UNIVERSITÉ ABDELHAMID IBN BADIS-MOSTAGANEM
FACULTÉ DES SCIENCES EXACTES ET L'INFORMATIQUE
DÉPARTEMENT DE MATHÉMATIQUES

Mémoire de Master

Spécialité : modélisation contrôle et optimisation

Thème

Présenté par

Soutenu le 2015//2016

Devant le jury

Examinateur	**U. MOSTAGANEM.**
Encadreur	**U. MOSTAGANEM.**

Table des matières

INTRODUCTION

Dans ce mémoire nous présentons un problème d'optimisation, bien connu, qui est le problème du voyageur de commerce, appelé aussi (en anglais "traveling salesman problem"), noté TSP. Ce problème consite à :

Soit un représentant de commerce qui doit visiter un certain nombre de clients situés dans n villes. Le coût de déplacement d'une ville i à une ville j est donné. L'objectif du représentant de commerce est de parcourir toutes les villes avec un coût total qui soit minimum, mathématiquement, cela consiste à déterminer, dans un graphe, un cycle Hamiltonien qui soit de longueur minimum.

Ce problème est connu pour être un problème d'optimisation combinatoire appartenant à la classe des problèmes dite NP-complets. Il possède énormément d'applications pratiques et à connu plusieurs généralisations.

Plusieurs méthodes de résolutions du problème (TSP) ont été proposées dans la littérature. Ces méthodes peuvent être classées en deux types :

- Les méthodes exactes qui garantissent la complétude de la résolution telle que la méthode de séparation et évaluation (en anglais Brunch and bound) ou des méthodes quasi-exactes comme la méthode de sous gradient.
- Les méthodes approchées qui "sacrifient" la complétude pour gagner en efficacité telles que : La méthode du plus proche voisin, du recuit simulé, les algorithmes génétiques, colonie de fourmis.

Notre travail est organisé de la manière suivante :

Au premier chapitre ,nous donnons quelques rappels sur la théorie des graphes, au deuxième chapitre nous introduisons le problème du voyageur de commerce , le troisième chapitre est consacré à la description de quelques méthodes de résolution puis au dernier chapitre nous présentons quelques tests numériques.

Nous terminons notre travail par une conclusion et quelques références bibliographiques.

Chapitre 1

Notions sur les graphes

1.1 Graphes non orientés

Un graphe $G = (X, U)$ est défini par la donnée d'un ensemble fini $X = \{x_1, x_2, ..., x_n\}$ ces éléments sont appelés sommets, et par un ensemble fini $U = \{u_1, u_2, ..., u_m\}$ ces éléments sont appelés arêtes .

Une arête u de l'ensemble U est définie par une paire non ordonnée de sommets, appelés les extrémités de u.

Si l'arête u relie les sommets x et y, on dira que ces sommets sont adjacents, ou bien que l'arête u est incidente avec les sommets x et y.

On appelle ordre d'un graphe G le nombre de sommets n de ce graphe.

Dans tout ce qui suit on suppose donné un graphe $G = (X, U)$

1.1.1 Quelques définitions sur les graphes

Graphe complet

Un graphe est dit complet si pour toute paire de sommets (x, y), il existe une arête u qui les relie. Si G est d'ordre n et est complet, alors G est noté par K_n .

Graphe simple

Un graphe est dit simple si pour tout couple de sommets (x, y) il existe au plus une arête u qui relie ces sommets.

Une arête u qui relie un sommet x à lui-même est dit boucle t s'il n'y a pas de boucle sur un sommet.

Graphe partiel et sous-graphe

Soit A ($A \subset X$) un sous-ensemble de sommets et W ($W \subset U$) un sous-ensemble d'arêtes.

-Le graphe partiel de G construit sur W est le graphe $I = (X, W)$.

-Le sous-graphe de G engendré par A est le graphe $H = (A, U(A))$ tel que

$U(A) = \{u; u \in U/$ les deux extrémités de u sont dans $A\}$.

-Le sous-graphe partiel engendré par A et W est le graphe $S = (A, W(A))$ tel que

$W(A) = \{u, u \in W/$ les deux extrémités de u sont dans $A\}$.

Degré

On définit le degré, $d_G(x)$, d'un sommet x dans G par le nombre d'arêtes qui sont adjacentes à x .

Chaines et Cycles

Une chaine, C , reliant deux sommets x et y est une séquence d'arêtes $\{u_1, u_2, ...u_m\}$ telle que : x est adjacent à u_1 ,les arêtes u_i et u_{i+1} sont adjacentes et enfin y est une extémité de u_m.

Une chaine est dite simple si pour toute arête u de U , u appartient au plus une fois à C.

Un cycle est une chaine qui relie un sommet à lui même.

La longueur

La longueur d'une chaine correspond au nombre d'arêtes qui la constituent ou à la somme des poids de ces dernières.

Graphe connexe

G est dit connexe si entre deux sommets x et y quelconques il existe une chaine qui les relient.

Graphes hamiltoniens

– Un cycle passant une et une seule fois par chacun des sommets de G est appelé cycle hamiltonien.
– Un graphe est dit hamiltonien s'il possède un cycle hamiltonien.
– Une chaîne passant une et une seule fois par chacun des sommets de G est appelé chaîne hamiltonienne.
– Les graphes complets K_n sont hamiltoniens.

1.2 Graphes orientés

Un graphe orienté $G = (X, U)$ est défini par la donné :
– D'un ensemble fini $X = \{x_1, x_2, ..., x_n\}$ dont les éléments sont appelés sommets.
– D'un ensemble fini $U = \{u_1, u_2, ..., u_m\}$ dont les éléments sont appelés arcs.
– D'une application $I : U \to X$

$$u \to I(u) \qquad \text{(extrémité initiale de } u\text{)}$$

–D'une application $T : U \to X$

$$u \to T(u) \qquad \text{(extrémité terminale de } u\text{)}$$

Chemin et circuit

Un chemin c du sommet x au sommet y est une séquence d'arcs $\{u_1, u_2, ...u_m\}$ telle que pour tout $i = 1...p-1$, $T(u_i) = I(u_{i+1})$, $I(u_1) = x$ et $T(u_m) = y$

Un circuit est un chemin reliant un sommet à lui même.

Circuit hamiltonien

Un circuit hamiltonien est un circuit qui passe exactement une fois par tous les sommets du graphe.

La longueur d'un circuit μ est la somme des longueurs des arcs qui le composent.

Distance

On appelle distance entre deux sommets d'un graphe, la longueur d'un plus court chemin les reliant. S'il n'existe pas de chemin entre les sommets x et y, on pose $d(x, y) = \infty$.

Arbre

- Un graphe $T = (X, U)$ est dit arbre si T est connexe et est sans cycles .

Soit c une fonction définie de $U \to \mathbb{R}$, qui à toute arête e on associe son poids $c(e)$.

Le poids d'un arbre T est défini par : $W(T) = \sum_{e \in U} c(e)$.

- Un arbre T de poids minimum d'un graphe G est un graphe partiel de poids minimum. Pour déterminer un arbre de poids minimum on applique l'algorithme de KRUSKAL ci-dessous :

Algorithme de Kruskal
Trier les arêtes de U par ordre de poids croissant (i.e : $c(e_1) \leq c(e_2) \leq ..... \leq c(e_m)$) Initialisation : $A \leftarrow \varnothing$ Pour $j = 1, 2, ..., m$ Si($(X, A \cup \{e_j\})$ ne contient pas de cycles) $A \leftarrow A \cup \{e_j\}$ fin si Fin pour Fin algorithme

Remarque

Il est trés facile de montrer que le graphe partiel obtenu par ce l'algorithme est un arbre de poids minimum.

Chapitre 2

Problème du voyageur de commerce

2.1 Historique

Les premières approches mathématiques exposées pour le problème du voyageur de commerce ont été traitées au 19°siècle par les mathématiciens Sir William Rowan Hamilton et Thomas Penyngton Kirkman.

Hamilton en a fait un jeu : "Hamilton's Icosian game". Les joueurs devaient réaliser une tournée passant par 20 points en utilisant uniquement les connections prédéfinies.

En 1930, ce problème est traité plus en profondeur par Karl Menger à Harvard. Il est ensuite développé à Princeton par les mathématiciens Hassler Whitney et Merril Flood, une attention particulière est portée sur les connections par Menger et Whitney ainsi que sur la croissance de ce problème.

En 1954, des solution de ce problème pour 49 villes ont été obtenues par Dantzig, Fulkerson et Johnson par une méthode de coupe.

En 1975, des Solution pour 100 villes par Camerini, Fratta and Maffioli.

En 1987, le problème à été résolu pour 532 et 2392 villes par Padberg et Rinaldi.

En 1998, des solution ont été trouvées pour 13 509 villes des Etats-Unis.

En 2001, Applegate, Bixby, Chvàtal et Cook des universités de Rice etPrinceton ont résolu le problème pour les 15 112 villes d'Allemagne.

En mai 2004, le problème du voyageur de commerce qui consiste à visiter chacune des 24.978 villes en Suède a été résolu :

Une excursion de longueur approximativement de 72.500 kilomètres a été trouvée et on a montrée l'inéxistence d'une tournée plus courte.

