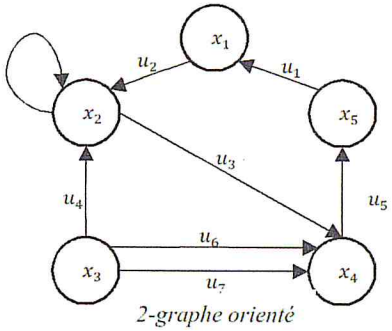
p-graphe et graphe simple.

- Si p est le nombre maximum des arcs de type (x_i, x_j), alors G est dit un *p-graphe*.

- Un graphe *simple* est un graphe sans boucles ni arête multiple (i.e. tout couple de sommets est relié par au plus une arête).



2-graphe non orienté



2-graphe orienté

Figure 1. 1 : Exemple de graphes orienté et non orienté

Chaîne et cycle

Une *chaîne* de longueur $k-1$ dans un graphe G est une séquence alternée de sommets et d'arêtes $x_1, u_1, x_2, u_2, \dots, x_{i-1}, u_{i-1}, x_i, \dots, u_{k-1}, x_k$ telle que $u_i = (x_i, x_{i+1})$ pour $i = 1,2, \dots, k-1$. Le nombre d'arêtes dans la chaîne définit sa longueur et le nombre de sommets définit son ordre. Une chaîne dans laquelle aucune arête ne se répète est dite *simple* et une chaîne dans laquelle aucun sommet ne se répète est dite *élémentaire*. Un *cycle* noté C_k de longueur k est une chaîne de longueur k dans lequel les deux extrémités initiale et terminale sont confondues, dans ce cas le nombre de sommets de C_k est égal à sa longueur. Dans le concept orienté on parle sur la notion de *chemin* et *circuit*.

Un chemin *élémentaire* qui passe par tous les sommets de graphe est dit *Hamiltonien*. Tandis qu'un circuit Hamiltonien est un chemin *élémentaire* dont les extrémités sont confondues.

Connexité

Un graphe G est dit connexe, si pour toute paire de sommets il existe une chaîne les reliant. Un graphe qui n'est pas connexe est dit *disconnexe* ou non connexe.

Dans tout ce qui suit on s'intéresse qu'aux graphes simples et finis.

1.2. Quelques types de graphes

Graphe orienté dérivé d'un graphe non orienté

Un *graphe orienté dérivé d'un graphe non orienté* est obtenu en introduisant deux arcs pour chaque arête, un dans chaque direction.

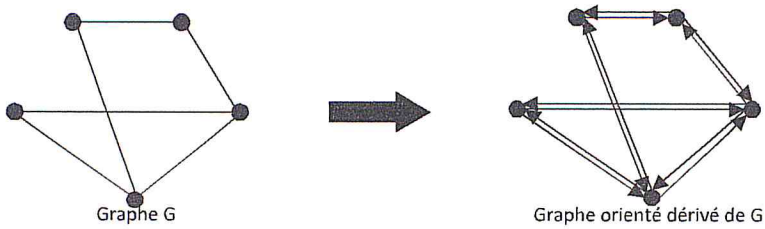


Figure 1. 2 : un graphe non orienté G et son graphe orienté dérivé

Graphe symétrique

Un graphe orienté $G = (X, U)$ est dit *symétrique* si et seulement si pour tout $x, y \in X$ *et* si $(x, y) \in U$, alors $(y, x) \in U$.

Graphe complet

- **Cas non orienté**

Un graphe non orienté est dit complet si tous ses sommets sont adjacents. Un graphe complet d'ordre n est noté K_n.

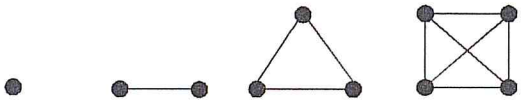


Figure 1. 3 : Graphes complets d'ordre n ∈ {1,2,3,4}

Dans un graphe non orienté d'ordre n le nombre des arêtes est $\frac{n(n-1)}{2}$ et le nombre des cycles est $(n-1)!$.

- **Cas orienté**

Un graphe orienté est dit complet si pour tout $x, y \in X$, $(x, y) \notin U$ alors $(y, x) \in U$.

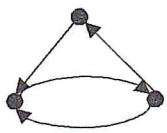


Figure 1. 4 : Graphes complet orienté d'ordre 3

Dans un graphe orienté d'ordre n le nombre des arcs est au plus $n(n-1)$ et le nombre des circuits est $(n-1)!$

Arbres et arborescences

Arbre : Un *arbre* est un graphe (orienté ou non) simple connexe et sans boucle.

Figure 1. 5 : Exemples d'arbres

Arborescence : Un sommet x d'un graphe orienté $G = (X, U)$ est dit *racine* si pour tout $y \in X$ il existe un chemin de x vers y. Une arborescence est un arbre admettant une racine.

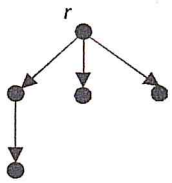


Figure 1. 6 : Une arborescence de racine r

Réseau

Un *réseau* $R = (X, U, d)$ est un graphe muni d'une application d de U dans R où d peut représenter les coûts sur les arêtes (arcs).

1.3. Représentation d'un graphe :

Le recours à un dessin sur une feuille de papier pour représenter un graphe (resp. un réseau) n'est pas toujours possible, en particulier lorsque l'ordre et la taille du graphe (resp. un réseau) sont grands. D'autre part, on est souvent amener à résoudre des problèmes posés sur les graphes (resp. les réseaux) par l'utilisation de micro-ordinateurs et de ce fait, il est nécessaire de chercher un autre mode de représentation.

Les représentations d'un graphe (réseau) sont en deux types :

- Les représentations tabulaires ou matricielles.
- Les représentations par listes chaînées.

On s'intéresse ici à la représentation matricielle en particulier per la matrice d'adjacence sommet-sommet.

Cas d'un graphe :

Définition : La matrice d'adjacence sommet-sommet d'un graphe $G = (X, U)$ est une matrice carrée $A(G) = \left(a_{ij}\right)_{\substack{i=1\ldots n \\ j=1\ldots n}}$ d'ordre $n \times n$ dont les lignes et les colonnes sont indexées par l'ensemble des sommets X et est définie par :

a_{ij} =Nombre d'arcs (d'arêtes) ayant le sommet x_i et le sommet x_j comme extrémités initiale et terminale, respectivement.

Remarque : Si G est un graphe simple alors $A(G)$ est une matrice formée de 0 et 1. De plus la diagonale est formée uniquement de 0 car G n'a pas de boucles.

Exemple :

Figure 1. 7 : Graphe G_1 et graphe G_2

Les matrices d'adjacences sommets-sommets sont :

$$A(G1) = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{array} \\ \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \end{pmatrix} \end{array}, \quad A(G2) = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Cas d'un réseau :

Considèrera des réseaux sur les graphes simples.

Définition : La matrice d'adjacence sommet-sommet d'un réseau $R = (X, U, d)$ est une matrice carrée $D(G) = \left(d_{ij}\right)_{\substack{i=1\ldots n \\ j=1\ldots n}}$ d'ordre $n \times n$ dont les lignes et les colonnes sont indexées par l'ensemble des sommets X et est définie par :

$$d_{ij} = \begin{cases} \text{la distance (le coût) entre le sommet } x_i \text{ et le sommet } x_j \\ \infty \qquad\qquad\qquad\qquad\quad s'\text{il n'existe pas un arc (arête)} \end{cases}$$

Exemple :

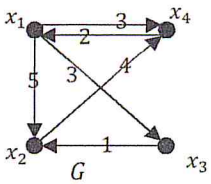


Figure 1. 8 : Un graphe G

La matrice d'adjacence du graphe de la Figure (8) est donnée par :

$$D(G) = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} \infty & 5 & 3 & 3 \\ \infty & \infty & \infty & 4 \\ \infty & 1 & \infty & \infty \\ 2 & \infty & \infty & \infty \end{pmatrix} \end{array}$$

1.4. Notions sur la complexité

Nous examinons brièvement les concepts de base de la théorie de la complexité, dans la mesure où ils sont utilisés dans ce travail.

Nous distinguons deux types de problèmes :

- Le problème de décision est un problème ayant deux réponses possibles : oui ou non.
- Le problème d'optimisation vise à détecter une solution minimisant (ou maximisant) une certaine fonction objectif dans un ensemble défini.

Définition 1 : Une fonction $f(n)$ est $O(g(n))$ ($f(n)$ est de complexité $g(n)$), s'il existe un réel $c > 0$ et un entier positif n_0 tel que pour tout $n \geq n_0$ on a $|f(n)| \leq c.g(n)$.

Définition 2 : Un *algorithme* est une suite finie d'opérations ou d'instructions permettant de résoudre un problème ou d'obtenir un résultat. Le mot algorithme vient du mot arabe الخوارزمي, nom du mathématicien du IX ^{e} siècle Al-Khwârizmî. Le domaine qui étudie les algorithmes est appelé l'algorithmique.

Définition 3 : Un algorithme en temps polynomial est un algorithme dont le temps de la complexité est en $O(p(n))$, où p est une fonction polynomiale et n est la taille de l'instance (ou sa longueur d'entrée). Si k est le plus grand exposant de ce polynôme en n, le problème correspondant est dit : résoluble en $O(n^k)$.

Définition 4 : (Classe P**)** Un problème de décision est dit appartenir à la *classe* P, s'il existe un algorithme pouvant le résoudre en temps polynomial.

