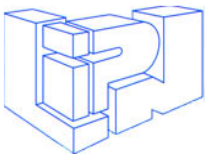


UNIVERSITÉ PARIS 13

INSTITUT GALILÉE

Rapport de Stage de Master de Recherche 2ème année
Mathématiques et Informatique

Option : Algorithmique, Modélisation et Image

de

Farah Ait Salaht

Branch and Cut pour le problème du voyageur de commerce asymétrique avec contraintes de saut

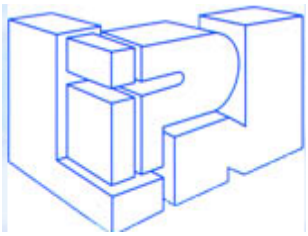
Encadrants : S. Borne, L. Létocart, L. Alfandari

Promotion 2009/2010

LIPN

Ce satge a été effectué au :

Laboratoire d'Informatique de Paris-Nord
UMR CNRS 7030
Institut Galilée - Université Paris-Nord
99, avenue Jean-Baptiste Clément
93430 Villetaneuse



Résumé

Le problème du voyageur de commerce asymétrique ($ATSP$) est un problème classique de l'optimisation combinatoire. Il consiste à trouver un circuit Hamiltonien de coût minimal contenu dans un graphe orienté sans boucle $G = (V,\ A)$ où $V = \{1, \ldots, n\}$ est l'ensemble des sommets et des coûts c_{ij} sont associés à chaque arc $(i,\ j) \in A$. Dans le cas asymétrique, les coûts c_{ij} et c_{ji} peuvent être différents pour tout couple $\{i \in V, j \in V\}$. Ce problème est connu pour être NP-difficile et a été très étudié dans la littérature [20, 48, 21, 35, 36, 16, 57, 56, 46].
Mais qu'en est-il si certaines villes doivent être éloignées d'au moins **s** liens dans le tour ?
Soit S l'ensemble des villes soumises à la condition de saut (i.e., dont la distance en nombre d'arêtes (resp. arcs) entre chacune de ces villes est d'au moins H). Le problème du voyageur de commerce asymétrique avec contraintes de saut ((HTSP), Hop Asymmetric Traveling Salesman Problem) consiste alors à trouver un circuit Hamiltonien de coût minimal qui respecte l'éloignement entre les villes de S.
On étudie dans ce travail deux formulations du problème sous la forme de programmes linéaires en nombres entiers. Nous utilisons notamment des variables de décision indiquant la position de chaque ville dans le tour ou des variables permettant de prendre en compte le nombre de sauts entre les villes dans le tour.
Nous adaptons certaines contraintes valides pour le problème ($ATSP$) présentes dans la littérature. Nous discutons de l'efficacité de ces inégalités, de la qualité des bornes inférieures obtenues par la résolution des relaxations linéaires. En utilisant ces résultats, nous présentons un algorithme de coupes et branchements pour le problème. Nous discutons de certains résultats expérimentaux.

Mots clés : Problème du voyageur de commerce asymétrique, contraintes valides, algorithme de coupes et branchements.

Abstract

Let $G = (V, A)$ be a directed graph and $S \subset V$ be a subset of nodes. Given a function c that associates a cost c_{ij} to each arc $(i, j) \in A$ and a bound $1 \leq H \leq |V|$, the Hop-constrained Traveling Salesman Problem (HTSP) consists in finding a minimum cost circuit visiting each node of V exactly once (an Hamiltonian circuit) such that two consecutive nodes of S in the circuit are separated by at least H arcs (or hops).

We study in this work two formulations of this problem and try to adapt some valid inequalities for the $ATSP$ problem mentioned in the literature. We discuss the effectiveness of these inequalities and the quality of lower bounds obtained by solving the linear relaxations.

Using these results, we present a branch and cut algorithm and some experimental results.

Key words : Asymmetric Traveling Salesman Problem, valid inequalities, branch and cut algorithm.

Table des matières

1

Introduction

La programmation linéaire en nombres entiers constitue un cadre général particulièrement utile pour modéliser, et bien souvent résoudre, des problèmes d'optimisation dans des domaines d'application extrêmement variés : logistique et transport, productique, réseaux de télécommunications ou de distribution d'énergie, etc ...

Grâce aux perfectionnements des techniques de programmation linéaire continue (algorithme du simplexe et méthodes de points intérieurs) ainsi qu'aux développements des techniques de la combinatoire polyédrique, des progrès très significatifs ont été accomplis depuis une dizaine d'années sur des problèmes combinatoires très fortement structurés tels que les problèmes de circuits Hamiltoniens dans les graphes (problème du voyageur de commerce). Des recherches très actives, menées depuis le début des années 80, sur le polyèdre correspondant à l'enveloppe convexe des cycles Hamiltoniens d'un graphe ont permis d'identifier les structures de nombreuses classes de facettes de ce polyèdre grâce auxquelles des relaxations très fortes peuvent être construites. De ce fait les évaluations par résolution continue des problèmes relaxés constituent des approximations très serrées des valeurs optimales entières, ce qui permet de diriger des recherches arborescentes (de type Branch and Bound) de façon très efficace. L'évolution, dans le temps, des progrès réalisés peut s'apprécier par la taille des records mondiaux des plus grands problèmes de voyageur de commerce résolus de façon exacte (c.à.d. en apportant la preuve de l'optimalité de la solution obtenue) : en 1980 par des techniques de combinatoire polyédrique