En mars 2005, un problème de taille égale 33.810 villes a été résolu (Cook et al. 2006).

2.2 Position du problème du TSP

Etant donné un graphe $G = (V, E)$ non orienté simple et sans boucles, le problème du voyageur de commerce consiste à déterminer un cycle hamiltonien (qu'on appelera tournée) de longueur minimale.

2.3 Formulation mathématique du problème

Soit $G = (V, E)$ un graphe non orienté que l'on peut, sans perte de généralité, supposer simple et complet.

On définit $c(e)$ comme le coût associé à l'arête e, pour toute $e \in E$ (que l'on note c_{ij} : le coût de déplacement entre les villes i et j $\forall (i,j) \in E$).

Dans tout ce qui sui on supposera que $c_{ij} = c_{ji}$ pour tout $(i,j) \in E$ (problème symétrique).

On définit la variable binaire x_{ij} et x_{ji} pour tout $(i,j) \in V^2$ par :

$$x_{ij} = \begin{cases} 1 & \text{si la ville } j \text{ est visitée juste aprés la ville } i \\ 0 & \text{sinon} \end{cases}$$

Alors notre problème peut se formuler comme suit :

$$\begin{cases} \min \sum_{1 \le i \le n} \sum_{1 \le j \le n} c_{ij} x_{ij} & & (1) \\ \sum_{\substack{i \in V \\ i \neq j}} x_{ji} = 1 & \forall j \in V & (2') \\ \sum_{\substack{j \in V \\ j \neq i}} x_{ji} = 1 & \forall i \in V & (3') \\ \sum_{\substack{i \neq j \\ i \in s}} x_{ji} \ \le |s| - 1 & \forall\ s = \{1, 2, 3, ..., n\} & (3) \\ x_{ij} \text{ et } x_{ji} \in \{0, 1\} & \forall (i,j) \in V^2 & (5') \end{cases}$$

Les contraintes $(2^{'})$ et $(3^{'})$ peuvent être remplacées par la contrainte (2) pour obtenir notre programme linéaire sous la forme suivante :

$$(P)\begin{cases} \min \sum_{1\leq i<j\leq n} c_{ij}x_{ij} & & (1)\\ \sum_{i<j} x_{ij} + \sum_{j<i} x_{ji} = 2 & \text{pour } i \in \{2,3,...,n\} & (2)\\ \sum_{i<j, ij\in S} x_{ij} \leq |S| - 1 & \text{pour } S \subset \{2,3,...,n\} & (3)\\ \sum_{j} x_{1j} = 2 & & (4)\\ \sum_{1\leq i<j\leq n} x_{ij} = n & & (5)\\ x_{ij} \in \{0,1\} & \forall i < j & (6) \end{cases}$$

(1) : fonction objectif (on minimise le cout total de la tournée).

(2) et (4) : ces contraintes signifient qu'on doit passer une et une seule fois par une ville i.

(3) : contrainte d'élimination de sous-tours.

(5) : on doit visiter toutes les villes.

(6) : contraintes d'intégrité.

Chapitre 3

Méthode de résolution

Le problème du voyageur de commerce, comme tout problème d'optimisation combinatoire, est théoriquement résolu : le graphe étant fini, donc il suffit de choisir un ensemble ordonné de n villes parmi n : il ya $n!$ possiblités. Néanmoins le problème reste entier sur le plan pratique car $factorielle\ n$ devient trés grand dès que le nombre de villes n est assez élevé d'où le recours à des procédures qui exploitent toutes les propriétés du problème.

Plusieurs méthodes de résolutions du problème (TSP) ont été proposées. Ces méthodes peuvent être classées en deux groupes :

3.1 Les méthodes approchées

Les méthodes approchées, appelées ausssi heuristiques, permettent quant à elles d'obtenir des solutions approchées en un temps raisonnable mais sans garanti d'optimalité.

Très souvent, dans le cas des problèmes de grandes taille, on préfère utiliser des méthodes approchées de moindre coût à des procédures exactes et très couteuses

Dans ce qui suit nous présentons plusieurs méthodes approchées de résolution du problème (TSP)

3.1.1 Algorithme de colonie de fourmis (ACO)

La méta-heuristique "ant colony " a été inspirée par les travaux de Deneubourg. Ce concept est relativement récent puisqu'il a commencé en 1991 avec Colorni, Dorigo et Maniezzo[1]. Il avait pour but de résoudre le problème du voyageur de commerce.

Le premier algorithme s'inspire du comportement des fourmis qui arrivent à trouver un plus court chemin entre leur fourmilière et une source de nourriture. l'idée originale s'est depuis diversifiée pour résoudre une classe plus large de problèmes.

La solution étant insatisfaisante, des améliorations ont été aportées en 1995. Cependant dès 1994 elle a pu être appliquée à d'autre problèmes d'optimisation combinatoire.

La méthode s'inspire du comportement social des fourmis.

Le comportement des fourmis est un comportement collectif, Chaque fourmi a pour priorité le bien être de la communauté.

Chaque individu de la colonie est à priori indépendant et n'est pas supervisé d'une manière ou d'une autre. Chaque individu est aidé par la communauté dans son évolution et en retour il aide au bon fonctionnement de celle-ci.

La colonie est donc autocontrôlée par le biais de mécanismes relativement simples à étudier.

En observant une colonie de fourmis à la recherche de nourriture dans les environs du nid, on s'aperçoit qu'elle "résout" des problèmes tels que celui de la recherche d'un plus court chemin.

Les fourmis "résolvent" des problèmes complexes par des mécanismes assez simples à modéliser.

Il est, ainsi, assez simple de simuler leur comportement par des algorithmes.

Les algorithmes de colonie de fourmis sont des algorithmes qui s'inspirent du comportement social des fourmis et qui constituent une famille de méta heuristiques d'optimisation.

Initialement proposé par Marco Dorigo et al[3], Dans les années 1990, pour la recherche de chemins optimaux dans un graphe, le premier algorithme s'inspire du comportement des fourmis recherchant un chemin entre leur nid et une source de nourriture.

L'idée originale s'est depuis diversifiée pour résoudre une classe plus large de problèmes et plusieurs algorithmes ont vu le jour, s'inspirant de divers aspects du comportement des Fourmis.

Fonctionnement

L'idée originale provient de l'observation de l'exploitation des ressources alimentaires chez les fourmis. En effet, celles-ci, bien qu'ayant individuellement des capacités cognitives limitées, sont capables collectivement de trouver le plus court chemin entre une source de nourriture et leur nid.

Un modèle expliquant ce comportement est le suivant :

En se déplaçant du nid à la source de nourriture et vice-versa (ce qui, dans un premier temps, se fait essentiellement de façon aléatoire), les fourmis déposent au passage sur le sol une substance odorante appelée phéromone. Cette substance permet ainsi donc de créer une piste chimique, sur laquelle les fourmis s'y retrouvent. En effet, d'autres fourmis peuvent détecter les phéromones grâce à des capteurs sur leurs antennes.

Les phéromones ont un rôle de marqueur de chemin : quand les fourmis choisissent leurs chemins, elles ont tendance à choisir la piste qui porte la plus forte concentration de phéromones. Cela leurs permet de retrouver le chemin vers leur nid lors du retour. D'autre part, les odeurs peuvent être utilisées par les autres fourmis pour retrouver les sources de nourritures déjà trouvées par leurs congénères.

Ce comportement permet de trouver un plus court chemin vers la source de nourriture lorsque les pistes de phéromones sont utilisées par la colonie entière. Autrement dit, lorsque plusieurs chemins marqués sont à la disposition d'une fourmi, cette dernière peut connaitre un plus court chemin vers sa destination. Cette constatation essentielle est la base de toutes les méthodes qui ont été développées.

A terme, l'ensemble des fourmis a donc déterminé et « choisi » la piste la plus courte.

Les fourmis utilisent l'environnement comme support de communication : elles échangent indirectement de l'information en déposant des phéromones, le tout décrivant l'état de leur « travail ».

L'information échangée a une portée locale, seule une fourmi située à l'endroit où les phéromones ont été déposées y a accès. Ce système porte le nom de « stigmergie ».

Expériences

Nous présentons trois expériences :

Pont binaire de Deneubourg[4]

L'expérience montre un nid d'une colonie de fourmis, qui est séparé d'une source de nourriture par un pont à deux voies de même longueur. On laisse évoluer les fourmis sur le pont, on trace ainsi en fonction du temps, le graphe du nombre de fourmis empruntant chaque branche. Le résultat de l'expérience est exposé à la (fig 1).

L'illustration (a) représente la configuration physique de l'expérience. Le graphique (b) indique l'évolution de ce système en fonction du temps : on constate que les fourmis ont tendance à emprunter le même chemin (par exemple celui du haut) après une dizaine de minutes.