Un exemple de problème polynomial est celui de la connexité dans un graphe.

Définition 5 : (Classe NP**))** La *classe* NP renferme tous les problèmes de décision dont on peut associer à chacun d'eux un ensemble de solutions potentielles (de cardinal au pire exponentiel) tel qu'on puisse vérifier en un temps polynomial si une solution potentielle satisfait la question posée. De toute évidence, $P \subset NP$ qu'on suppose communément acceptée. De plus cette inclusion est stricte, c'est-à-dire $P \neq NP$.

Définition 6 : (Classe *NP-complet***)** On distingue également dans la *classe* NP, la classe des problèmes *NP-complets*. La NP-complétude s'appuie sur la notion de réduction polynomiale. Un problème de décision $P1$ se réduit polynomialement en un problème de décision $P2$ s'il

existe une fonction polynomiale f telle que, pour toute instance I de $P1$, la réponse est oui si et seulement si la réponse de $f(I)$ pour $P2$ est oui.

Définition 7 : (Classe *NP-difficile*) Un problème est *NP-difficile* si savoir le résoudre en temps polynomial impliquerait que l'on sait résoudre en problème (de décision) *NP-complet* en temps polynomial. Les problèmes *NP-difficiles* sont donc dans un sens plus dur que les problèmes *NP-complet*.

Ce schéma représente la relation entre les classes des problèmes :

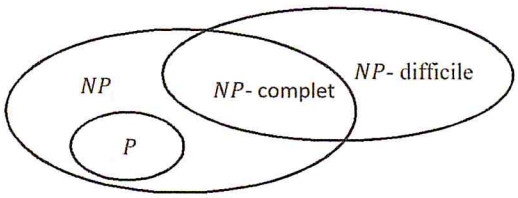


Figure 1. 9 : Relation entre les classes des problèmes.

Exemples :

L'ensemble des problèmes d'optimisation est très vaste on peut citer :

- Le problème de voyageur de commerce (classe *NP*-difficile).
- Le problème de connexité d'un graphe (classe *P*).
- Le problème de circuit Hamiltonien (classe *NP*-complet).

Chapitre 2

Chapitre 2

Problème de voyageur de commerce

Ce chapitre composé de cinq sections sera consacré à donner la description du problème du voyageur de commerce avec un historique ainsi que ses variantes.

2.1. Description et historique

Le problème du voyageur de commerce P.V.C. ou TSP (Traveling Salesman Problem), est défini comme suit : Un voyageur de commerce ayant n villes à visiter souhaite établir une tournée qui lui permette de passer exactement une fois par chaque ville et de revenir à son point de départ pour un moindre coût. Donc c'est la recherche d'un circuit Hamiltonien dont le coût total de parcours est le minimum possible [10].

En général le problème de voyageur de commerce est donné par un réseau $R(X, U, c)$ où le graphe associé à ce problème est un graphe simple et sans boucle (orienté ou non orienté).

X : est l'ensemble des sommets.

U : est l'ensemble des arcs.

c : est une application de U dans $\mathbb{R}^+$.

On peut associer à ce réseau la matrice C $(n \times n)$ d'adjacence sommets-sommets où :

$$C_{ij} = \begin{cases} c_{ij} \left(le\ coût\ d'aller\ de\ x_i\ vers\ x_j\right) si\ (x_i, x_j) \in U \\ \infty \qquad\qquad sinon \end{cases}$$

Le P.V.C appartient à la famille des problèmes NP-difficiles [3] dont les méthodes de résolution peuvent s'appliquer à d'autres problèmes mathématiques discrets. La complexité en temps des algorithmes exacts proposés croît exponentiellement avec n, la taille du problème ou le nombre de villes.

Ce problème est mentionné pour la première fois en 1832 dans un manuel comportant des exemples de tours dans l'Allemagne et la Suisse, mais ce dernier n'est pas un traitement mathématique du problème. La première relation entre ce problème et les mathématiques est traité en 1859 par le mathématicien Irlandais Sir William Roman Hamilton et le mathématicien Angler Tomas Kirkman par le jeu Icosien.

Figure 2. 1 : jeu d'Icosien

En 1930, le P.V.C. est traité plus en profondeur par Karl Menger à Harvard. Mais la première référence sous sa forme moderne est posée en 1949 par J. B. Robinson puis il a été popularisé dans la communauté mathématique par M. Flood et les chercheures de RAND (Research ANd Developpent, recherche et développement).

Par la suite, divers chercheurs ont étudié ce problème et ont donné des algorithmes exacts et approchés pour le résoudre, citons quelques noms. Concernant les algorithmes exacts on trouve les algorithmes de coupe posés par Gomory en 1958, la méthode de séparation et évaluation de Little et al. 1963, Faure en 1978 et Gondran & Minoux en 1979 et les algorithmes de la programmation dynamique donnés par Srivasturva et al. qui ont étudiés le cas symétrique et Hanry-Labordere qui a étudié le cas asymétrique en 1969.

Laporte et Nobert 1983_1987 ont formulé les deux problèmes (symétrique et asymétrique) en un programme linéaire en nombre entier et ont établi un algorithme de séparation et évaluation pour les résoudre.

Pour les heuristiques on trouve les travaux de M. Held & R.M. Karp en 1962 qui ont donnés une heuristique qui se base sur la programmation dynamique. En 1973 on trouve les travaux de Lin & Kernighan et Christofiedes.

L'augmentation de la résolution de P.V.C. sous forme du tableau suivant :

Années	Nom des chercheurs	Nombre de ville
1954	G. Dantzig, R. Fulkerson et S. Johnson	49
1971	M. Held et R.M. Karp	64
1975	P.M. Camerini, L. Fratta et F. Maffoli	67
1977	M. Grötshel	120
1980	H. Crowder et M.W. Padberg	318
1987	M.W. Padberg et G. Rinaldi	532
1987	M. Grötshel et O. Halland	666
1987	M.W. Padberg et G. Rinaldi	2392
1994	D. Applegate, R. Bixby, V. Chvatel et W. Cook	7397
1998	D. Applegate, R. Bixby, V. Chvatel et W. Cook	13509
2001	D. Applegate, R. Bixby, V. Chvatel et W. Cook	15112
2004	D. Applegate, R. Bixby, V. Chvatel, W. Cook et K. Helsgaun	24978
2006	D. Applegate, R. Bixby, V. Chvatel et W. Cook	85900

Tableau 2.1 divers augmentation de P.V.C (www.tsp.gatech.edu/history)

2.2. Variantes du problème du voyageur de commerce

Selon les conditions et les types des données, on distingue plusieurs variantes du problème du voyageur de commerce qu'on peut les indiquer comme suit :

- **Le problème du voyageur de commerce est symétrique :** Si on a des arêtes c.-à-d. le coût d'aller de la ville i vers la ville j est le même que d'aller de la ville j vers la ville i ($c_{ij} = c_{ji} \ \forall i,j \quad i \neq j$).
- **Le problème du voyageur de commerce est asymétrique :** Si on a des arcs c.-à-d. le coût d'aller de la ville i vers la ville j n'est pas forcément le même que d'aller de la ville j vers la ville i (il existe au moins un couple (i,j) telle que $c_{ij} \neq c_{ji}$).

Le passage du cas symétrique au cas asymétrique se fait par la notion de graphe orienté dérivé d'un graphe non orienté.

- **Problème du voyageur de commerce avec fenêtres de temps** (TSPTW pour Traveling Salesman Problem with Time Windows) dans lequel la visite de chaque ville doit se faire dans un intervalle de temps donné.

- **Problème du voyageur de commerce généralisé** où le voyageur doit passer par un nombre prédéfini de sous-ensembles de villes. Il doit visiter au moins une ville de chaque sous-ensemble et minimiser la somme des coûts du voyage.
- **Problème de tournée des véhicules** (VRP pour Vehicule Routing Problem) dans lesquels on ne considère plus un unique représentant de commerce pour visiter les villes mais une équipe de véhicules et qui peuvent être vu comme des problèmes de flot.

2.3. Utilisation de P.V.C.

Le problème du voyageur de commerce est beaucoup étudié en recherche opérationnelle à cause de son utilisation très vaste dans les différents domaines, voir [8] :

- **En informatique**, pour les problèmes de routage quand il y a plusieurs routeurs dans un réseau on utilise le PVC pour faciliter le transport.
- **En industrie**, les techniques du PVC sont utilisées pour déterminer l'ordre de pose des microcomposants (transistors, portes logiques, etc...) pour réduire le temps de production des processeurs et micro-processeurs. Pour les circuits imprimés, le PVC est utilisé pour déterminer l'ordre de perçage et de soudure des trous de connexion.
- **En astronomie**, que ça soit sur terre ou dans l'espace, faire pivoter le télescope pour l'orienter et le pointer vers une succession de cibles à observer, est coûteux en temps et même en carburant dans le cas d'un télescope spatial, il est donc plus que judicieux d'user des méthodes permettant de minimiser ces mouvements.
- **En logistique**, l'intérêt du PVC est évident même pour les non-initiés. En effet, plus courte est la distance parcourue pour livrer un ensemble de clients, moins coûteux est le carburant et plus importante devient la marge bénéficiaire, de même quand l'objectif est de minimiser le temps nécessaire à la réalisation de l'ensemble des livraisons, plus rapidement un livreur termine sa tournée, plus il pourra en faire en une journée ou sur une durée déterminée.
- **En neurologie**, le principe du problème est expliqué à des patients victimes de lésions au niveau du cortex préfrontal, où sont situées les facultés de planification visuo-spatiale, à qui l'on demande de résoudre une instance donnée. Les médecins observent alors les délais pris par ces patients pour déterminer la prochaine ville à visiter, ces délais servent ensuite à déterminer la sévérité de leur lésion cérébrale.