mettant en œuvre des inégalités d'élimination de sous-tours et des inégalités dites "de peigne" (comb inequalities) Crowder et Padberg [19] ont pu obtenir pour la première fois la solution exacte d'un problème à $n = 318$ villes. En 1987, Padberg et Rinaldi [52], perfectionnant les mêmes techniques atteignent la taille de $n = 532$. Avec des techniques plus avancées, les derniers records obtenus se situent au dessus de $n = 10000$ (cf par exemple Applegate, Bixby, Chvátal et Cook 2001).
Le principe commun à tous les travaux effectués consiste, par une analyse mathématique préalable approfondie exploitant au maximum toute la connaissance que l'on a de la structure des problèmes à résoudre, à mettre en évidence des classes d'inégalités valides (formant si possible des facettes) pour l'enveloppe convexe des solutions entières. Ces classes d'inégalités valides (de facettes) sont ensuite exploitées pour construire de bonnes relaxations selon un principe de génération d'inégalités non satisfaites par la solution courante ; ce procédé de génération est appelé séparation.

Le problème du voyageur de commerce asymétrique (ATSP) est un problème classique de l'optimisation combinatoire. Il consiste à trouver le circuit Hamiltonien de coût minimal contenu dans un graphe orienté sans boucle $G = (V, A)$ où $V = \{1, \ldots, n\}$ et les coûts c_{ij} sont associés à chaque arc $(i, j) \in A$. Dans le cas asymétrique, les coûts c_{ij} et c_{ji} peuvent être différents pour tout couple $\{i \in V, j \in V\}$. Ce problème est connu pour être *NP*-difficile et a été trés étudié dans la littérature.
Mais qu'en est-il si certaines villes doivent être éloignées d'au moins H liens dans le tour ?
Soit S l'ensemble des villes soumises à la condition de saut (i.e., dont la distance en nombre d'arcs entre chacune de ces villes est d'au moins H). Le problème du voyageur de commerce asymétrique avec contraintes de saut ((HTSP), Hop Asymmetric Traveling Salesman Problem) consiste alors à trouver un circuit Hamiltonien de coût minimal qui respecte l'éloignement entre les villes de S.

L'objectif du travail abordé ici est de présenter les différentes formulations du problème du voyageur de commerce asymétrique avec contraintes de saut, d'essayer d'adapter les inégalités valides connues pour le problème (ATSP) afin de déterminer leur efficacité pour le (HTSP).

Dans un premier temps, nous introduisons quelques notions de base et quelques notations utiles tout au long de ce document. Le chapitre 2 présente le problème du voyageur de commerce asymétrique, avec d'une part, un état de l'art sur les différentes modélisations et méthodes de résolution existantes, puis d'autre part, la présentation des différentes inégalités valides correspondant à ce problème. Le chapitre 3 se focalise sur le cas particulier étudié dans ce travail, à savoir le problème du voyageur de commerce asymétrique avec contraintes de saut, nous

présentons des formulations originales pour ce problème sous la forme de programmes linéaires en nombres entiers, et nous donnons certaines inégalités que nous jugeons valides et qui feront l'objet d'expérimentations. Dans le chapitre 4, nous présenterons une série de résultats d'expériences portant sur l'efficacité des inégalités valides associées au HTSP, nous ferons aussi une comparaison des divers résultats obtenus.

2

Notions Préliminaires

Pour être complet dans la description de notre problème, nous commençons par résumer quelques définitions ainsi que certains résultats de la théorie des graphes, de l'algèbre linéaire et de la théorie des polyèdres. Le but n'est pas d'être exhaustif, mais de fournir les concepts de base ainsi que les notations utilisées tout au long de ce document.

2.1 Théorie des graphes

Un **digraphe** (ou **graphe orienté**) $D = (V, A)$ se compose d'un ensemble fini non vide de nœuds V et d'arcs A. Un arc $a \in A$ est une paire ordonnée de nœuds $i, j \in V \times V$ noté $a = (i, j)$. On appelle i extrémité initiale (nœud origine) et j extrémité terminale (nœud destination) de l'arc $a = (i, j)$, i est considéré comme le prédécesseur de j, alors que j est le successeur de i. Si i est une extrémité de a, alors i (resp. a) est dit *incident* à a (resp. i). Dans tout ce qui suit, nous considérons des graphes sans boucle, à savoir, $i \neq j$ pour tout $a \in A$.
Un digraphe $D = (V, A)$ est dit **complet**, si tous les couples de nœuds $i, j \in V \times V$, $i \neq j$ sont reliés par les arcs $(i, j) \in A$ et $(j, i) \in A$. Un digraphe complet sur $n = |V|$ nœuds est noté $G_n = (V, A_n)$. Un digraphe $D = (V, A)$ est appelé transitivement fermé, si, pour tout $i, j, k \in V \times V \times V$, $i \neq j \neq k \neq i$, $(i, j) \in A$ et $(j, k) \in A$ implique $(i, k) \in A$.