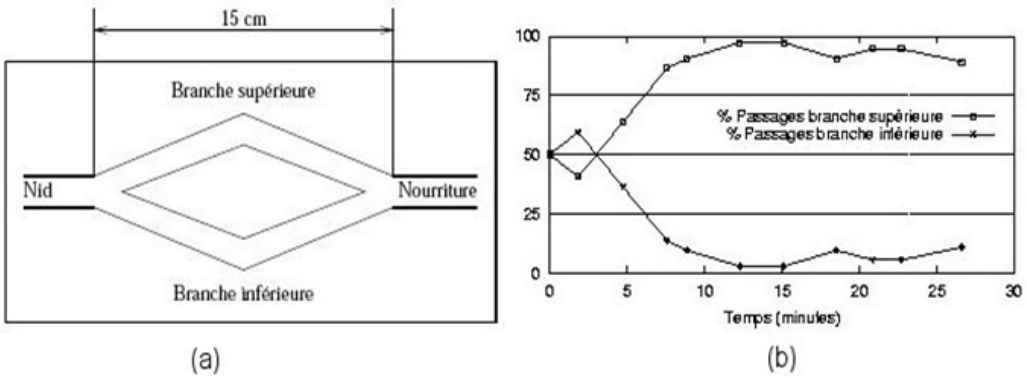

Fig.1 – Pont binaire de Deneubourg.

Explication

Au départ, il n'y a pas de phéromones sur le pont. Donc, chaque branche peut être choisie par une fourmi avec la même probabilité.

Néanmoins, dans notre exemple, après une certaine période, des variations aléatoires ont fait qu'un peu plus de fourmis ont choisi le chemin du haut plutôt que celui du bas.

Puisque les fourmis déposent des phéromones en avançant et puisqu'elles sont plus nombreuses en haut qu'en bas, le chemin du haut comportera plus de phéromones. Cette quantité

supérieure de phéromone incite plus de fourmis à choisir la branche du haut, donc la quantité de phéromone déposée augmentera encore plus.

On en déduit que plus des fourmis suivent un chemin, plus ce chemin devient intéressant à suivre. Ainsi la probabilité avec laquelle une fourmi choisit un chemin, augmente avec le nombre de fourmis qui ont pris ce chemin précédemment.

Expérience du double pont binaire [2]

On peut se demander à présent quel serait l'effet de l'augmentation de la longueur d'une des deux branches du pont. L'effet produit sera que la branche la plus courte sera sélectionnée.

En effet, les premières fourmis qui reviennent au nid avec de la nourriture sont celles qui ont emprunté le chemin le plus court dans les deux sens. Ce chemin, marqué deux fois par les phéromones, attire plus les autres fourmis que le long chemin, qui lui, est marqué une seule fois dans le sens de l'aller. Cet effet se renforce au fur du temps, jusqu'à ce que toutes les fourmis choisissent le plus court chemin.

C'est ainsi que dans cette expérience, on voit que les variations aléatoires sont réduites, puisque les deux chemins n'ont plus la même longueur. Contrairement à la première expérience, le comportement des fourmis qui consistait à suivre les pistes de phéromones n'est plus le seul mécanisme présent : maintenant on associe ce mécanisme à une notion de distance.

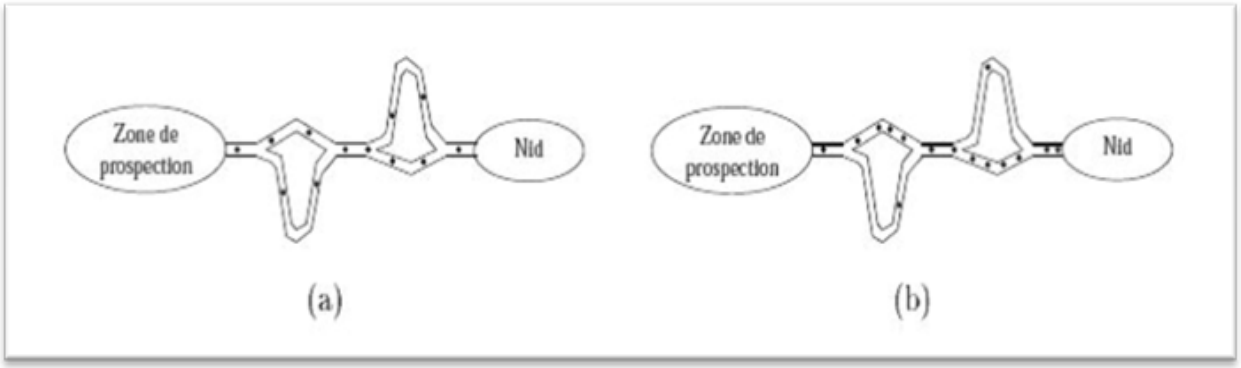

Fig. 2 – Expérience du double pont binaire.

Toutefois, quand le chemin le plus court n'est ouvert qu'après le chemin le plus long (par exemple, quand le chemin était au départ bloqué par un obstacle), les fourmis continuent de parcourir le chemin le plus long car c'est le seul à être recouvert de phéromone. C'est ainsi que, l'évaporation des phéromones joue un rôle capital : les pistes de phéromones sont moins présentes sur les chemins les plus longs, car le réapprovisionnement en phéromone demande plus de temps que sur les chemins plus courts. Donc il suffit alors qu'une poignée de fourmis choisissent le chemin auparavant bloqué pour favoriser celui-ci. Ainsi, même quand des voies les plus courtes ont été découvertes après, les fourmis finissent par les emprunter.

On peut généraliser cela à plus de deux chemins possibles : dans la figure 2 (a), on a utilisé un double pont avec quatre chemins possibles de différentes longueurs. On voit dans le dessin (b) que la plupart des fourmis finissent par choisir le chemin le plus court. Les expériences montrent que : quand cent fourmis environ ont déjà emprunté le pont, plus de 90 pourcent d'entre elles sélectionnent le chemin le plus court : les fourmis convergent donc assez rapidement.

Effet de la coupure d'une piste de phéromone

Cette fois, les fourmis sont en train de suivre une piste de phéromones, comme présenté à la figure 3 (a). A un moment donné, on a un obstacle qui barre la route des fourmis. Les fourmis qui arrivent à côté de l'obstacle doivent choisir soit d'aller à gauche ou à droite (b).

Puisqu'aucune quantité de phéromones n'est déposée le long de l'obstacle, il y a autant de fourmis qui partent à gauche et à droite.

Le chemin de droite est plus court que celui de gauche, les fourmis qui l'empruntent, auront tendance à emprunter la piste de phéromone de départ. Pour chaque fourmi allant du nid à la source de nourriture, on associe également une fourmi qui va de la nourriture au nid (en fait elles ont été séparées par l'apparition brutale de l'obstacle). Les phéromones de ces fourmis vont se superposer à droite. Donc quand elles vont rejoindre le chemin initial, le chemin de droite sera deux fois plus imprégnée de phéromone que la piste de gauche, où les deux fourmis

n'ont pas encore pu rejoindre la piste initiale (ce chemin étant plus long).

Les fourmis qui arrivent à l'obstacle à partir de ce moment, préféreront suivre la piste de droite. Le nombre de fourmis qui passent par la droite va augmenter, ce qui augmentera encore la concentration de phéromones. De plus, l'évaporation des phéromones sera plus forte sur la piste de gauche du fait que sa longueur est supérieure. La piste de gauche sera dont rapidement abandonnée, parce qu'elle en est beaucoup moins imprégnée : les fourmis passeront toutes très rapidement par la piste la plus courte.

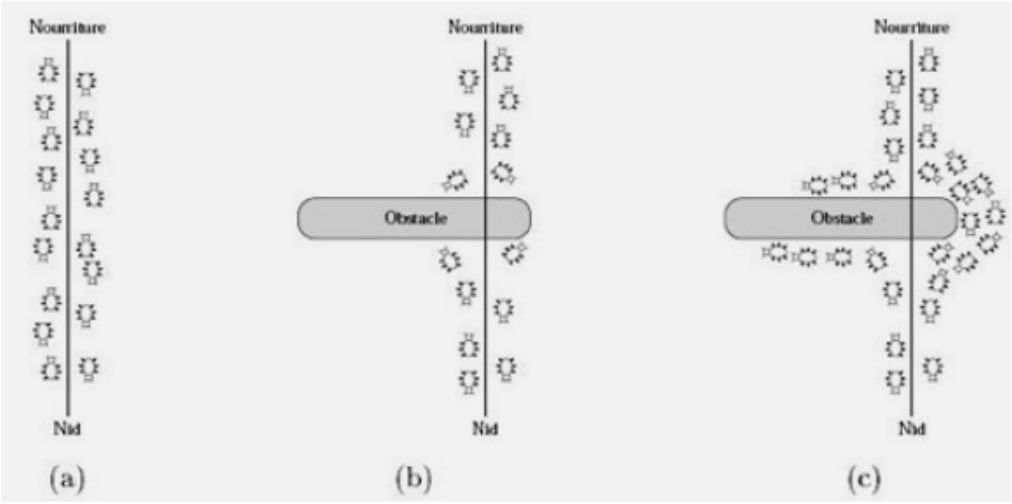

Fig.3 – Effet de la coupure d'une piste de phéromone

Remarque 3.1.1 *Il est intéressant de remarquer que bien qu'une seule fourmi soit capable de construire une solution (i.e. de trouver un chemin du nid à la source de nourriture), c'est seulement le comportement de l'ensemble de la colonie qui crée le chemin le plus court*

Application de l'algorithme de colonie de fourmis au problème (TSP)

Dans l'algorithme de colonie de fourmis, chaque fourmi est initialement placée sur une ville (sommet de graphe),chacune possède une mémoire qui stocke la solution partielle qu'elle a construite auparavant.