2.4. Définition et formulation d'un PLNE

Un problème linéaire en nombre entier (PLNE) est un problème d'optimisation où la fonction objectif est linéaire ainsi que les contraintes mais les variables sont entiers.

Un PLNE est sous la forme :

$$\min Z = C.x$$
$$\begin{cases} A.x \leq b \\ x \in \mathbb{N}^n \end{cases}$$

Avec $C \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ et A est une matrice à valeur réelles d'ordre $m \times n$ où n est nombre de variable et m est le nombre de contraintes.

Donc un PLNE est chercher le minimum d'une fonction linéaire $C.x$ dans un ensemble des solutions réalisables discrètes $S = \{x \in \mathbb{N}^n / A.x \leq b\}$.

2.5. Modélisation de PVC sous forme d'un PLNE

Il existe plusieurs modélisations du problème de voyageur de commerce dans le cas asymétrique (qui sera applicable même sur le cas symétriques par l'application de la notion de dériver le graphe orienté du graphe non orienté) mais on va donner uniquement la modélisation de Dantzig sous forme d'un programme linéaire en nombre entier établi en 1954 [5] :

$$\min \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij} \qquad (1)$$

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

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1 \dots n \qquad (3)$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \quad S \subset V, 2 \leq |S| \leq n - 2 \qquad (4)$$

$$x_{ij} \in \{0,1\} \quad i,j = 1, \dots, ni \neq j \qquad (5)$$

On peut expliquer les formules (1), (2), (3), (4) et (5). Comme suit :

(1) La fonction dont l'objectif est la somme des coûts unitaires des arcs pris dans le circuit.

(2) Ce groupe de contraintes signifié qu'il y a un arc qui a le sommet i comme successeur.

(3) Ce groupe de contraintes signifié qu'il y a un arc qui a le sommet j comme prédécesseur.

(4) Ce groupe de contraintes signifié que le graphe soit connexe.

(5) Les variables $x_{ij} = \begin{cases} 0 & si\ on\ prend\ pas\ l'arc\ (i,j) \\ 1 & si\ on\ prend\ l'arc\ (i,j) \end{cases}$

Cependant, cette formulation a un nombre de variables de $n^2 - n$ et $O(n^2)$ contraintes, ceci rend la résolution du modèle très difficile.

Chapitre 3

Chapitre 3

Résolutions des problèmes NP-difficiles

Il n'existe pas des algorithmes polynomiaux pour résoudre les problèmes NP-difficiles. Donc il est impératif de trouver d'autre type d'algorithmes pour les résoudre. Dans ce chapitre, on donne un aperçu général des différentes approches, qu'elles soient exactes comme les algorithmes de séparation et évaluation, ou les algorithmes heuristiques et métaheuristiques, en particulier les algorithmes de colonie de fourmi et la recherche tabou. Pour plus d'information l'auteur est invité à consulter les ouvrages [6] et [2].

3.1 Les méthodes exactes :

L'intérêt des méthodes exactes réside dans le fait qu'elles assurent l'obtention de la solution optimale du problème traité. En fait, elles permettent de parcourir la totalité de l'ensemble de l'espace de recherche de manière à assurer l'obtention de toutes les solutions ayant le potentiel d'être meilleures que la solution optimale trouvée au cours de la recherche.

Cependant, les méthodes exactes sont très connues par le fait qu'elles nécessitent un coût de recherche souvent prohibitif en termes de ressources requises. En effet, le temps de recherche et/ou l'espace mémoire nécessaire pour l'obtention de la solution optimale par une méthode exacte sont souvent trop grands, notamment avec des problèmes de grandes tailles.

De ce fait, la complexité de ce type d'algorithme croit exponentiellement avec la taille de l'instance à traiter, elle devient très importante face à des problèmes comprenant plusieurs variables, fonctions objectifs et/ou critères.

3.1.1. Les méthodes Branch and Bound

Les méthodes par séparation et évaluation notées B&B (Branch & Bound) sont des méthodes arborescentes qui énumèrent d'une façon intelligente à travers les nœuds (qui représentent des sous-ensembles de l'ensemble global des solutions). Le principe est d'éliminer de l'arbre de la recherche les branches prouvés non optimale afin d'éviter l'énumération complète des solutions. Dans les problèmes de minimisation, on cherche pour chaque nœud une borne inférieure (pour les problèmes de maximisation on cherche une borne supérieure) qui donne l'efficacité du nœud.

Il n'existe pas un procédé unique pour ces méthodes, mais pour chacune de ces méthodes on doit donner :

- Une procédure par évaluation.
- Une procédure pour séparation.
- Une stratégie de parcours.

Début

Initialiser avec le calcul d'une solution réalisable de valeur poser b_{best} ou poser $b_{best} = +\infty$.

Tant qu'il existe des nœuds non stérilisés

Faire

Chercher un nœud S tel que $b(S) < b_{best}$

Si (S est une solution réalisable)

$b_{best} = b(S)$.

Sinon

séparer S.

Fin si

Fin tant que

S est la solution optimale du problème.

Fin

Algorithme 3.1 : Algorithme de séparation et évaluation (cas général)

3.1.1.1. Procédure de séparation (branching process)

La séparation consiste à diviser -selon une condition fixée- le problème en un certain nombre de sous-nœuds, le nœud étudié de telle sorte que l'union de tous les sous-nœuds donne le nœud séparé pour éviter la perte des solutions. Ainsi, en résolvant tous les sous-problèmes (le nœud racine est l'ensemble des solutions réalisables et un sous-nœud est un sous-ensemble de l'ensemble des solutions réalisables du problème est donc un sous-nœud est un sous-problème) et en prenant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Ce principe de séparation peut être appliqué de manière récursive à chacun des sous-ensembles de solutions obtenus, et ceci tant qu'il y a des ensembles contenant plusieurs solutions. Les ensembles de solutions (et leurs sous-problèmes associés) ainsi construits ont une hiérarchie naturelle en arbre, souvent appelée *arbre de recherche* ou *arbre de décision*.

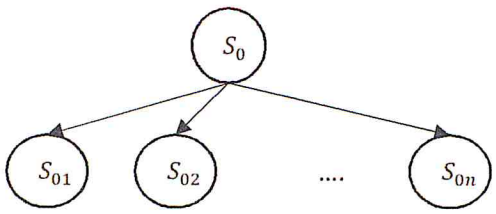


Figure 3.1: séparation d'un nœud S en n sous nœuds.

3.1.1.2. Procédure d'évaluation (bounding process)

L'évaluation d'un nœud de l'arbre de recherche a pour but de déterminer l'optimum de l'ensemble des solutions réalisables associé au nœud en question ou, au contraire, de prouver mathématiquement que cet ensemble ne contient pas de solution intéressante pour la résolution du problème (typiquement, qu'il n'y a pas de solution optimale). Lorsqu'un tel nœud est identifié dans l'arbre de recherche, il est donc inutile d'effectuer la séparation de son espace de solutions.

À un nœud donné, l'optimum du sous-problème peut être déterminé lorsque le sous-problème devient « suffisamment simple ». Par exemple, lorsque l'ensemble des solutions réalisables devient un singleton, le problème est effectivement simple : l'optimum est l'unique élément de l'ensemble. Dans d'autres cas, il arrive que par le jeu des séparations, on arrive à un sous-problème dans lequel les décisions « difficiles » ont été prises et qui peut ainsi être résolu en temps polynomial.

Pour déterminer qu'un ensemble de solutions réalisables ne contient pas de solution optimale, la méthode la plus générale consiste à déterminer un minorant du coût des solutions contenues dans l'ensemble (s'il s'agit d'un problème de minimisation). Si on arrive à trouver un minorant qui est supérieur au coût de la meilleure solution trouvée jusqu'à présent, on a alors l'assurance que le sous-ensemble ne contient pas l'optimum.

3.1.1.3. Procédure de cheminement (stratégie de parcours)

La façon dont on parcourt l'arbre des solutions est donc le choix du prochain nœud actif à diviser. Plusieurs stratégies sont à envisager :

Largeur d'abord :

Cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les autres stratégies présentées.

Meilleur d'abord :

On divise le nœud de meilleure évaluation. Cette stratégie n'est utilisée que dans certains problèmes seulement.

Profondeur d'abord :

On choisit pour prochain nœud actif l'un des fils du nœud qui vient d'être divisé, si aucun de ses nœuds n'est actif on revient en arrière dans l'arbre. Cette stratégie à plusieurs avantages qui sont :

- Le nombre de nœuds actifs reste relativement faible et nécessite donc moins de mémoire.
- L'évaluation d'un fils peut profiter de l'évaluation du père. En d'autres termes, si on vient d'évaluer le père, ce qu'il y a en mémoire permet parfois d'évaluer un fils beaucoup plus rapidement que si on le faisait plus tard.
- Normalement tout nœud de l'arbre a au moins 2 fils, mais il arrive que, dans la description utilisée, certains nœuds n'aient qu'un fils. Si on sait qu'un nœud n'a qu'un fils, il est inutile de l'évaluer, mieux vaut évaluer son fils. Dans une recherche en profondeur, puisque l'on va de toute façon créer les fils, cela ne coûte rien d'inverser l'ordre.
- C'est au profond dans l'arbre qu'on a le plus de chances de tomber sur une solution l'hors de l'évaluation et donc d'obtenir une valeur qui permet d'élaguer des nœuds.