Un **graphe non orienté** est noté $G = (V, E)$ où V est l'ensemble de nœuds et E l'ensemble des arêtes. Si e est une arête reliant deux nœuds u et v, alors u et v seront appelés les extrémités de e, et nous écrirons $e = uv$ ou $e = \{u, v\}$.
Un graphe $G = (V, E)$ est appelé **biparti**, si V peut être séparé en deux ensembles V_1 et V_2, de telle sorte que $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, et pour tout $e = uv$, $u \in V_i$, $v \in V_j$, $i \neq j$.

Dans ce travail, nous allons surtout travailler sur les graphes orientés, même si ce n'est pas dit explicitement.
Soient $D_1 = (V_1, A_1)$ et $D_2 = (V_2, A_2)$ deux digraphes, tels que $V_2 \subseteq V_1$ et $A_2 \subseteq A_1$, alors D_2 est appelé *sous graphe orienté* de D_1.
Étant donné un digraphe $D = (V, A)$. Si W est un sous-ensemble de V, alors

$$A(W) := \{(i, j) \in A | i, j \in W\}$$

représente l'ensemble de tous les arcs ayant leur origine et leur destination dans W.
Étant donnés les ensembles de nœuds U, $W \subset V$ avec $U \cap W = \emptyset$,

$$\{U : W\} := \{(i, j) \in A | i \in U, j \in W\}$$

désigne l'ensemble des arcs ayant leur origine dans U et leur destination dans W. Pour simplifier la notation, nous écrivons $(W : j)$ et $(j : W)$ au lieu de $(W : \{j\})$ et $(\{j\} : W)$. Si $U = \emptyset$ ou $W = \emptyset$, alors $(U : W) = \emptyset$.
Soit l'ensemble de nœuds $W \subset V$, $W \neq \emptyset$, nous avons

$$\begin{aligned} \delta^-(W) &:= \{(i, j) \in A | i \in V \setminus W, j \in W\}, \\ \delta^+(W) &:= \{(i, j) \in A | i \in W, j \in V \setminus W\}, \\ \delta(W) &:= \delta^-(W) \cup \delta^+(W). \end{aligned}$$

L'ensemble d'arcs $\delta(W)$ est appelé **coupe**. On sait que $\delta^-(W) = \delta^+(V \setminus W)$. Pour simplifier la notation, on écrit $\delta^-(v)$, $\delta^+(v)$, $\delta(v)$, au lieu de $\delta^-(\{v\})$, $\delta^+(\{v\})$, $\delta(\{v\})$. Les quantités $|\delta^-(v)|$, $|\delta^+(v)| et |\delta(v)|$ sont appelées degré entrant, degré sortant, et degré du nœud $v \in V$. Un nœud $v \in V$ est appelé **isolé**, s'il a un degré zéro, c'est-à-dire, $|\delta(v)| = 0$.
Si un digraphe $D = (V, A)$ est muni du poids c_a associé à chaque arc $a \in A$, on dit alors que D est un digraphe pondéré.
Étant donné un digraphe $D = (V, A)$, alors l'ensemble d'arcs

$$P = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\},\ k \geq 2,\ v_i \in V\ \forall i = 1, \ldots, k$$

est appelé chemin, ou plus exactement un $[v_1, v_k]$-chemin. La longueur du chemin est égale au nombre d'arcs qui le composent et est notée $|P|$ égale à $k - 1$. Chaque arc qui n'appartient pas

au chemin, mais qui est relié à deux autres nœuds qui le sont est appelé une **corde**.
Un chemin orienté de longueur $|V| - 1$ est appelé *chemin hamiltonien*, si le nœud de départ et le nœud d'arrivée coïncident alors il est appelé **circuit**. Pour ne pas risquer la confusion, nous utiliserons l'expression cycle au lieu de circuit. Un cycle est appelé **élémentaire**, si tous les nœuds sont deux à deux différents, à savoir., $v_i \neq v_j$ pour tout $i, j = 1, \ldots, k$, $i \neq j$. Un cycle simple de longueur $|V|$ est appelé un **cycle hamiltonien** ou un **tour**. Un cycle simple de longueur inférieure à $|V|$ est appelé *sous-tour*. On dit qu'un digraphe $D = (V, A)$ est **acyclique**, si l'ensemble des arcs de A ne contient pas de cycle.
Un graphe G est **connexe** si, pour toute paire de nœuds u, v de V, il existe au moins une chaîne entre u et v.
Soit un graphe $G = (V, E)$, un **arbre** est un ensemble d'arcs connexe de G qui ne contient aucun cycle. L'arbre est dit **couvrant** s'il contient tous les nœuds du graphe.
Une clique dans un graphe $D = (V, A)$ est un ensemble de nœuds $V_C \subseteq V$ tel que $D_C = (V_C, A(V_C))$ est un *sous graphe orienté* complet de D.