Soient $b_i(t)$ l'ensemble des fourmis dans la ville i au temps t tel que $i = 1, ..., k$ et $n = \sum_{i=1}^{k} b_i(t)$ le nombre total des fourmis.

Chaque fourmis est un agent avec les caractéristiques suivantes :

La fourmi choisit la prochaine ville de destination avec une probabilité qui est en fonction de la distance et de la quantité des phéromones présente sur l'arête de connexion. Cette probabilité est formulée à travers la règle de déplacement suivante :

$$p_{ij}^{k}(t) = \begin{cases} \frac{(\tau_{ij}(t))^{\alpha}.(\eta_{ij})^{\beta}}{\sum_{l \in N^k}(\tau_{il}(t))^{\alpha}.(\eta_{il})^{\beta}} & \text{si } \; j \in N^k \\ 0 & sinon \end{cases} \qquad (R1)$$

avec :

$\tau_{ij}(t)$ est l'intensité de la trace de phéromones dans l'arête (i, j) à l'instant t.

$\eta_{ij} = \frac{1}{d_{ij}}$ "visibilité" : une information heuristique à priori valable, ou d_{ij} la distance entre la ville i et la ville j,l'idée étant d'attirer les fourmis vers les villes les plus proches.

α, β sont deux paramètres positif non nul qui déterminent l'influence relative à la trace de phéromones et de l'information heuristique.

N^k est le voisinage faisable de la fourmis k . Il représente l'ensemble des villes qui restent à visiter. Celles déja visitées sont retirées de la liste .

La construction de la solution s'achève lorsque chaque fourmi complète un tour et dépose une quantité de phéromones sur toutes les arêtes (i, j) visitées. Au temps t, elle choisit la prochaine ville où elle s'y positionnera au temps $t + 1$. Afin d'accomplir leurs tours, les fourmis effectuent n itération et les traces des phéromones sont mises à jour selon la formule ci-dessous :

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \Delta\tau_{ij}(t) \qquad (R2)$$

Où,

ρ est un coefficient tel que $0 < \rho < 1$

$(1 - \rho)$ est l'évaporation de la phéromone entre le temps t et $t + n$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{n} \Delta\tau_{ij}^{k}(t)$$

représente la quantité des phéromones par unité de longeur déposée sur l'arête (i, j) par la $k^{ème}$ fourmi entre t et $t + n$, elle est donnée par :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L_k} & \text{si } (i,j) \in Tabou^k \\ 0 & \text{sinon} \end{cases} \qquad (R3)$$

Avec,

L_k :la longeur du tour de la $k^{ème}$ fourmi.

Q : un nombre positif constant.

$Tabou^k$: la liste des villes déjà visitées par la fourmi k.

Algorithme de colonie de fourmis

Début
- Donner $nville$ (le nombre de villes)
- Déterminer la matrice des distances ou la matrice des coûts $d(i, j)$
- Choisir le nombre de fourmis $nfourmi$
  - $nfourmi \longleftarrow nville$
- Initialisation $\alpha, \beta, Q, \rho, \tau_{ij}(0) = c(avec\ c > 0)$
- Donner la solution optimale (sol_opt) pour calculer l'erreur

- pour $i = 1$ à $nville$ faire
  - Pour $j = 1$ à $nville$ faire
    - Si $i \neq j$
      - $\eta_{ij} \leftarrow 1/d_{ij}$
      - $\tau_{ij} \leftarrow c$
    - sinon
      - $\tau_{ij} \leftarrow 0$
    - Fin si
  - Fin pour
- Fin pour

pour t= 1 à $t_{\max}$
Placer aléatoirement les fourmis sur les n sommets du graphe
pour chaque fourmi k ($k = 1$ à $nfourmi$)
initialiser la liste $Tabou^k$ comme suit :
$Tabou^k \leftarrow$sommet affecté
pour $i = 1$à $nville$ faire
Pour $k = 1$ à $nfourmi$ faire
N^k : ensemble des villes non visitées par la fourmi k.
A chaque pas, la fourmi k située dans une ville i choisira une ville j
du voisinage possible N^k (villes non visitées) selon la règle de déplacement $R1$
Fin pour
Actualiser la liste $Tabou^k$ (villes visitées par la fourmi k)
$Tabou^k \leftarrow Tabou^k \cup \{j\}$
Fin pour
pour $k = 1$ à $nfourmi$ faire
Calculer le coût de la tournée L^k (égal à la somme des poids des arêtes qui la composent L^k)
Pour $i = 1$ à $nville$ faire
déposer une piste $\Delta\tau_{ij}^k(t)$ sur le trajet $Tabou^k$ conformément à l'équation $R3$
Fin pour
Fin pour
Mettre à jour les traces de phéromone selon la règle $R2$
$L \leftarrow \min(L^k)$ on trouve le longueur minimum pour chaque tour de chaque fourmi
$gap \leftarrow (L - sol\_opt)/sol\_opt$
Afficher (le nombre d'itération t, le coût de la tournée L et l'erreur gap)
si $gap = 0$
Stop
Fin si
Fin pour
Fin pour
Fin de l'algorithme.

3.1.2 Algorithme du plus proche voisin (PPV)

L'algorithme du plus proche voisin (PPV) est trés simple, c'est l'un des premiers algorithmes utilisés pour déterminer une solution du problème (TSP).L'algorithme du (PPV) est de type glouton et consiste à :

Au départ, on choisit une première ville au hasard puis on construit un chemin allant de cet ville vers la ville la plus proche (au sens des distances). supposons qu'à l'étape k on ait déja choisi k ville et soit a_k la dernière ville choisie alors on détermine la ville la plus proche (parmi les villes non encore choisies) de a_k . on continue le processus jusqu'à épuisement de tous les sommets. Ainsi, on construit un cycle hamiltonien par mémorisation de tous les plus coûts chemins.

Soit l'algorithme de manière formelle :

Algorithme
Début
choisir un sommet $v_1 \in V$.
poser $k \leftarrow 1$
poser : $U = \{v_1\}$.
tant que $k < n$
$k \leftarrow k + 1$.
choisir v_k dans $X = V \backslash U$.qui vérifie :
$d(v_{k-1}, v_k) = \min_{x \in X} \{d(v_{k-1}, x)\}$
faire :$U \leftarrow U. \cup \{v_k\}$
fin tant que.
Fin de l'algorithme.

On prend l'exemple de 5 villes avec la matrice des distances suivante :

$$d = \begin{bmatrix} - & 11 & 17 & 21 & 12 \\ 11 & - & 11 & 22 & 14 \\ 17 & 11 & - & 19 & 10 \\ 21 & 22 & 19 & - & 9 \\ 12 & 14 & 10 & 9 & - \end{bmatrix}$$

On applique la méthode du plus proche voisin pour obtenir une tournée (i.e. le cycle hamiltonien)

1. Etape 1 : (on choisit le sommet 1 comme sommet de départ et on pose : $U = \{1\}$) On calcule le minimum de la première ligne : $\min d(1, j) = 11 = d(1, 2)$
2. On actualise U $(U \leftarrow \{1, 2\})$
3. On calcule $\min_{x \notin U} \{d(2, x)\} = d(2, 3)$
4. On actualise U $(U \leftarrow \{1, 2, 3\})$
5. A la fin on trouve la tournée :

$$1 - 2 - 3 - 5 - 4 - 1$$

avec un coût :

$$Z = 62$$

3.2 Les méthodes exactes

Le problème du (TSP) formulé comme un programme linéaire à variables bivalentes n'est pas différentiable, donc on ne peut lui appliquer la méthode du simplexe ou autres méthodes utilisées dans l'optimisation différentiable.

Les méthodes exactes de résolution du problème du (TSP) sont basées sur des procédures par séparation et évaluation .

Ces méthodes permettent l'obtention de solutions optimales. Elles sont intéressantes dans le cas de problème de taille raisonnable. Pour des problèmes de taille assez élevée ces méthodes sont exécutées pendant un certain nombre (fixé à l'avance) et sont donc utilisées comme heuristiques.

Pour résoudre le problème du (TSP) on relaxe (on supprime) les contraintes de type $(2')$ pour obtenir un problème (dit relaxé) qu'on résout habituellement par des procédures du sous gradient. Dans notre travail, nous avons introduit une méthode du sous gradient de remplacement (surrogate subgradient) afin d'accélérer la procédure.

3.2.1 Relaxation lagrangienne

La relaxation lagrangienne s'articule autour de l'idée qui consiste à relâcher les contraintes difficiles, non pas en les supprimant , mais en les prenant en compte dans la fonction objectif de sorte qu'elles pénalisent la valeur des solutions qui violent ces dernières.