Mixte :

Cette stratégie combine les deux stratégies en profondeur d'abord et le meilleur d'abord, ainsi on va en profondeur tant qu'on peut sinon on saute au nœud de meilleur évaluation.
Le choix de la stratégie est pris en général selon le type de problème et leurs données.

3.2. Les méthodes approchées

3.2.1. Les méthodes heuristiques

Une heuristique est une méthode approchée se réclamant simple, rapide et adaptée à un problème particulier. Sa capacité à résoudre un problème difficilement résolvable par des méthodes exactes est nuancée par le fait qu'elle n'offre aucune garantie quant à la qualité de la solution calculée. Ce défaut n'est pas toujours un problème lorsque seule une approximation de la solution optimale est recherchée.

3.3. Les méthodes métaheuristiques

Dans la vie pratique, on se retrouve souvent confronté à des problèmes de différentes complexités, pour lesquelles on cherche des solutions qui satisfont deux notions antagonistes : la rapidité et la qualité. Devant le coût de recherche prohibitif des méthodes exactes (particulièrement avec des problèmes de grande taille) et la spécificité des heuristiques au problème donné, les métaheuristiques construisent une solution moins exigeante. En fait, elles sont applicables sur une grande variété de problèmes d'optimisation de différentes complexités. En outre, elles permettent de fournir des solutions de très bonne qualité (pas nécessairement optimales) en temps de calcul raisonnable.

La majorité des métaheuristiques sont inspirées des systèmes naturels, nous pouvons citer à titre d'exemple: le recuit simulé qui est inspiré d'un processus métallurgique, les algorithmes évolutionnaires et les algorithmes génétiques qui sont inspirés des principes de l'évolution Darwinienne et de la biologie, la recherche tabou qui s'inspire de la mémoire des êtres humains, les algorithmes basés sur l'intelligence d'essaim comme l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis et l'algorithme de colonies d'abeilles.

3.3.1. Algorithmes de colonie de fourmis

L'algorithme de colonies de fourmis est un des algorithmes basés sur l'intelligence par essaim. Il a été introduit au début des années 90 par le trinôme Colorni, Dorigo et Maniezzo. L'idée de base du trinôme imite le comportement collectif des fourmis lors de leur déplacement entre la fourmilière et la source de nourriture. L'objectif du comportement collectif des fourmis est de collecter la nourriture sans perdre le chemin le plus court menant à leur nid.

L'algorithme de colonies de fourmis a été proposé pour la première fois pour résoudre le problème du voyageur de commerce, il se base sur trois phases essentielles :

- La construction du trajet de chaque fourmi.
- La distribution de phéromones sur le trajet de chaque fourmi.
- Evaporation des pistes de phéromones.

```
Début
    Initialiser une population de m fourmis ;
    Evaluer les m fourmis ;
    Tant que la condition d'arrêt n'est pas satisfaite
    faire
        Pour i = 1 à n
        faire
            Construire le trajet de la fourmi i;
            Déposer des phéromones sur le trajet de la fourmi i;
        Fin pour
        Evaluer les m fourmis ;
        Evaporer les pistes de phéromones ;
    Fin Tant que
    Retourner la ou les meilleures solutions ;
Fin
```

Algorithme 3.2 : colonie de fourmi pour le PVC

Chapitre 4

Chapitre 4

Algorithme Branch and Bound de Little pour le PVC

En 1963, Little, Murty, Sweeney et Karel ont proposé un algorithme de séparation et évaluation pour résoudre le problème du voyageur de commerce (P.V.C.) qui adopte la stratégie "profondeur d'abord" [10].

4.1. Principe de l'algorithme

L'algorithme de Little et al est un algorithme de séparation et d'évaluation, adoptant la stratégie "profondeur d'abord". Dans cet algorithme, Little et al. [10] déterminent à partir de la matrice initiale des coûts une seconde matrice, appelée matrice réduite, telle que le coût de chaque circuit hamiltonien donné par la matrice initiale est supérieur ou égal à celui donné par la matrice réduite plus une constante fixée (l'égalité est atteinte pour tout circuit hamiltonien optimal). L'évaluation à chaque nœud de l'arbre de branchements est déterminée par la somme des constantes calculées pour chaque matrice depuis la racine jusqu'au nœud considéré. Des pénalités sont également ajoutées pour chaque arc interdit dans la solution. Le branchement consiste, étant donné un arc, à considérer un premier sous-problème dans lequel cet arc appartient au circuit hamiltonien et un deuxième dans lequel le circuit ne peut contenir cet arc (arc interdit). Durant l'algorithme, la matrice réduite est modifiée en ajoutant des coûts infinis associés aux arcs induisant des solutions non réalisables pour le problème (création de sous-circuits) [9].

4.2. Algorithme de Little et al.

Pour un nœud S on a les informations suivantes :

P_S est l'ensemble des arcs pris.
M_S est l'ensemble des arcs non pris.
$b(S)$ est la borne (l'évaluation) du nœud.
C_S est la matrice correspondante aux données précédentes.
$\bar{b}$ est la meilleure borne actuelle.
r_{ij} est le regret de l'arc (x_i, x_j) quand peut le définir comme la pénalité maximale de ne pas prendre l'arc.
S_D est le nœud droit et S_G est le nœud gauche.

La séparation :

La séparation consiste à considérer l'inclusion ou l'exclusion d'un trajet (x_k, x_l) dans une tournée. Donc on aura deux nœuds fils le premier (à droite) on accepte l'arc et le deuxième (à gauche) on le refuse. Chaque séparation produisant deux branches et par conséquent l'arborescence de recherche de l'algorithme de Little est binaire.

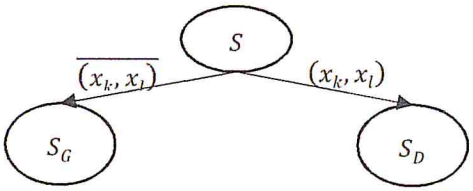


Figure 4.1 : séparation d'un nœud dans l'algorithme de Little.

Le pseudo code de la séparation d'un nœud est :

Début

Si $|P_S| = n - 1$

$P_S = P_S \cup \{(x_i, x_j), (x_k, x_l)\}$ où (x_i, x_j) et (x_k, x_l) sont les arcs restants non interdit.

Sinon

Pour chaque zéro trouvé

$$r_{ij} = \min\left\{C''_{ik} \ ; \ \begin{matrix} k = \overline{1, n_S} \\ k \neq j \end{matrix}\right\} + \min\left\{C''_{kj} \ ; \ \begin{matrix} k = \overline{1, n_S} \\ k \neq i \end{matrix}\right\}$$

Fin pour

Choisir l'arc (x_k, x_l) qui a le plus grand regret i.e. $r_{kl} = \max\{r_{ij} \ tq \ C''_{ij} = 0\}$.

Séparer S en S_G etS_D tel que :

$P_{S_D} = P_S \cup \{(x_k, x_l)\}$, $M_{S_D} = M_S$ et C_{S_D} est la matrice C''_S dont on supprime la ligne correspond à l'arc x_k et la colonne correspond à l'arcx_l et on met ∞ à l'arc qui introduit un sous circuit (l'intersection de la ligne et la colonne qui ne contient aucun infini).

Réduire les coûts de la matrice C_{S_D}.

$P_{S_G} = P_S$, $M_{S_G} = M_S \cup \{(x_k, x_l)\}$, $b(S_G) = b(S) + r_{kl}$ et C_{S_G} est la matrice C''_S dont on met ∞ à l'arc (x_k, x_l).

stériliser le nœud S.

Fin si

Fin

Algorithme 4.1 : Procédure de séparation pour Little

L'évaluation

Elle consiste à fournir une borne inférieure du coût de la tournée en effectuant des opérations sur la matrice de coûts.

La réduction des coûts est une initialisation à la séparation et une évaluation du nœud choisi leur pseudo code est présenté ci-dessous :

Début

Pour chaque ligne de C_S

Chercher c_i le minimum de cette ligne.

$b(S) = b(S) + c_i$.

On retranche la valeur trouvée de toute la ligne on obtient la matrice C'_S.

Fin pour

Pour chaque colonne de C'_S

Chercher l_i le minimum de cette colonne.

$b(S) = b(S) + l_i$.

On retranche la valeur trouvée de toute la colonne on obtient la matrice C''_S.