2.2 Approche polyédrale et méthode de coupes et branchements

Un grand nombre de problèmes issus de cas réels peuvent être formulés sous la forme de problèmes d'optimisation combinatoire. À première vue, leur résolution semble facile. Nous pouvons par exemple faire une énumération de toutes les solutions possibles et choisir la meilleure. Cependant, le nombre de ces solutions devient vite exponentiel et la méthode énumérative n'est plus possible. D'où la nécessité de développer des techniques permettant de résoudre ces problèmes plus efficacement. Une des méthodes les plus puissantes est la méthode polyédrale. L'approche polyédrale des problèmes d'optimisation doit son succès grandissant à ses bons résultats quand elle est appliquée au problème du voyageur de commerce (traveling salesman problem TSP). Padberg et Rinaldi [51], Grötschel et Holland [37] et récemment Applegate et al. [3] ont résolu des instances de TSP de trés grande taille (plusieurs milliers de sommets). Ces résultats sont dûs à une bonne connaissance du polyèdre enveloppe convexe des solutions du TSP ainsi qu'au développement d'algorithmes performants autant pour l'analyse des solutions des relaxations linéaires du TSP que pour l'utilisation des bornes obtenues dans des procédures d'énumérations.

Dans cette section, nous allons présenter brièvement cette approche. Pour cela, nous donnons tout d'abord quelques définitions.

2.2.1 Définitions

Dans cette section, nous allons rappeler quelques définitions et propriétés liées à la théorie des polyèdres.

Soit $n \in I\!N$. Le symbole $I\!R^n$ représente l'ensemble des vecteurs ayant n composantes réelles. L'ensemble des nombres réels positifs sera noté $I\!R_+$.

Etant donné un ensemble de points $x^1, \ldots, x^m \in I\!R^n$, un point $x \in I\!R^n$ est dit *combinaison linéaire* de $x^1, \ldots, x^m$ s'il existe $\lambda_1, \ldots, \lambda_m \in I\!R$ tels que

$$x = \sum_{i=1}^{m} \lambda_i x^i.$$

Si de plus

$$\sum_{i=1}^{m} \lambda_i = 1$$

$$\text{(resp. } \lambda_i \in I\!R_+ \text{ pour i=1,...,m et } \sum_{i=1}^{m} \lambda_i = 1),$$

x est dit *combinaison affine* (resp. *combinaison convexe*) de ces points.

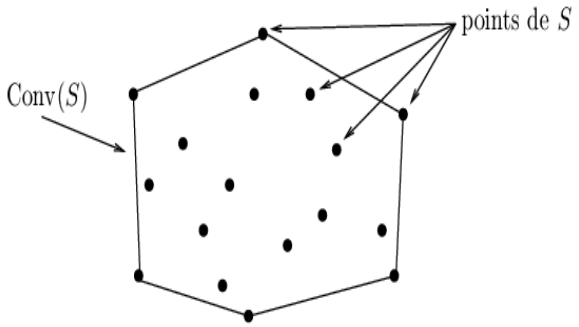


FIG. 2.1 – Enveloppe convexe

Etant donné un ensemble S de points $x^1, \ldots, x^m \in I\!R^n$, l'*enveloppe convexe* de $x^1, \ldots, x^m$ est

$$conv(S) = \{x \in \mathcal{R}^n | x \text{ combinaison convexe de } x^1, \ldots, x^m\}.$$

La figure 2.1 illustre cette notion.

Des points $x^1, \ldots, x^m \in I\!R^n$ sont *linéairement indépendants* (resp. *affinement indépendants*) si le système

$$\sum_{i=1}^{m} \lambda_i x^i = 0$$

$$(\text{resp. } \sum_{i=1}^{m} \lambda_i x^i = 0 \text{ et } \sum_{i=1}^{m} \lambda_i = 0)$$

admet une solution unique $\lambda_i = 0$ pour $i = 1, \ldots, m$.

Un *polyèdre* P est l'ensemble des solutions d'un système linéaire $Ax \leq b$ c'est-à-dire $P = \{x \in \mathcal{R} | Ax \leq b\}$, où A est une matrice à m lignes et n colonnes, et b un vecteur à m composantes. Un *polytope* est un polyèdre borné.

Un polyèdre $P \subseteq I\!R^n$ est dit de *dimension* p si le nombre maximum de points de P affinement indépendants est $p + 1$. Nous noterons alors $dim(P) = p$. Un polyèdre P de $I\!R^n$ est *de pleine dimension* si $dim(P) = n$.

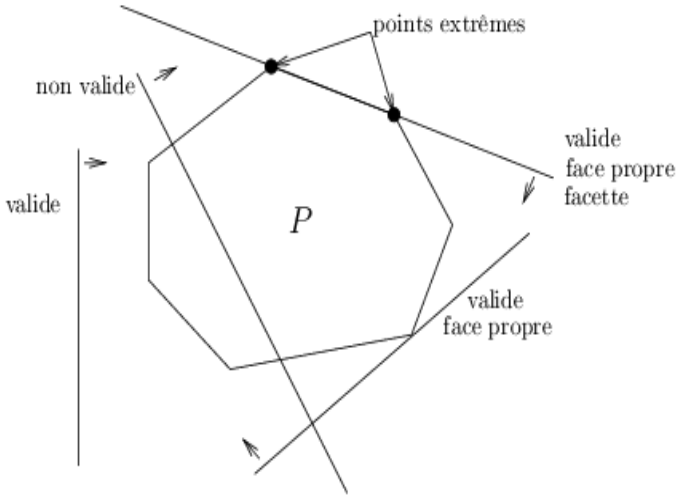


FIG. 2.2 – Contraintes valides, faces et facettes

Si a et x sont deux vecteurs colonne à n composantes, on note par $a^T x$, le produit scalaire de la transposée de a par x.