Soit (P1) le programme mathématique suivant :

$$(P1) : \begin{cases} \min z = f(x) \\ g(x) \leq 0 \\ x \in S \subset \mathbb{R}^n \end{cases}$$

Supposons que la contrainte $g(x) \leq 0$ soit difficile à satisfaire et S un ensemble fini discret ayant une structure spéciale telle que l'on connaisse un "bon" algorithme pour résoudre le problème dit relaxé suivant :

$$(L_\pi) \begin{cases} \min f(x) + \pi.g(x) \\ x \in S \end{cases}$$

On associe à chaque contrainte i ; $g_i(x) \leq 0$ une variable duale $\pi_i \geq 0$ (les coefficients

$\pi_1^k, \pi_2^k, ...$et π_n^k sont appellés multiplicateurs de lagrange). Nous définissons la fonction duale :

$$L(\pi) = \min_{x \in S} \left(f(x) + \pi.g(x) \right) \tag{3.2.1}$$

La fonction duale L est en général une fonction concave (enveloppe inférieure d'une famille de fonction linéaire affine) continue mais non différentiable.

Nous devons résoudre alors le problème duale de (L_π) :

$$(D) \begin{cases} L(\pi^*) = \max L(\pi) \\ \pi \geq 0 \end{cases}$$

3.2.2 Méthode du sous-gradient"SG"

Les méthodes du sous-gradient sont fréquemment utilisées pour résoudre les problèmes d'optimisation non différentiables. Elles ont été introduites par Shor. Les premiers résultats de convergence sont dus à Ermol'ev,Polyak ou Nemirovskii
Shor a présenté un aperçu général des premiers développements sur les méthodes du sous-gradient[5].

Ces méthodes s'apparentent aux algorithmes du gradient dans le cas différentiable, en effectuant un pas dans la direction opposée au sous-gradient.
a- soit un vecteur $\gamma = (\gamma_1, \gamma_2, ..., \gamma_m)^t$. Il est dit sous-gradient de L au point π si :

Définition 3.2.1

$$\forall \pi' \in \mathbb{R}_+^m \; L(\pi') - L(\pi) \leq (\pi' - \pi) \, .\gamma$$

b- l'ensemble des sous-gradients au point π (noté $\partial L(\pi)$) est appelé sous-différentiel de L au point π.

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

$$\pi^{k+1} = \text{Proj}\left(\pi^k + s_k.\xi^k\right) \tag{3.2.2}$$

où les coeffficients $\{\pi_j^k\}$ sont les multiplicateurs de Lagrange et ξ^k est le sous gradient de L au point π^k, s_k est le pas, et Proj est l'operateur projection de $\mathbb{R}^m$ sur $\mathbb{R}_+^m$ c'est-à-dire

$$\text{Proj}(\pi) = \max(0, \pi) = (\max(0, \pi_1), ..., \max(0, \pi_m))^T$$

Algorithme	
Etape 0 :	(initialisation) Choisir π^0
Etape 1 :	à l'itération k $L(\pi^k) = f(y^k) + \pi^k.g(y^k) = \min x \in S\{f(x) + \pi^k.g(x)\}$ définir π^{k+1} par : $\pi^{k+1} = \text{Proj}\left(\pi^k + s^k\xi^k\right)$ et aller à l'étape 1

$$\text{où} \qquad s^k = \rho\frac{L(\pi^*) - L(\pi^k)}{\left\|\xi^k\right\|^2} \tag{3.2.3}$$

le coéfficient de relaxation ρ vérifie : $0 < \rho < 2$.

$L(\pi^*)$ n'étant pas connue, si l'on remplace dans (3.2.3) $L(\pi^*)$ par une estimation par défaut $L_1 \leq L(\pi^*)$ polyak 1969 a montré :

- $L(\pi)$ converge vers $L(\pi^*)$
- ou bien, en un nombre fini d'itération on obtien un point π^j vérifiant : $L_1 \leq L(\pi^j) \leq L(\pi^*)$

En pratique, on choisit pour L_1 la valeur de $f(x)$ correspondant à la meilleure solution de (P) obtenue durant les étapes précédentes du calcul.

choix des coefficients ρ

plusieure stratégies peuvent être envisagées. Help et al (1974) ont rapporté une expérience satisfaisante avec la règle R1 : legendre et minoux en 1977 ont déterminé les coefficients ρ de façon dynamique en tenant compte de la progression de la fonction L pour proposer la règle R2.

- R1 : Prendre $\rho = 2$ pendant les m (nombre de composantes de π) premières itérations ;puir diviser par 2 la valeur de ρ et le nombre d'itérations jusqu'à atteindre une limite inferieure q (fixée à l'avance) du nombre d'itérations ; enfin diviser par 2 la valeur de ρ toutes les q itérations (généralement $q \simeq 5$) jusqu'à ce que les π^k soient suffisamment petits (ε fixé).
- R2 : au départ, faire $\rho = 2$;

si $L(\pi^k) > \min\{L(\pi^{k-r}), r = 1, 2, ..., k\}$ alors $\rho_{k+1} = \rho_k$

sinon $\rho_{k+1} = \alpha.\rho_k$(où α vérifie : $0 < \alpha < 1$)

Plusieurs remarques sur les limitations de ces méthodes peuvent être faites :

1. On ne dispose pas de test d'arrêt naturel permettant de détecter que l'on est suffisamment proche de la solution.

2. Ces méthodes ne garantissent pas la descente de l'objectif entre deux itérés consécutifs, pouvant mener à des phénomènes oscillatoires et à de mauvaises performances numériques.
3. Les méthodes de mise à jour des longueurs de pas les plus efficaces sont parfois dépourvues de résultats de convergence.
4. La vitesse de convergence est souvent faible, en particulier lorsque l'on s'approche de la solution.

Malgré ces limites, ces méthodes restent très largement utilisées par les praticiens, notamment dans la communauté combinatoire.Ce succès est du en partie à leur simplicité d'implémentation, mais également à la qualité de leur comportement numérique pour des problèmes de grande taille.

Application de la méthode du sous- gradient au problème du (TSP)

En relaxant les contraintes de (P) de type (2) on obtient le programme relaxé (L_π) suivant :

$$(L_\pi)\begin{cases} \min L(\pi) = \displaystyle\sum_{1\le i<j\le n} c_{ij}x_{ij} + \sum_{i=2}^{n-1}\pi_i(2-(\sum_{i<j}x_{ij}+\sum_{j<i}x_{ji})) & \\ \displaystyle\sum_{i<j,ij\in S^2} x_{ij} \le |S|-1 & \text{pour } S\subset\{2,3,...,n\} \\ \displaystyle\sum_{j} x_{1j} = 2 & \\ \displaystyle\sum_{1\le i<j\le n} x_{ij} = n & \\ x_{ij}\in\{0,1\} & \forall i<j \end{cases}$$

La fonction objectif L peut s'écrire :

$$L(\pi) = 2\sum_{i=2}^{n-1}\pi_i + \sum_{1\le i<j\le n}(c_{ij}-\pi_i-\pi_j)x_{ij}$$

On peut facilement voir que le vecteur (ξ^k) suivant est un sous-gradient de L au point π.

$$\xi^k = 2-(\sum_{i<j}x_{ij}+\sum_{j<i}x_{ji})$$

L'algorithme du sous -gradient pour le dual lagrangien est donné par :

L'algorithme de sous gradient pour le dual lagrangien	
Etape 0 :	(initialisation)
	Choisir π^0
Etape 1 :	
	Résoudre le problème lagrangien
	$(L_{\pi^k}) = \min_{x\in X} \sum_{1\leq i<j\leq n} (c_{ij} - \pi_i^k - \pi_j^k)x_{ij} + 2\sum_{i=2}^{n-1} \pi^k$
	Calcul de x^k, (on utilise la méthode du 1-arbre)
	calculer $\xi^k = 2 - (\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji})$
	si $\xi^k = 0$ stop (π^k est la solution optimal de (D))
Etape 2 :	calculer :
	$\pi^{k+1} = \text{Proj}\left(\pi^k + s^k\xi^k\right)$
	où $s^k > 0$, le pas $\quad s^k = \rho\frac{L(\pi^*)-L(\pi^k)}{\left\|\xi^k\right\|^2}$, $\ 0 < \rho < 2$
Etape 3 :	
	poser : $k = k + 1$ et aller à l'étape 1

Exemple On suppose que la solution optimal n'est pas connu.