Fin pour

Fin

Algorithme 4.2 : Procédure de l'évaluation (réduction des coûts de la matrice)

Exemple d'une évaluation :

$$\begin{pmatrix} & x_1 & x_2 & x_3 & x_4 & x_5 & \\ x_1 & \infty & 11 & 1 & 7 & 9 & \mathbf{1} \\ x_2 & 5 & \infty & 3 & 12 & 3 & \mathbf{3} \\ x_3 & 7 & 1 & \infty & 9 & 13 & \mathbf{1} \\ x_4 & 14 & 9 & 5 & \infty & 4 & \mathbf{4} \\ x_5 & 3 & 12 & 7 & 1 & \infty & \mathbf{1} \end{pmatrix} \rightarrow C'_{S_0} = \begin{pmatrix} & x_1 & x_2 & x_3 & x_4 & x_5 \\ x_1 & \infty & 10 & 0 & 6 & 8 \\ x_2 & 2 & \infty & 0 & 9 & 0 \\ x_3 & 6 & 0 & \infty & 8 & 12 \\ x_4 & 10 & 4 & 1 & \infty & 0 \\ x_5 & 2 & 11 & 6 & 0 & \infty \end{pmatrix}$$

$$\begin{pmatrix} & x_1 & x_2 & x_3 & x_4 & x_5 & \\ x_1 & \infty & 10 & 0 & 6 & 8 & \\ x_2 & 2 & \infty & 0 & 9 & 0 & \\ x_3 & 6 & 0 & \infty & 8 & 12 & \\ x_4 & 10 & 4 & 1 & \infty & 0 & \\ x_5 & 2 & 11 & 6 & 0 & \infty & \\ & \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \end{pmatrix} \rightarrow C''_{S_0} = \begin{pmatrix} & x_1 & x_2 & x_3 & x_4 & x_5 \\ x_1 & \infty & 10 & 0 & 6 & 8 \\ x_2 & 0 & \infty & 0 & 9 & 0 \\ x_3 & 4 & 0 & \infty & 8 & 12 \\ x_4 & 8 & 4 & 1 & \infty & 0 \\ x_5 & 0 & 11 & 6 & 0 & \infty \end{pmatrix}$$

$b_{S_0} = 0 + 1 + 3 + 1 + 4 + 1 + 2 = 12.$

L'algorithme de Little est :

Début

$b(S) = 0, M_S = P_S = \emptyset, \overline{b} = +\infty$ et$C_S = C$.

Réduction des coûts de la matriceC_S.

Tant qu'il existe un nœud S tel que $b(S) < \overline{b}$.

Si $\overline{b} = +\infty$

Choisir S parmi les nœuds tels que $|P(S)|$ est maximum.

Si P_S réduit à un circuit hamiltonien

poser $\overline{b} = b(S)$

Sinon

séparer S.

Fin si

sinon

Choisir S tel que $b(S)$ est minimum parmi les nœuds tels que $|P(S)|$ est maximum.

Si P_S réduit à un circuit hamiltonien

poser $\overline{b} = \min\{\overline{b}, b(S)\}$

Sinon

séparer S.

Fin si

Fin si

Stériliser tout nœud S tel que $b(S) > \overline{b}$.

Fin tant que

S est une solution optimale du problème et $b(S)$ est le coût optimal.

Fin

Algorithme 4.3 : Algorithme de Little

4.3. Exemple :

Pour plus d'explication on applique l'algorithme de Little sur l'exemple suivant :

$$\begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 11 & 1 & 7 & 9 \\ x_2 & 5 & \infty & 3 & 12 & 3 \\ x_3 & 7 & 1 & \infty & 9 & 13 \\ x_4 & 14 & 9 & 5 & \infty & 4 \\ x_5 & 3 & 12 & 7 & 1 & \infty \end{array}.$$

On initialise pour le nœud S_0

$$C_{S_0} = \begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 11 & 1 & 7 & 9 \\ x_2 & 5 & \infty & 3 & 12 & 3 \\ x_3 & 7 & 1 & \infty & 9 & 13 \\ x_4 & 14 & 9 & 5 & \infty & 4 \\ x_5 & 3 & 12 & 7 & 1 & \infty \end{array}$$

$P_{S_0} = \emptyset$, $M_{S_0} = \emptyset$ et $b_{S_0} = 0$.

Réduction des coûts :

$$\begin{array}{c|ccccc|c} & x_1 & x_2 & x_3 & x_4 & x_5 & \\ \hline x_1 & \infty & 11 & 1 & 7 & 9 & \mathbf{1} \\ x_2 & 5 & \infty & 3 & 12 & 3 & \mathbf{3} \\ x_3 & 7 & 1 & \infty & 9 & 13 & \mathbf{1} \\ x_4 & 14 & 9 & 5 & \infty & 4 & \mathbf{4} \\ x_5 & 3 & 12 & 7 & 1 & \infty & \mathbf{1} \end{array} \rightarrow C'_{S_0} = \begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 10 & 0 & 6 & 8 \\ x_2 & 2 & \infty & 0 & 9 & 0 \\ x_3 & 6 & 0 & \infty & 8 & 12 \\ x_4 & 10 & 5 & 1 & \infty & 0 \\ x_5 & 2 & 11 & 6 & 0 & \infty \end{array}$$

$$\begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 10 & 0 & 6 & 8 \\ x_2 & 2 & \infty & 0 & 9 & 0 \\ x_3 & 6 & 0 & \infty & 8 & 12 \\ x_4 & 10 & 5 & 1 & \infty & 0 \\ x_5 & 2 & 11 & 6 & 0 & \infty \\ & \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \rightarrow C''_{S_0} = \begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 10 & 0 & 6 & 8 \\ x_2 & 0 & \infty & 0 & 9 & 0 \\ x_3 & 4 & 0 & \infty & 8 & 12 \\ x_4 & 8 & 5 & 1 & \infty & 0 \\ x_5 & 0 & 11 & 6 & 0 & \infty \end{array}$$

$b_{S_0} = 0 + 1 + 3 + 1 + 4 + 1 + 2 = 12$.

Comme P_{S_0} ne se réduit pas à un circuit alors on sépare le nœud S_0.

Calcul des regrets :

Par exemple pour le zéro de la case (x_1, x_3) on calcule le regret comme :

$r_{13} = \min\{10,6,8\} + \min\{0,1,6\} = 6$

$$\begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 10 & 0_6 & 6 & 8 \\ x_2 & 0_0 & \infty & 0_0 & 9 & 0_0 \\ x_3 & 4 & 0_9 & \infty & 8 & 12 \\ x_4 & 8 & 5 & 1 & \infty & 0_1 \\ x_5 & 0_0 & 11 & 6 & 0_6 & \infty \end{array}$$

Le maximum des regrets est $r_{32} = 9$ donc on sépare par rapport à l'arc (x_3, x_2).

Pour le nœud S_G :

$P_{S_G} = \emptyset, M_{S_G} = \{(x_3, x_2)\}$,

$$C_{S_G} = \begin{array}{c|ccccc} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 10 & 0 & 6 & 8 \\ x_2 & 0 & \infty & 0 & 9 & 0 \\ x_3 & 4 & \infty & \infty & 8 & 12 \\ x_4 & 8 & 4 & 1 & \infty & 0 \\ x_5 & 0 & 11 & 6 & 0 & \infty \end{array},$$

$b_{S_G} = 12 + 9 = 21.$

Pour le nœud S_D :

$P_{S_D} = \{(x_3, x_2)\}, M_{S_D} = \emptyset$,

$$C_{S_D} = \begin{array}{c|cccc} & x_1 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 6 & 8 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \end{array}$$

Pour calculer la borne du nœud S_D on réduit la matrice C_{S_D} :

$$\begin{array}{c|cccc} & x_1 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 6 & 8 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \end{array} \begin{array}{c} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{array} \rightarrow C'_{S_D} = \begin{array}{c|cccc} & x_1 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 6 & 8 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \end{array}.$$

$$\begin{array}{c|cccc} & x_1 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 6 & 8 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \\ & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \rightarrow C''_{S_D} = \begin{array}{c|cccc} & x_1 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 6 & 8 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \end{array}.$$

$b_{S_D} = 12 + 0 = 12.$

On note $S_1 = S_G$ et $S_2 = S_D$.

On stérilise le nœud S_0.

$\max_{i=1,2}\{|P_{S_i}|\} = |P_{S_2}|$.

Comme P_{S_2} ne se réduit pas à un circuit alors on sépare le nœud S_2.

Calcul des regrets :

$$\begin{array}{c|cccc} & x_1 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & 0_7 & 6 & 8 \\ x_2 & 0_0 & \infty & 9 & 0_0 \\ x_4 & 8 & 1 & \infty & 0_1 \\ x_5 & 0_0 & 6 & 0_6 & \infty \end{array}$$

Le maximum des regrets est $r_{13} = 7$ donc on sépare par rapport à l'arc (x_1, x_3).

Pour le nœud S_G :

$P_{S_G} = \{(x_3, x_2)\}$,$M_{S_G} = \{(x_1, x_3)\}$,

$$C_{S_G} = \begin{pmatrix} & x_1 & x_3 & x_4 & x_5 \\ x_1 & \infty & \infty & 6 & 8 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \end{pmatrix},$$

$b_{S_G} = 12 + 7 = 19.$

Pour le nœud S_D :

$$P_{S_D} = \{(x_3, x_2), (x_1, x_3)\}, M_{S_D} = \emptyset, C_{S_D} = \begin{pmatrix} & x_1 & x_4 & x_5 \\ x_2 & \infty & 9 & 0 \\ x_4 & 8 & \infty & 0 \\ x_5 & 0 & 0 & \infty \end{pmatrix}$$

Pour calculer la borne du nœud S_D on réduit la matrice C_{S_D} :

$$\begin{pmatrix} & x_1 & x_4 & x_5 & \\ x_2 & \infty & 9 & 0 & \mathbf{0} \\ x_4 & 8 & \infty & 0 & \mathbf{0} \\ x_5 & 0 & 0 & \infty & \mathbf{0} \end{pmatrix} \rightarrow C'_{S_D} = \begin{pmatrix} & x_1 & x_4 & x_5 \\ x_2 & \infty & 9 & 0 \\ x_4 & 8 & \infty & 0 \\ x_5 & 0 & 0 & \infty \end{pmatrix}.$$

$$\begin{pmatrix} & x_1 & x_4 & x_5 \\ x_2 & \infty & 9 & 0 \\ x_4 & 8 & \infty & 0 \\ x_5 & 0 & 0 & \infty \\ & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \rightarrow C''_{S_D} = \begin{pmatrix} & x_1 & x_4 & x_5 \\ x_2 & \infty & 9 & 0 \\ x_4 & 8 & \infty & 0 \\ x_5 & 0 & 0 & \infty \end{pmatrix}.$$

$b_{S_D} = 12 + 0 = 12.$

On note $S_3 = S_G$ et$S_4 = S_D$. Et on stérilise le nœud S_2.