Une contrainte $a^T x \leqslant \alpha$ est dite *valide* pour un polyèdre P de $I\!R^n$ si elle est vérifiée pour toute solution de P. Soit $x^* \in I\!R^n$. L'inégalité $a^T x \leqslant \alpha$ est dite *serrée* pour x^* si $a^T x^* = \alpha$. Une contrainte est dite *violée* par x^* si x^* ne satisfait pas la contrainte (voir FIG. 2.2).

Etant donnés un polyèdre P et une contrainte $a^T x \leqslant \alpha$ valide pour P, le sous-ensemble $F = \{x \in P | a^T x = \alpha\}$ est appelé *face* de P *définie* par $a^T x \leqslant \alpha$. De plus, nous avons $dim(F) \leqslant dim(P)$. Une face F est dite *propre* si $F \neq P$ et $F \neq \emptyset$. Une face propre F est une *facette* de P si $dim(F) = dim(P) - 1$.

Un *point extrême* d'un polyèdre P est une face de P de dimension 0. Il est facile de voir qu'un point $x \in I\!R^n$ est un point extrême d'un polyèdre P, s'il ne peut pas être écrit comme combinaison convexe d'autres points de P.

2.2.2 Approche polyédrale

Soient $\mathcal{P}$ un problème d'optimisation combinatoire, $\mathcal{S}$ l'ensemble des solutions de $\mathcal{P}$, E l'ensemble de base de $\mathcal{P}$ et c la fonction poids associée aux variables du problème. Le problème $\mathcal{P}$ s'écrit donc $min\{cx \mid x \in \mathcal{S}\}$.

Si F est un sous-ensemble de E, le vecteur $x^F \in I\!R^E$ (ayant $|E|$ composantes associées aux éléments de E) tel que

$$x^F(e) = \begin{cases} 1 & \text{si } e \in F, \\ 0 & \text{sinon} \end{cases}$$

est appelé *vecteur d'incidence* de F. Le polyèdre

$$P(\mathcal{S}) = conv\{x^S \mid S \in \mathcal{S}\}$$

est appelé *polyèdre des solutions* de $\mathcal{P}$ (ou *polyèdre associé* à $\mathcal{P}$).

Le problème $\mathcal{P}$ est donc équivalent au programme linéaire $min\{cx \mid P(\mathcal{S})\}$. $P(\mathcal{S})$ peut être caractérisé par un ensemble de contraintes linéaires où chaque contrainte définit une facette. Si on peut décrire entièrement le polyèdre $P(\mathcal{S})$ par un système d'inégalités linéaires, le problème $\mathcal{P}$ se ramène donc à la résolution d'un programme linéaire.

L'*approche polyédrale* consiste à ramener la résolution du problème à la résolution d'un (ou d'une séquence) de programmes linéaires. Cette transformation nécessite, par conséquent, une étude approfondie du polyèdre associé au problème. Néanmoins, la caractérisation complète de ce polyèdre est généralement difficile à établir (voire impossible si le problème est NP-difficile). De plus, le nombre de contraintes nécessaires pour décrire le polyèdre, est souvent exponentiel. Cependant, en utilisant une méthode de coupes et branchements (Branch-and-Cut method), une description partielle peut être suffisante pour résoudre le problème à l'optimum. Cette méthode combine la méthode de coupe (Cutting planes method) et la méthode de *séparation et évaluation* (Branch & Bound method).

2.2.3 Méthode de coupes et branchements

La méthode de coupes permet de résoudre un problème $\mathcal{P}$ comme une séquence de programmes linéaires, chacun contenant un nombre raisonnable de contraintes.

Soit P un polyèdre dans $I\!R^n$. Le *problème de séparation* associé à P consiste à vérifier pour un point $x^* \in I\!R^n$ s'il appartient à P, et dans le cas contraire, à trouver une contrainte $a^T x \leqslant b$ valide pour P et violée par x^* (voir FIG.2.3).

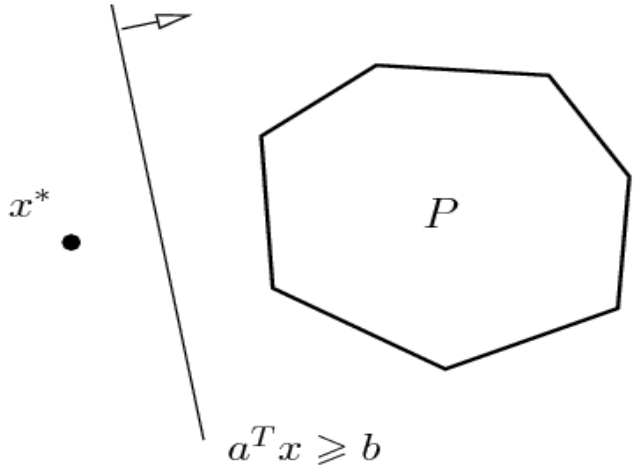


FIG. 2.3 – Hyperplan séparant x^* et P

Grötschel, Lovász et Schrijver [38] ont montré qu'un problème d'optimisation, sur un polyèdre, est polynomial si et seulement si le problème de séparation associé à ce polyèdre peut être résolu en temps polynomial.