On prend l'exemple de 5 villes avec la matrice de distance suivante :

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

On applique la méthode du plus proche voisin pour obtenir une borne superieure (zub) d'où :

$$zub = 148$$

et on a trouvé le tour $1-2-3-4-5-1$ de coût 148

$$\begin{aligned}
\pi^{k+1} &= \pi^k + s^k.\xi^k \\
\pi^{k+1} &= \pi^k + \rho\frac{z_{ub} - L(\pi^k)}{\left\|\xi^k\right\|^2}.\xi^k \\
\pi^{k+1} &= \pi^k + \frac{z_{ub} - L(\pi^k)}{\left\|2 - \sum_{i<j} x_{ij} + \sum_{j<i} x_{ji}\right\|^2}\left(2 - (\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji})\right)
\end{aligned}$$

Itération 1 : $\pi^1 = (0, 0, 0, 0, 0)$

$$c_{ij} = (c_{ij} - \pi_i^1 - \pi_j^1) = \begin{bmatrix} - & 30 & 26 & 50 & 40 \\ 30 & - & 24 & 40 & 50 \\ 26 & 24 & - & 24 & 26 \\ 50 & 40 & 24 & - & 30 \\ 40 & 50 & 26 & 30 & - \end{bmatrix}$$

puis on applique le kruskal pour obtenir l'arbre couvrant minimal

$$\begin{aligned}
(2, 3) &= 24 \\
(3, 4) &= 24 \\
(3, 5) &= 30
\end{aligned}$$

et on cherche les 2 minimum de sommet 1

$$\begin{aligned}
\min 1 &= 26 \\
\min 2 &= 30
\end{aligned}$$

d'où

$$L(\pi^1) = 130$$

$$\begin{aligned}
\xi^1 &= 2 - (2, 2, 4, 1, 1) = (0, 0, -2, 1, 1) \\
\pi^2 &= (0, 0, -6, 3, 3)
\end{aligned}$$

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

$$
\begin{aligned}
c_{ij} &= (c_{ij} - \pi_i^2 - \pi_j^2) = \begin{bmatrix} - & 30 & 32 & 47 & 37 \\ 30 & - & 30 & 37 & 47 \\ 32 & 30 & - & 27 & 29 \\ 47 & 37 & 27 & - & 24 \\ 37 & 47 & 29 & 24 & - \end{bmatrix} \\
z_{ub} &= 148 \\
L(\pi^2) &= 143 \\
\pi^3 &= (0, 0, -8.5, 3, 5.5)
\end{aligned}
$$

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

$$
\begin{aligned}
c_{ij} &= (c_{ij} - \pi_i^3 - \pi_j^3) = \begin{bmatrix} - & 30 & 34.5 & 47 & 34.5 \\ 30 & - & 32.5 & 37 & 44.5 \\ 34.5 & 32.5 & - & 29.5 & 29 \\ 47 & 37 & 29.5 & - & 21.5 \\ 34.5 & 44.5 & 29 & 21.5 & - \end{bmatrix} \\
z_{ub} &= 148 \\
L(\pi^3) &= 147.5
\end{aligned}
$$

3.2.3 Méthode du sous-gradient de remplacement"SSG"(surrogate subgradient)[7]

La méthode de sous-gradient nécessite la minimisation de tous les sous-problèmes pour obtenir une direction de déplacement, et cela peut prendre beaucoup de temps notamment pour des problèmes de grande taille. Il est donc souhaitable d'obtenir une bonne direction au moindre coût.

L'idée principale de la méthode du sous-gradient de remplacement est d'obtenir une direction de déplacement sans optimiser tous les sous-problèmes.

Le dual de remplacement est défini par :

$$\tilde{L}(\pi, x) = \sum_{1 \le i < j \le n} c_{ij} x_{ij} + \sum_{i=2}^{n-1} \pi_i (2 - (\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji})) \tag{3.2.4}$$

où $\pi = \{\pi_1 \pi_2, ..., \pi_n\} \in \mathbb{R}_+^n$ est un vecteur des multiplicateurs de lagrange.
(Ici on ne minimise pas $\tilde{L}$ pour obtenir plus loin un sous-gradient de remplacement)

Le sous-gradient de remplacement correspondant est défini comme suit :

$$\tilde{\xi}^k = 2 - (\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji}) \tag{3.2.5}$$

On montrera plus loin que $\tilde{L}(\pi, x) \leq L^*$ (où $L^* = V(D)$)

Proposition

Étant donné un point (π^k, x^k) , si la valeur de l'objectif du dual de remplacement est inférieur à la valeur optimale du dual, i.e,

$$\tilde{L}^k = \tilde{L}(\pi^k, x^k) < L^* \tag{3.2.6}$$

alors le sous-gradient de remplacement vérifie :

$$0 \leq L^* - \tilde{L}^k \leq \left(\pi^* - \pi^k\right) \tilde{\xi}^k \tag{3.2.7}$$

Démonstration

D'après la définition du dual de remplacement on a :

$$\tilde{L}(\pi, x^k) = \sum_{1 \leq i < j \leq n} c_{ij} x_{ij}^k + \sum_{i=2}^{n-1} \pi_i (2 - (\sum_{i<j} x_{ij}^k + \sum_{j<i} x_{ji}^k)) = \tilde{L}^k + \left(\pi^* - \pi^k\right) (2 - (\sum_{i<j} x_{ij}^k + \sum_{j<i} x_{ji}^k)) \tag{3.2.8}$$

En utilisant (3.2.1) on en déduit :

$$L(\pi) \leq \tilde{L}(\pi, x) \tag{3.2.9}$$

Ce qui précède est également vrai au point (π^*, x^k),i.e :

$$L^* = L(\pi^*) \leq \tilde{L}(\pi^*, x^k)$$

(3.2.8) entraine :

$$L^* \leq \tilde{L}^k + \left(\pi^* - \pi^k\right) \tilde{\xi}^k$$

Compte tenu de (3.2.6) et (3.2.5), on obtient

$$0 \leq L^* - \tilde{L}^k \leq \left(\pi^* - \pi^k\right) \tilde{\xi}^k$$

D'où (3.2.7) est démontré.

Procédure de détermination d'une solution

Soit (π^{k+1}, x^k), (X, A) l'arbre donné par l'algorithme du kruskal. Pour déterminer x^{k+1} / $\tilde{\xi}(x^{k+1})$ soit un sous-gradient de déplacement il suffit de permuter une arête u ($c(u) = \max_{v \in A}(c(v))$)de l'arbre qui soit de poids maximum par une arête e adjacente à u / $c(e) < c(u)$

Sinon on prend $x^{k+1} = x^k$

Soit l'algorithme de façon formelle

L'algorithme
début
(π^{k+1}, x^k), (X, A) l'arbre donné par l'algorithme du kruskal
$x^{k+1} \leftarrow x^k$
déterminer u / $c(u) = \max(c(v))$
trouver $e \in U - A$ / $c(e) < c(u)$
si (e a été trouvé)
$A \leftarrow A - \{u\} \cup \{e\}$
Actualiser x^{k+1} ($x_u = 0$ et $x_e = 1$)
Fin si
Fin l'algorithme

Méthode du sous-gradient de remplacement

D'aprés la proposition précédente, si l'optimisation appropriée est effectuée de sorte que (3.2.6) soit satisfaite d'une itération à la suivante et (3.2.7) est garanti. Cela implique que l'algorithme peut trouver une bonne direction, et la distance entre les solutions courantes et le multiplicateur optimale peut être diminuée étape par étape. Cette idée conduit à la méthode de sous-gradient de remplacement, où seule une solution approchée est requise pour obtenir un sous-gradient de remplacement. Les étapes de base de la méthode de sous-gradient de remplacement sont :

L'algorithme du sous-gradient de remplacement (SSG)

Étape 0 :

On applique une itération de (SG) pour obtenir x^0,$V(L(\pi^0)$,$\xi(\pi^0)$,π^1.

$$x^0 = \arg\min \left[\sum_{1\leq i<j\leq n} c_{ij}x_{ij} + \sum_{i=2}^{n-1} \pi^0(2 - (\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji})) \right]$$

Étape 1 : A l'étape k : soit (π^k, x^k) alors $\tilde{L}^k$ est donné par :

$$\tilde{L}^k = \tilde{L}(\pi^k, x^k) = \sum_{1\leq i<j\leq n} c_{ij}x_{ij} + \sum_{i=2}^{n-1} \pi^k(2 - (\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji}))$$

calculer π^{k+1} =Proj$(\pi^k + s^k\tilde{\xi}^k)$

(s^k vérifie : $0 < s^k < \rho\frac{L^* - \tilde{L}^k}{\left\|\tilde{\xi}^k\right\|^2}$ où $\rho \in [0, 1]$).

puis : $\tilde{\xi}^k = 2 - (\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji})$

si $\tilde{\xi}^k = 0$ stop (la solution est optimale)

Étape 2 : Déterminer x^{k+1} (une solution approchée par la procédure1)

$\tilde{L}(\pi^{k+1}, x^{k+1}) < \tilde{L}(\pi^{k+1}, x^k)$ avec :

$$\tilde{L}(\pi^{k+1}, x^{k+1}) = \sum_{1\leq i<j\leq n} c_{ij}x_{ij}^{k+1} + \sum_{i=2}^{n-1} \pi^{k+1}(2 - \sum_{i<j} x_{ij}^{k+1} + \sum_{j<i} x_{ji}^{k+1})$$

Étape 3 :

poser : $k = k + 1$ et aller à l'étape 1

Remarque

– Pour le choix du coefficient ρ, On le prendra égal à 1 pendant les n premières itérations puis nous suivons la même stratégie qui a été utilisée dans la procédure(SG).