$\max_{i=1,3,4}\{|P_{S_i}|\} = |P_{S_4}|$.

Comme P_{S_4} ne se réduit pas à un circuit alors on sépare le nœud S_4.

Calcul des regrets :

$$\begin{pmatrix} & x_1 & x_4 & x_5 \\ x_2 & \infty & 9 & 0_9 \\ x_4 & 8 & \infty & 0_8 \\ x_5 & 0_8 & 0_9 & \infty \end{pmatrix}$$

Le maximum des regrets est $r_{25} = 9$ donc on sépare par rapport à l'arc (x_2, x_5).

Pour le nœud S_G :

$P_{S_G} = \{(x_3, x_2), (x_1, x_3)\}$, $M_{S_G} = \{(x_2, x_5)\}$,

$$C_{S_G} = \begin{pmatrix} & x_1 & x_4 & x_5 \\ x_2 & \infty & 9 & \infty \\ x_4 & 8 & \infty & 0 \\ x_5 & 0 & 0 & \infty \end{pmatrix},$$

$b_{S_G} = 12 + 9 = 21.$

Pour le nœud S_D :

$P_{S_D} = \{(x_3, x_2), (x_1, x_3), (x_2, x_5)\}, M_{S_D} = \emptyset,$

$$C_{S_D} = \begin{pmatrix} & x_1 & x_4 \\ x_4 & 8 & \infty \\ x_5 & \infty & 0 \end{pmatrix}$$

Pour calculer la borne du nœud S_D on réduit la matrice C_{S_D} :

$$\begin{pmatrix} & x_1 & x_4 & \\ x_4 & 8 & \infty & \mathbf{8} \\ x_5 & \infty & 0 & \mathbf{0} \end{pmatrix} \rightarrow C'_{S_D} = \begin{pmatrix} & x_1 & x_4 \\ x_4 & 0 & \infty \\ x_5 & \infty & 0 \end{pmatrix}.$$

$$\begin{pmatrix} & x_1 & x_4 \\ x_4 & 0 & \infty \\ x_5 & \infty & 0 \\ & \mathbf{0} & \mathbf{0} \end{pmatrix} \rightarrow C''_{S_D} = \begin{pmatrix} & x_1 & x_4 \\ x_4 & 0 & \infty \\ x_5 & \infty & 0 \end{pmatrix}.$$

$b_{S_D} = 12 + 8 = 20.$

On pose $S_5 = S_G$ et $S_6 = S_D$. Et on stérilise le nœud S_4.

$\max_{i=1,3,5,6} \{|P_{S_i}|\} = |P_{S_6}|.$

On sépare le nœud S_6.

$n_{S_6} = 2$, donc :

$P_{S_6} = \{(x_3, x_2), (x_1, x_3), (x_2, x_5), (x_4, x_1), (x_5, x_4)\}$

P_{S_6} se réduit au circuit $\boldsymbol{x_1 x_3 x_2 x_5 x_4 x_1}$ donc $\bar{b} = b_{S_6} = 20.$

On stérilise les nœuds S_1 et S_5 et on sépare le nœud S_3 car $b_{S_3} < \bar{b}$ et on pose $\bar{b} = b_{S_3}$.

$$C_{S_3} = \begin{pmatrix} & x_1 & x_3 & x_4 & x_5 \\ x_1 & \infty & \infty & 6 & 8 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \end{pmatrix}$$

Réduction des coûts :

$$\begin{pmatrix} & x_1 & x_3 & x_4 & x_5 & \\ x_1 & \infty & \infty & 6 & 8 & \mathbf{6} \\ x_2 & 0 & \infty & 9 & 0 & \mathbf{0} \\ x_4 & 8 & 1 & \infty & 0 & \mathbf{0} \\ x_5 & 0 & 6 & 0 & \infty & \mathbf{0} \end{pmatrix} \rightarrow C'_{S_D} = \begin{pmatrix} & x_1 & x_3 & x_4 & x_5 \\ x_1 & \infty & \infty & 0 & 2 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \end{pmatrix}.$$

$$\begin{pmatrix} & x_1 & x_3 & x_4 & x_5 \\ x_1 & \infty & \infty & 0 & 2 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 1 & \infty & 0 \\ x_5 & 0 & 6 & 0 & \infty \\ & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \end{pmatrix} \rightarrow C''_{S_D} = \begin{pmatrix} & x_1 & x_3 & x_4 & x_5 \\ x_1 & \infty & \infty & 0 & 2 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & 0 & \infty & 0 \\ x_5 & 0 & 5 & 0 & \infty \end{pmatrix}.$$

$b_{S_3} = 12 + 6 = 19.$

Comme P_{S_3} ne se réduit pas à un circuit alors on sépare le nœud S_3.

Calcul des regrets :

$$\begin{array}{c|cccc} & x_1 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & \infty & 0_2 & 2 \\ x_2 & 0_0 & \infty & 9 & 0_0 \\ x_4 & 8 & 0_5 & \infty & 0_0 \\ x_5 & 0_0 & 5 & 0_6 & \infty \end{array}$$

Le maximum des regrets est $r_{43} = 5$ donc on sépare par rapport à l'arc (x_4, x_3).

Pour le nœud S_G :

$P_{S_G} = \{(x_3, x_2)\}, M_{S_G} = \{(x_1, x_3), (x_4, x_3)\},$

$$C_{S_G} = \begin{array}{c|cccc} & x_1 & x_3 & x_4 & x_5 \\ \hline x_1 & \infty & \infty & 0 & 2 \\ x_2 & 0 & \infty & 9 & 0 \\ x_4 & 8 & \infty & \infty & 0 \\ x_5 & 0 & 5 & 0 & \infty \end{array},$$

$b_{S_G} = 19 + 5 = 24.$

Pour le nœud S_D :

$P_{S_D} = \{(x_3, x_2), (x_4, x_3)\}, M_{S_D} = \{(x_1, x_3)\},$

$$C_{S_D} = \begin{array}{c|ccc} & x_1 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 2 \\ x_2 & 0 & \infty & 0 \\ x_5 & 0 & 0 & \infty \end{array}$$

Pour calculer la borne du nœud S_D on réduit la matrice C_{S_D} :

$$\begin{array}{c|ccc} & x_1 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 2 & \mathbf{0} \\ x_2 & 0 & \infty & 0 & \mathbf{0} \\ x_5 & 0 & 0 & \infty & \mathbf{0} \end{array} \rightarrow C'_{S_D} = \begin{array}{c|ccc} & x_1 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 2 \\ x_2 & 0 & \infty & 0 \\ x_5 & 0 & 0 & \infty \end{array}.$$

$$\begin{array}{c|ccc} & x_1 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 2 \\ x_2 & 0 & \infty & 0 \\ x_5 & 0 & 0 & \infty \\ & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \rightarrow C''_{S_D} = \begin{array}{c|ccc} & x_1 & x_4 & x_5 \\ \hline x_1 & \infty & 0 & 2 \\ x_2 & 0 & \infty & 0 \\ x_5 & 0 & 0 & \infty \end{array}.$$

$b_{S_D} = 19 + 0 = 19.$

On note $S_7 = S_G$ et $S_8 = S_D$. Et on stérilise les nœuds S_3 et S_7.

$\max_{i=8}\{|P_{S_i}|\} = |P_{S_8}|.$

Comme P_{S_8} ne se réduit pas à un circuit alors on sépare le nœud S_2.

Calcul des regrets :

$$\begin{pmatrix} & x_1 & x_4 & x_5 \\ x_1 & \infty & 0_2 & 2 \\ x_2 & 0_0 & \infty & 0_2 \\ x_5 & 0_0 & 0_0 & \infty \end{pmatrix}$$

Le maximum des regrets est $r_{14} = 2$ donc on sépare par rapport à l'arc (x_1, x_4).

Pour le nœud S_G :

$P_{S_G} = \{(x_3, x_2), (x_4, x_3)\}$, $M_{S_G} = \{(x_1, x_4)(x_1, x_3)\}$,

$$C_{S_G} = \begin{pmatrix} & x_1 & x_4 & x_5 \\ x_1 & \infty & \infty & 2 \\ x_2 & 0 & \infty & 0 \\ x_5 & 0 & 0 & \infty \end{pmatrix},$$

$b_{S_G} = 19 + 2 = 21.$

Pour le nœud S_D :

$P_{S_D} = \{(x_3, x_2), (x_4, x_3), (x_1, x_4)\}$, $M_{S_D} = \{(x_1, x_3)\}$,

$$C_{S_D} = \begin{pmatrix} & x_1 & x_5 \\ x_2 & \infty & 0 \\ x_5 & 0 & \infty \end{pmatrix}$$

Pour calculer la borne du nœud S_D on réduit la matrice C_{S_D} :

$$\begin{pmatrix} & x_1 & x_5 & \\ x_2 & \infty & 0 & \mathbf{0} \\ x_5 & 0 & \infty & \mathbf{0} \end{pmatrix} \rightarrow C'_{S_D} = \begin{pmatrix} & x_1 & x_5 \\ x_2 & \infty & 0 \\ x_5 & 0 & \infty \end{pmatrix}.$$

$$\begin{pmatrix} & x_1 & x_5 \\ x_2 & \infty & 0 \\ x_5 & 0 & \infty \\ & \mathbf{0} & \mathbf{0} \end{pmatrix} \rightarrow C''_{S_D} = \begin{pmatrix} & x_1 & x_5 \\ x_2 & \infty & 0 \\ x_5 & 0 & \infty \end{pmatrix}.$$

$b_{S_D} = 19 + 0 = 19.$

On pose $S_9 = S_G$ et $S_{10} = S_D$. Et on stérilise les nœuds S_8 etS_9.