La méthode de coupe permet donc de résoudre un problème d'optimisation combinatoire en temps polynomial, si le problème de séparation associé aux contraintes du polyèdre de ses solutions est polynomial et ce, même si le nombre de ces contraintes est exponentiel.

Le nombre de facettes du polyèdre des solutions d'un problème $\mathcal{P}$ peut être exponentiel. Si le problème est NP-complet, il y a très peu d'espoir de pouvoir déterminer une description complète du polytope de ses solutions. Néanmoins, une description partielle peut parfois suffire pour résoudre le problème. La méthode dite de *"coupes et branchement"* (Branch & Cut method) peut permettre de résoudre le problème en utilisant un nombre réduit de contraintes. Cette méthode combine la méthode de génération de nouvelles contraintes (coupes) et la méthode dite de *"séparations et évaluations"* (Branch & Bound method).

Soit $Ax \leqslant b$ un système de contraintes valides pour le problème et pour lesquelles il existe des méthodes de séparation. Nous considérons un sous-système $A_1x \leqslant b_1$ avec un nombre raisonnable de contraintes. Nous résolvons le programme linéaire $P_1 = max\{cx \text{ — } A_1x \leqslant b_1\}$. Si la solution optimale x_1 de P_1 est réalisable pour le problème d'optimisation combinatoire $\mathcal{P}$, alors elle est optimale pour $\mathcal{P}$. Sinon, il doit exister une contrainte valide pour $\mathcal{P}$ et violée par x_1. On résoud le problème de séparation associé à $Ax \leqslant b$ et x_1 et on essaye de trouver une contrainte $a'x \leqslant \alpha'$ qui soit violée par x_1. Cette nouvelle contrainte est ajoutée au système $A_1x \leqslant b_1$. Nous obtenons un nouveau programme linéaire $max\{cx \text{ — } A_1x \leqslant b_1, a'x \leqslant \alpha'\}$ et ainsi de suite. En continuant ce processus appelé *phase de coupe*, nous pouvons trouver, soit une solution optimale pour $\mathcal{P}$, soit une solution x^*qui ne soit pas réalisable pour $\mathcal{P}$ et pour laquelle nous ne

pouvons plus générer de contrainte violée. Dans le deuxième cas, nous commençons une phase dite de *branchement* qui consiste à construire un arbre de Branch & Bound. Nous choisissons une variable fractionnaire x_i^*. Nous résolvons deux nouveaux programmes linéaires (nouveaux sommets de l'arbre) en ajoutant soit la contrainte $x_i = 0$, soit $x_i = 1$ au programme linéaire obtenu à la fin de la phase de coupe. Pour chacun de ces nouveaux sommets, nous appliquons une procédure de coupes. Si une solution optimale n'a pas été trouvée, nous sélectionnons une feuille de l'arbre et nous recommençons une phase de branchement.

Cette approche est maintenant largement utilisée pour les problèmes d'optimisation combinatoire.

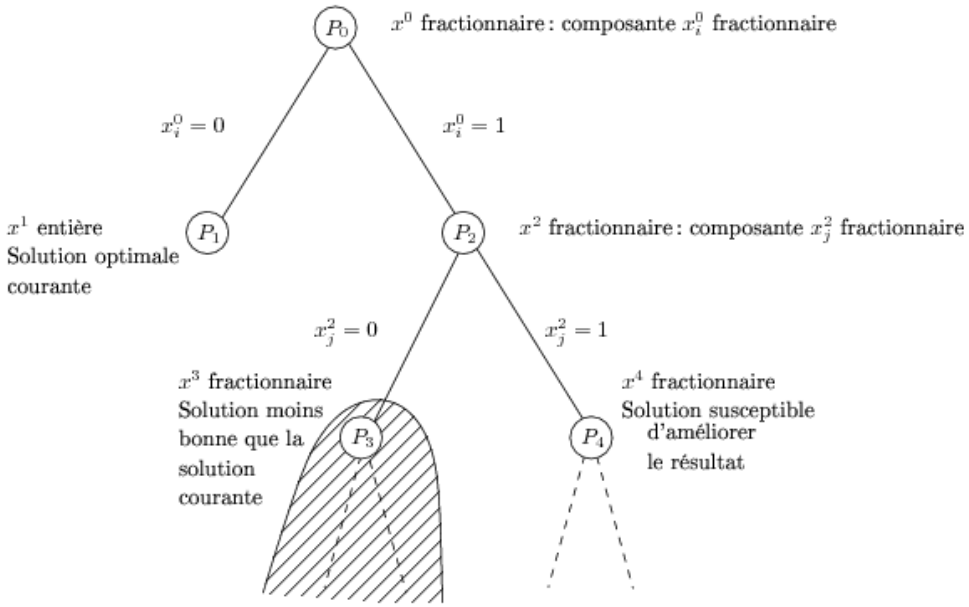


FIG. 2.4 – Phase de branchement

2.3 Théorie de la complexité

Nous examinons brièvement de manière informelle les concepts de base de la théorie de la complexité, dans la mesure où ils sont utilisés dans ce travail. Pour une description approfondie des concepts et une étude approfondie nous nous référons à Garey et Johnson (1979) [30].
Nous distinguons deux types de problèmes :

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

Un problème de décision est dit appartenir à la classe $\mathcal{P}$, s'il existe un algorithme pouvant le résoudre en temps polynomial. La classe $\mathcal{NP}$ 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, $\mathcal{P} \subset \mathcal{NP}$ qu'on suppose communément acceptée de plus cette inclusion est stricte, c'est-à-dire, $\mathcal{P} \neq \mathcal{NP}$. On distingue également dans la classe $\mathcal{NP}$, la classe des problèmes $\mathcal{NP}$-complets.