– L^* n'étant pas connu au départ, en pratique on le remplace par Zub et $\tilde{L}^k$ par Zlb (Zlb =max($\tilde{L}^j$: j=1, 2,....., k)

Proposition

le dual du sous-gradient de remplacement est toujours inférieure au dual optimale

$$\tilde{L}^k < L^*, \text{ pour tout } k \tag{3.2.10}$$

Théoreme

la suite $\left\{\pi^k\right\}_k$ converge vers π^*

$$\left\|\pi^* - \pi^{k+1}\right\| < \left\|\pi^* - \pi^k\right\|, \text{ pour tout } k \tag{3.2.11}$$

Remarque 3.2.1 *On peut facilement démontrer que si* $\pi^k = \pi^{k+1}$ *et* $x^k = x^{k+1}$*, alors* $L(\pi^k) = L^*$*, et* x^k π^k *est la solution optimale du problème dual.*

Trés souvent, pour pouvoir appliquer une méthode de résolution quelconque et notamment une méthode exacte, la connaissance d'une bonne solution de départ est requise d'où le recours à des méthodes approchées et peu couteuses. On peut utiliser la méthode du plus proche voisin ou la métaheuristique de colonie de fourmis.

Chapitre 4

tests numeriques

Afin de pouvoir apprécier l'efficacité des différentes méthodes et de comparer les méthodes entre elles, nous avons programmé toutes les méthodes citées ci-dessus en langage Matlab, et utilisé des instances tirées de OR-library[10,11,12]sur un PC Dell inspiron N5040 : Processeur Intel(R) Pentium(R) CPU P6200 @ 2.13GHz 2.13GHz et RAM 2.00 Go.

Pour déterminer la matrice des plus courtes distances dans le graphe, nous avons utilisé l'algorithme de Dijkstra (DA) pour sa simplicité et son efficacité.

Notre but est de comparer deux méthodes qui nécessitent l'exécution de la même procédure de recherche des plus courtes distances donc nous n'avons pas jugé nécessaire de comptabiliser le temps CPU de l'algorithme (DA) dans le calcul des temps CPU des procédures (SG) et (SSG). Pour toutes les procédures, nous notons par :

- sol_opt : la valeur optimal
- Z_{ub} : borne supérieure
- Z_{lb} : borne inferieure
- $iter$: nombre d'itérations
- CPU : temps CPU en secondes
- nbv :nombre des villes
- Z_{ACO} : valeur de la solution donnée par la méthode (AC)
- Z_{ppv} :valeur de la solution donnée par la méthode (PP)
- ACO : Méthode du colonie de fourmis
- PPV : Méthode du plus proche voisin

4.1 1^{er} cas : pour sol_opt connu

Nous avons testé et comparé la méthode du sous-gradient de remplacement et la méthode du sous-gradient. Nous avons utilisés neuf exemples tirés de 'OR-Library'. Les tests numériques confirment l'efficacité de la méthode du sous-gradient de remplacement par rapport à la méthode du sous-gradient. La procédure (SSG) nous fournit des valeurs proches des valeurs optimales en des temps raisonnables. la procédure (SSG) donne de meilleurs résultats que ceux qui sont founies par la méthode (SG) comme le montre le tableau1, figure1 et figure2

			Sous-gradient			Sous-gradient de remplacement			mixte		
problème	nbv	Sol_opt	zlb	iter	CPU(s)	zlb	iter	CPU(s)	zlb	iter	CPU(s)
gr17	17	2085	2085	35	8.169987	2085	19	4.074343	2085	88	6.37864
gr21	21	2707	2707	26	6.861040	2707	19	4.200957	2707	36	4.97593
dantzig42	42	699	699	178	53.26958	699	18	4.997645	699	181	64.8032
berlin52	52	7542	7542	217	160.4611	7542	22	5.609543	7542	89	77.9285
brazil58	58	25395	25354	240	220.7828	25395	19	12.65735	25369	238	288.4853
eil101	101	629	611	281	780.7431	629	16	17.68243	629	39	435.4501
ch130	130	6110	6099	301	2012.010	6110	17	35.47865	6101	280	1833.123
ch150	150	6528	6520	329	12096.23	6528	17	59.83987	6520	321	11012.11
a280	280	2579	2562	402	26099	2579	25	817.1103	2570	398	25991.58

Tableau 1 : sol_opt connu

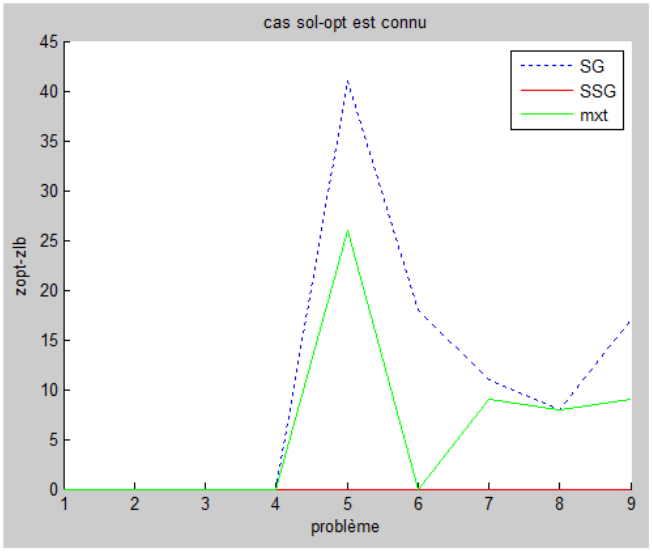

Figure 1

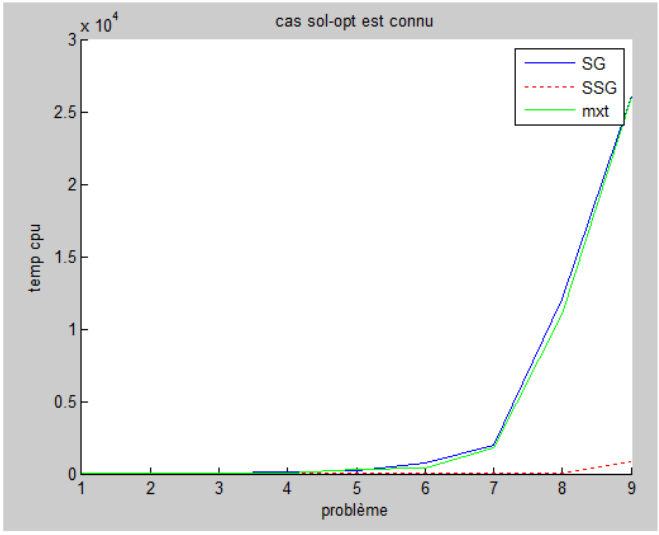

Figure 2

4.2 $2^{ème}$ *cas : pour* sol_opt *inconnu*

Ici on considère le cas qui nous intéresse le plus à savoir la comparaison entre l'algorithme du sous-gradient et l'algorithme du sous-gradient de remplacement quand la valeur optimale est inconnue. La valeur optimale sol_opt n'étant, en général, pas connue au préalable ,en pratique, elle est remplacé par Z_{ub} (on applique la méthode du proche voisin), une évaluation par excès.

La difficulté dans la méthode du sous-gradient de remplacement, quand sol_opt n'est pas connue à l'avance, réside dans la façon de choisir le coefficient de relaxation ρ (l'étape 2 dans l'Algorithme 2) ; si ρ est choisi assez petit, l'algorithme peut converger vers une mauvaise estimation par défaut de sol_opt. Au contraire, si la valeur de ρ est choisie relativement élevée, l'algorithme peut diverger. Cependant si on arrive à trouver un bon choix (ou un compromis) nous aurons l'avantage majeur de pouvoir réduire de manière significative le phénomène de zigzag qui est trés présent dans la méthode du sous-gradient.