$\max_{i=10}\{|P_{S_i}|\} = |P_{S_{10}}|$.

On sépare le nœud S_{10}.

$n_{S_{10}} = 2$, donc :

$P_{S_{10}} = \{(x_3, x_2), (x_4, x_3), (x_1, x_4), (x_2, x_5), (x_5, x_1)\}$

$P_{S_{10}}$ se réduit au circuit $\boldsymbol{x_1x_4x_3x_2x_5x_1}$ et comme $\overline{b} > b_{S_{10}}$ donc $\overline{b} = b_{S_{10}} = 19$.

Il n'existe aucun nœud non stérilisé donc la solution optimale du problème est :

$\boldsymbol{x_1x_4x_3x_2x_5x_1}$ avec le coût optimal est 19.

L'arborescence de recherche est :

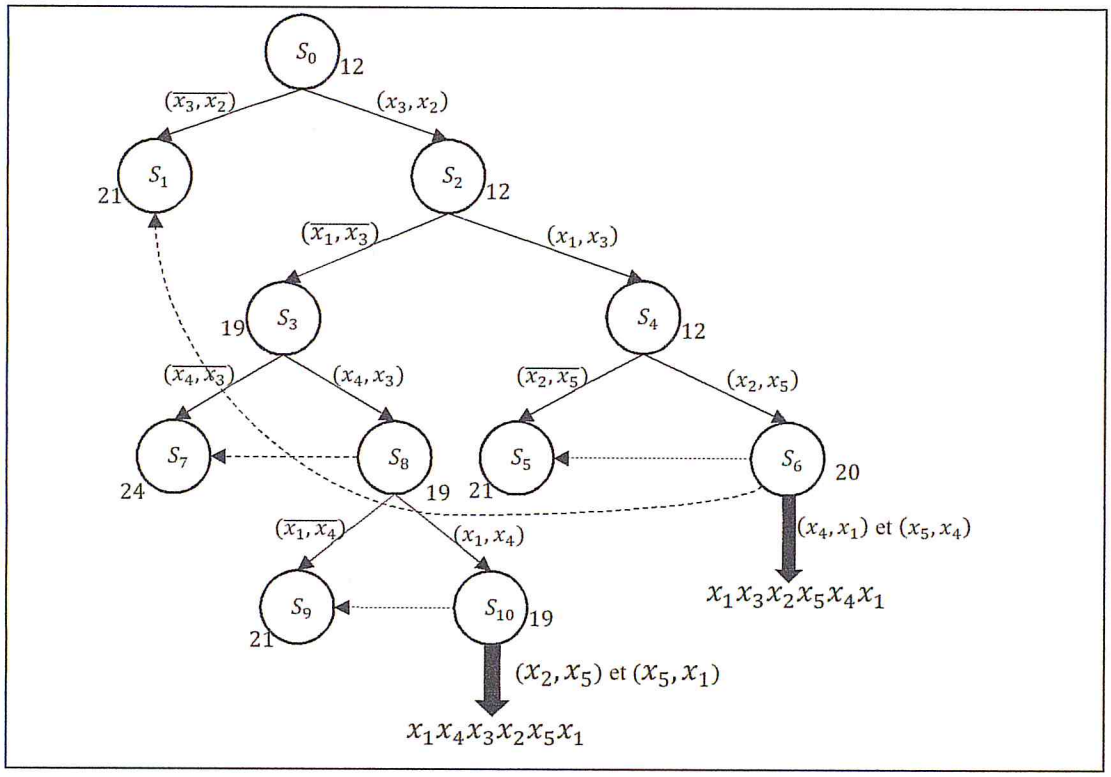


Figure 4.2 : l'arborescence de recherche

4.4. Implémentation de l'algorithme de Little

Nous avons réalisé une implémentation (un logiciel) simple en C++ Builder de cet algorithme. Ce logiciel marche pour n'importe quelle taille compris entre 2 et 99 et donne le circuit de coût optimal. Mais s'il le nombre des nœuds dépasse le 1000 le programme donne une solution proche de la solution optimale.

Pour mieux comprendre le fonctionnement de ce logiciel on repend l'exemple précèdent. Le logiciel comporte plusieurs fenêtres alors :

Fenêtre 1 :

C'est la fenêtre d'initialisation

Figure 4.3 : Fenêtre 1 du logiciel

Pour sortir du logiciel, il faut cliquer sur le bouton "sortie"

Pour commencer il faut cliquer sur le bouton "suivant". Ainsi la fenêtre 2 s'ouvre

Fenêtre 2

Dans cette fenêtre on introduit le nombre des villes du problème que l'on veut le traiter.

Figure 4.4 : fenêtre 2 du logiciel

Si on donne la valeur 0, 1 ou plus de 99 pour le nombre de villes on aura la fenêtre 3.

Fenêtre 3

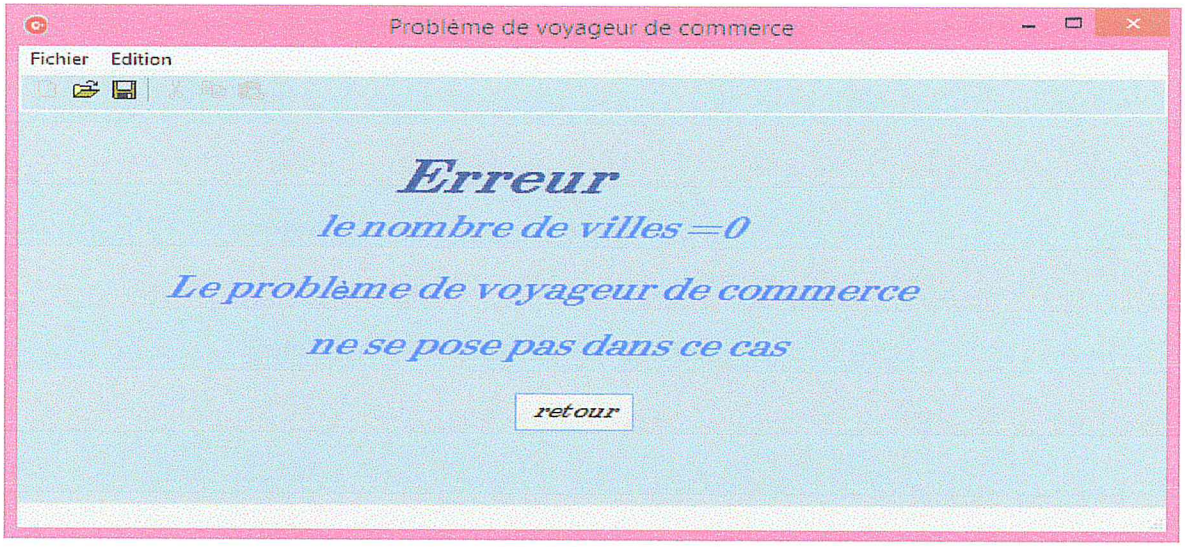


Figure 4.5 : la fenêtre 3 du logiciel

Pour revenir à la fenêtre 2 on clique sur retour.

Dans notre exemple où le nombre de villes est 5, on entre cette valeur. En validant et on obtient la fenêtre 4.

Fenêtre 4

Dans cette fenêtre, on a une matrice à remplir.

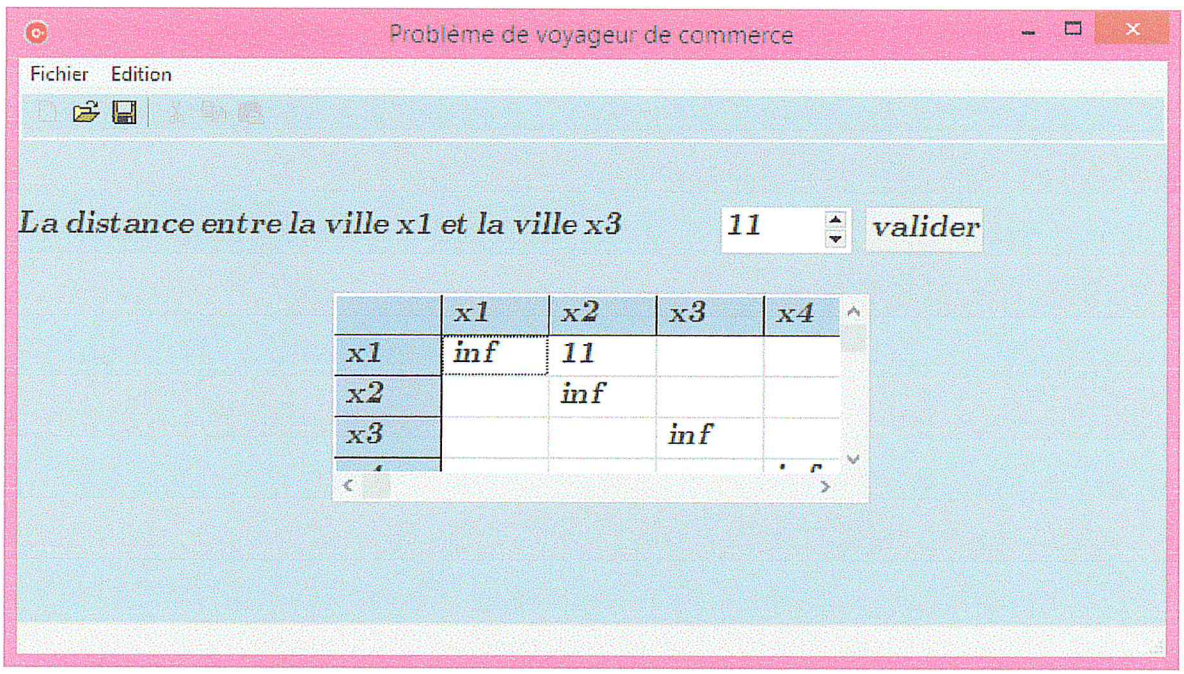


Figure 4.6 : la fenêtre 4 du logiciel

Le remplissage se fait de gauche vers la droite et de haut vers le bas.