La NP-complétude s'appuie sur la notion de réduction polynomiale. Un problème de décision P_1 se réduit polynomialement en un problème de décision P_2 s'il existe une fonction polynomiale $\boldsymbol{f}$ telle que, pour toute instance $\boldsymbol{I}$ de P_1, la réponse est oui si et seulement si la réponse de $f(I)$ pour P_2 est oui.

3

Problème du voyageur de commerce asymétrique

3.1 État de l'art

3.1.1 Problème du voyageur de commerce

Historique et définition

On connaît mal l'origine exacte du nom de *"Problème du voyageur de commerce"* noté TSP (*Traveling Salesman Problem*). Cependant, il s'agit d'un des plus vieux problèmes combinatoires. Les premières approches mathématiques exposées pour le TSP ont été traitées au $19^{ème}$ 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, le TSP est traité plus en profondeur par Karl Menger à Harvard, il a ensuite été développé à Princeton par les mathématiciens Hassler Whitney et Merril Flood. On constate ainsi, que des mathématiciens s'y sont intéressés depuis le début du vingtième siècle cherchant à apporter une réponse scientifique à ce problème réel parmi eux : Dantzig et *al.* (1954) a résolu un problème du TSP de 49 villes en utilisant la méthode de génération de coupe [20] , Camerini, Fratta et Maffioli ont trouvé la solution pour un problème de 100 villes en 1974 [12], Padberg et Rinaldi (1987) ont résolu un problème de 532 villes [52] puis un autre de 2392 villes [51], Applegate, Bixby, Chvàtal et Cook (2001) des universités de Rice et Princeton ont obtenu la solution

optimale pour un problème de 15112 villes en Allemagne [4], en 2006 Cook et *al.* ont résolu un problème de 85900 villes,. . ..

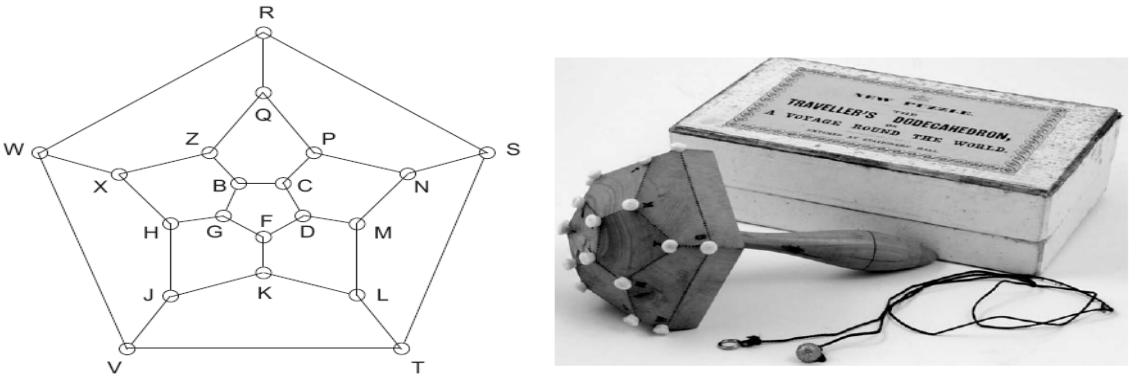


FIG. 3.1 – Hamilton's Icosian game

La définition du problème du voyageur de commerce est simple : un voyageur de commerce doit visiter n villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Il faut trouver le chemin qui minimise la distance parcourue (voir [45]). La notion de distance peut-être remplacée par d'autres notions comme le temps qu'il met ou l'argent qu'il dépense.

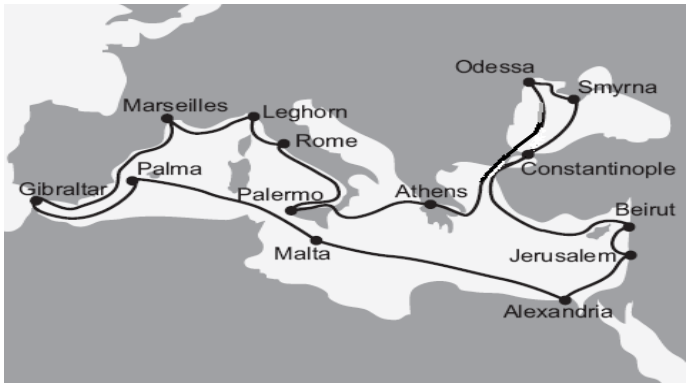


FIG. 3.2 – Problème du voyageur de commerce

Formellement, à partir d'une matrice $C = (c_{ij})$ où c_{ij} représente le coût du déplacement de la ville i à la ville j $(1 \leq i, j \leq n)$, il faut trouver une permutation qui minimise le coût total de la tournée. Autrement dit, il faut trouver un circuit (respectivement cycle) Hamiltonien de longueur minimale dans un graphe orienté (respectivement non orienté) valué complet. Si $c_{ij} = c_{ji}$ pour tout $i, j \in \{1, \ldots, n\}, i \neq j$ le problème est dit symétrique sinon il est asymétrique. De plus, C satisfait les inégalités triangulaires si et seulement si $(c_{ij} + c_{jk} \geq c_{ik}, \quad \forall i, j, k \in V)$, dans ce cas,