Pour palier aux inconvénients des deux méthodes citées ci-dessus et afin de profiter des avantages de l'une et de l'autre nous avons pensé à combiner les deux méthodes : cela consiste à exécuter quelques itérations de la procédure du sous-gradient de remplacement et de terminer avec la méthode classique du sous-gradient,cette méthode nous avons appelé "mixte", comme le montre le tableau2, figure3 et figure4

			Sous-gradient				Sous-gradient de remplacement				mixte			
problème	nbv	Sol_opt	zlb	zub	iter	CPU(s)	Zlb	zub	iter	CPU(s)	zlb	zub	iter	CPU(s)
gr17	17	2085	2085	2085	35	10.4410	2085	2085	3	5.4851	2085	2085	46	6.3645
gr21	21	2707	2707	2707	26	6.896622	2707	2707	5	5.1245	2707	2707	50	4.3467
dantzig42	42	699	679.08	710.40	178	65.1932	698.12	700.0135	149	5.3920	698.89	700.013	176	71.6055
berlin52	52	7542	7520	7544	202	174.5432	7541.7	7544	184	6.2345	7523.7	7544	205	172.41
brazil58	58	25395	25354	25619	235	282.234	25360,4	25400	118	7.1593	25354	25601	231	216.34
eil101	101	629	611.34	635.13	301	900.211	630.01	642.3098	204	19.6754	617.11	635.13	235	750.345
ch130	130	6110	6099	6125	351	2050.12	6102.7	6110.9	262	40.5987	6106.11	6110.9	267	1911.99
ch150	150	6528	6498.93	6550.1	391	12190.55	6495.7	6532.3	302	66.9971	6501.82	6550.06	405	11211.23
a280	280	2579	2516.31	2586.76	617	28213.231	2560	2586	562	955.88	2523.56	2583.19	623	26030.145

Tableau 2 : Sol_opt inconnu

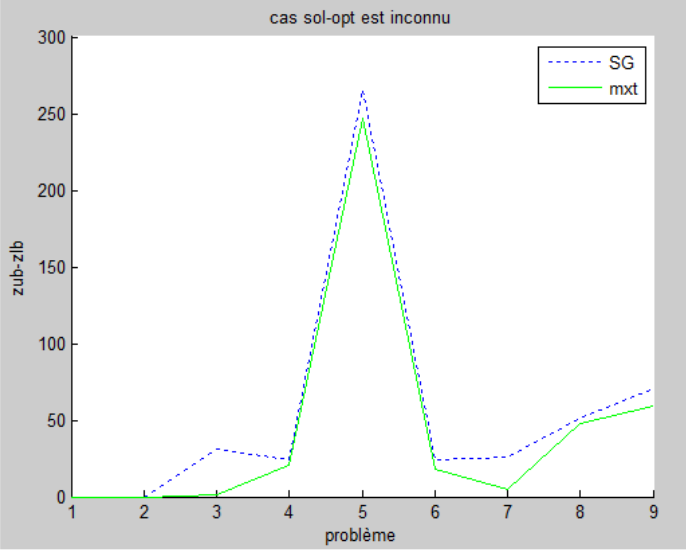

Figure 3

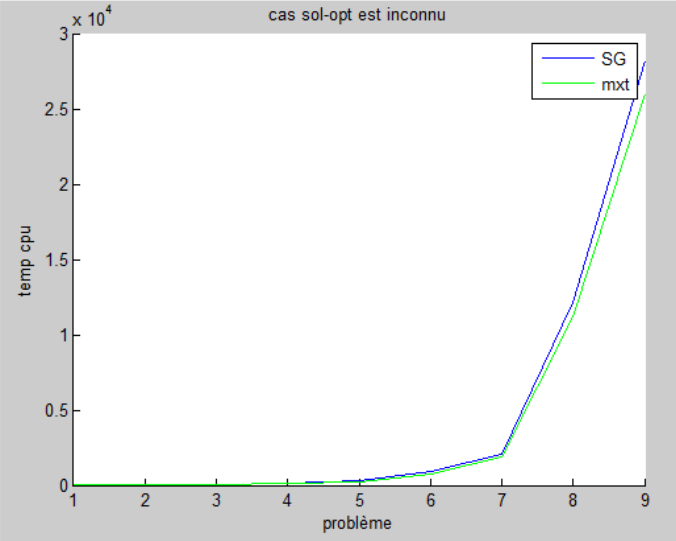

Figure 4

4.3 Comparaison entre l'algorithme (ACO) et la méthode (PPV)

Nous avons testé la méthode de colonie de fourmi avec la méthode du plus proche voisin sur les mêmes exemples qu'auparavant, Si la méthode (PPV) donne des temps CPU assez intéressant l'algorithme (ACO) nous fournit des solutions beaucoup plus précises (voir tableau3, figure 5 et figure 6)

		ACO			PPV		
problème	Sol_opt	iter	CPU(s)	Z_{ACO}	iter	CPU(s)	Z_{PPV}
gr17	2085	33	2.3754	2085	16	0.018458	2187
gr21	2707	65	7.2200	2707	3	0.022518	3333
dantzig42	699	405	288.1037	700.0135	15	0.025539	862.1250
berlin52	7542	961	1098	7544.4	2	0.032986	8980.9
brazil58	25395	323	408.63	25666	7	0.034229	30774
eil101	629	546	3172.4	642.3095	41	0.051425	825.2423
ch130	6110	840	10547	6110.9	82	0.061698	7575.3
ch150	6528	767	11314	6532.3	128	0.077961	8194.6
a280	2579	24	1875.5	2586.8	242	0.123894	3148.1

Tableau 3

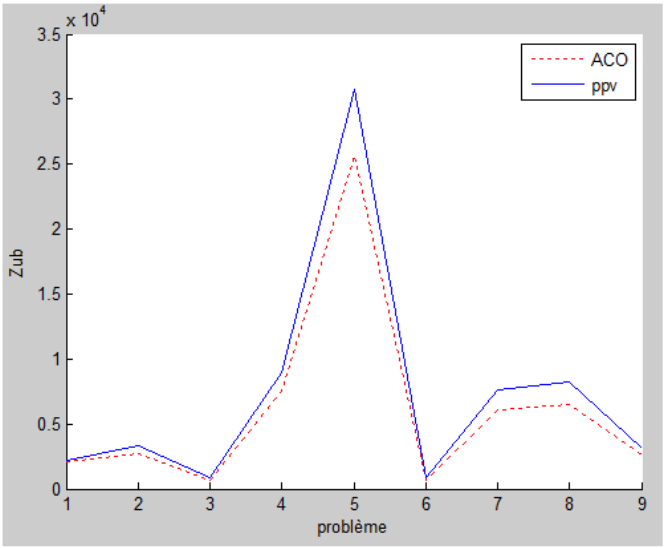

Figure 5

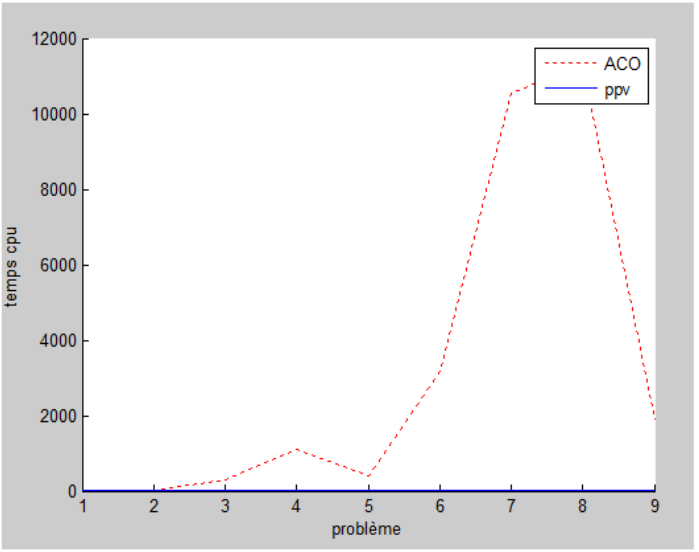

Figure 6

CONCLUSION

La méthode du sous-gradient de remplacement (SSG) peut donner de meilleures solutions que la méthode classique du sous-gradient (cela est du principalement à l'existence du phénomène de zigzag dans cette dérnière) pourvu que l'on sache choisir convenablement le coefficient de relaxation. Quoique la méthode (SSG) présente l'inconvénient du choix du coefficient de relaxation. Cependant, la combinaison des méthodes (SSG) et (SG) semble être intéressante et constitue une bonne alternative.

Nous avons utilisé la méthode du plus proche voisin pour déterminer une solution de départ des différentes méthodes (SG), (SSG) pour sa simplicité et surtout à cause du fait que les problèmes traités ne sont pas de très grandes tailles.

Nous aurions souhaité disposé d'une station de calcul pour pouvoir procéder à des tests numériques sur des problèmes de grandes tailles.

Bibliographie

[1] A. Colorni, M. Dorigo et V. Maniezzo, Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle, Paris, France, Elsevier Publishing, 134-142, (1991).

[2] M. Dorigo, M. Birattari, and T. Stiitzle, «Ant Colony Optimization», Tsinghua University Press, Beijing, (2007).

[3] M. Dorigo, Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italie, (1992)

[4] J. Dréo, al, «Méta heuristiques pour l'optimisation difficile» livre éditions Eyrolles ,(2003).

[5] J.L.Goffin, On Convergence Rates of Subgradient Optimization Methods, Mathematical Programming 13, 329-347,(1977).

[6] N.Z.Shor, Convergence Rtae of the Gradient Descent Method with Dilatation of the Space, Cybernatic 6, 102-108, (1970).

[7] X. Zhao, P. B. Luh, and J. Wang Communicated by W.B. Gong and D. D. Ya Surrogate subGradient Algorithm for Lagrangian Relaxation

[8] Modelisation et resolution de problemes d'optimisation combinatoire issus d'applications spatiales

[9] http ://fr.wikipedia.org/wiki/Algorithme_de_colonies_de_fourmis Lagrangian Relaxation

[10] http ://people.brunel.ac.uk/~mastjjb/jeb/info.html

[11] http ://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/

[12] http ://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/STSP.html

[13] https ://tel.archives-ouvertes.fr/tel-00009238

[14] http ://www.isima.fr/f4/projets2005/2005-2006/Boussa%EFdBouraoui.pdf