Par exemple pour remplir la case (x_1, x_2) on entre la valeur 11 et on clique valider alors la valeur s'inscrit et ainsi de suite jusqu'à l'obtention de la fenêtre 5.

Fenêtre 5

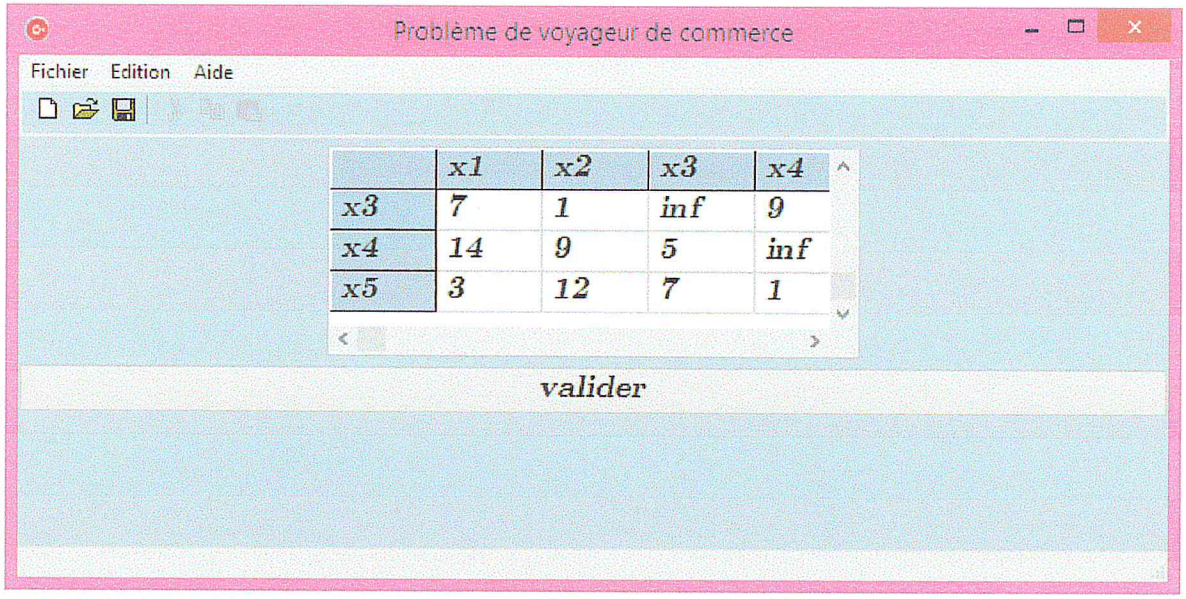


Figure 4.7 : la fenêtre 5 du logiciel

Une fois les données saisies, la résolution s'entame en cliquant sur "valider" la fenêtre 6 s'ouvre en affichent la solution actuelle avec le temps d'exécution.

Fenêtre 6

La solution sera affichée.

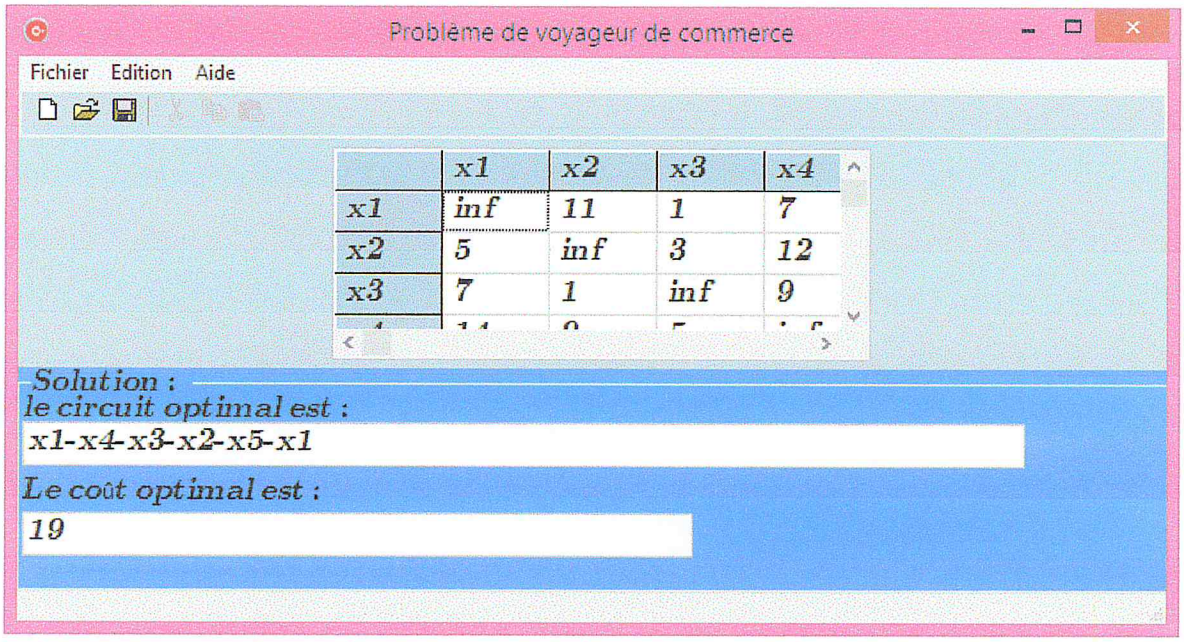


Figure 4.8 : fenêtre 6 du logiciel (le résultat).

Conclusion

Conclusion

Dans ce mémoire, nous avons étudié l'algorithme de Little de type de séparation et évaluation est (B & B) qui est un des algorithmes permettant la résolution exacte de problèmes d'optimisation en nombre entier dont on cherche à minimiser le coût de la recherche. La méthode B & B propose un mécanisme de recherche très intelligent, grâce à lequel elle permet une bonne exploitation de l'espace de recherche et l'aboutissement à la solution optimale plus rapidement que d'autres méthodes exactes en combinant deux principes primordiaux : la séparation et l'évaluation.

Le point fort de cette méthode réside dans le fait qu'elle ne parcourt pas les sous branches dont on peut savoir à priori qu'elles ne permettent pas d'améliorer la solution rencontrée, ce qui est établi grâce aux bornes des nœuds. Cela permet de trouver de bonnes solutions en un temps de recherche relativement court. L'efficacité de cette méthode a attiré l'attention de nombreux chercheurs. Par conséquent, plusieurs améliorations de l'algorithme B & B ont été proposées, y compris les algorithmes : Branch and Cut (noté B & C), Branch and Price (noté B & P), Branch and Cut and Price (B & C & P).

Références

Références

[1] C. Berge, Graphes et Hypergraphes. Dunod, deuxième Édition, 1970.

[2] S. Beetayaeb, Optimisation des conditions des coupes par la méthode de colonie de fourmi. Mémoire de magistère Université de Hadj Lakhdar -Batna- 2013.

[3] S. Cook. The Complexity of Theorem-Proving Procedure. Proc. 3rd ACP Symp. On Theory of Complexity, Association of Computing Machinery. New York. pp. 151-158, 1971

[4] G. Chartrand, et L. Lesniak, Graphs &Digraphs. Third Edition, Chapman&Hall, London, 1996.

[5] G. Dantzig, R. Fulkerson et S. Joheson, Solution of a Large-Scale Traveling Selesman problem. The Rand Corporation, Santa Monica California, Received August 9, 1954.

[6] Amira Gherboudj. Méthodes de résolution de problèmes difficiles académiques. Thèse de doctorat Université de Constantine2 2013.

[7] Michel Gondran et Michel Minoux, Graphes et algorithmiques. Troisième Edition – Paris : Eyrolles, 1995.

[8] Mounir Hamani, Nouvelle heuristique pour la résolution du problème de voyageur de commerce. Thèse de master 2 Université A. Mira de Bejaia 2016.

[9] Mathieu Lacroix, Le problème de ramassage et livraison préemptifs : complexité, modèle et polyèdres. Thèse du doctorat Université Blaise Pascal-Cleront-Ferrand II 2009.

[10] J. D. C. Little, K. G. Murty, D. W. Sweeney et C. Karel, An algorithm for the traveling salesman problem. Oper. Res., vol. 11, 1963.