le problème est appelé TSP métrique.
Le TSP peut être formulé comme un problème de programmation linéaire en nombres entiers. De plus, il présente plusieurs caractéristiques communes aux problèmes d'optimisation combinatoire et fournit ainsi une plate-forme idéale pour étudier les méthodes générales applicables à un grand nombre de ces problèmes.
L'existance d'un algorithme déterministe exact avec une complexité polynomiale reste inconnue. Un calcul rapide de la complexité montre qu'elle est en $O(n!)$ où n est le nombre de villes.
Le TSP appartient à la famille des problèmes NP-*difficiles* 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).

Applications et typologie

Ce problème a de nombreuses applications pratiques et sert de référence pour d'autres problèmes, notamment dans le transport et la logistique. Par exemple, trouver le chemin le plus court pour les bus de ramassage scolaire ou, dans l'industrie, pour trouver la plus courte distance que devra parcourir le bras mécanique d'une machine pour percer les trous d'un circuit imprimé.
Il existe de nombreuses variantes du TSP, obtenues soit par adjonctions de contraintes – comme le TSP 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, –soit par modification, comme par exemple les problèmes de tournée de véhicules (VRP pour *Vehicule Routing Problem*) dans lesquels on ne considère plus un unique représentant de commerce pour visiter les villes mais une équipe (une flotte de véhicules) et qui peuvent être vu comme des problèmes de flot.

3.1.2 Problème du voyageur de commerce asymétrique

Historique et variantes

Le problème du voyageur de commerce asymétrique $ATSP$ est défini sur un graphe orienté $G = (V, E)$, où $V = \{1, \ldots, n\}$ représente l'ensemble des nœuds (sommets), $E = \{(i, j) : i, j \in V\}$ est l'ensemble des arcs, et la matrice non symétrique des coûts c_{ij} est définie sur E. $ATSP$ consiste à déterminer un circuit (tour) Hamiltonien sur G ayant un moindre coût. Le problème est souvent interprété comme étant la détermination d'une tournée optimale du voyageur à travers les n villes. $ATSP$ est NP-*difficile* même si les coûts sont euclidiens.
Différentes variantes peuvent intervenir lors de la description du problème $ATSP$, en voici quelques-unes :

- ***ATSP périodique*** : Cette généralisation du problème ATSP consiste à couvrir le planning d'une période de k-jours, pour k donné. On veut trouver ainsi k cycles dans G tel que chaque nœud de G soit visité un nombre déterminé de fois au cours de la période de k-jours et le coût total de la tournée pour toute la période de k-jours doit être minimale (voir [55])
- ***ATSP "with replenishment arcs"*** : Le problème du voyageur de commerce asymétrique avec "replenishment arcs" noté (RATSP), consiste à trouver une tournée de coût minimum. Dans ce contexte, on envisage la possibilité d'attribuer deux arcs parallèles entre deux nœuds, à savoir un arc de reconstitution et un arc ordinaire, aussi longtemps que l'arc ordinaire sera de moindre coût.
  Le RATSP est une généralisation de ATSP qui a diverses applications dans les problèmes de routage. On peut citer le problème de routage des avions où l'on considère les possibilités d'entretien comme arcs de reconstitution, mais aussi le problème de tournées de véhicules où la demande des clients ainsi que la capacité des véhicules peuvent être modélisées par des arcs de reconstitution, (voir [10]).
- ***ATSP avec contraintes de précédence*** : Le problème ATSP avec contraintes de précédence, aussi appelé *Sequential ordering problem* (SOP), exige que certains nœuds doivent être précédés par d'autres et cela dans n'importe quelle tournée réalisable. Ce problème se présente comme étant un modèle essentiel dans la planification et le routage et dispose d'une large gamme d'applications, allant du routage de véhicules, le séquençage dans l'industrie manufacturière,....[56, 34].
- ***ATSP avec fenêtres de temps ATSP-TW*** : Le problème du voyageur de commerce asymétrique avec fenêtres de temps (*ATSP with time windows*) est un problème classique de planification et d'acheminement des demandes. Le concept consiste à imposer des contraintes de temps pour la traversée de chaque sommet [2, 5].
- ***ATSP avec contraintes de saut*** : Soit S l'ensemble des villes soumises à la condition de saut (c'est-à-dire, dont la distance en nombre d'arcs entre chacune de ces villes est d'au moins s). Le problème du voyageur de commerce asymétrique avec contraintes de saut (HTSP) consiste alors à trouver un circuit hamiltonien de coût minimal qui respecte l'éloignement entre les villes de S. C'est le problème traité dans ce document.

Les différentes formulations du problème ATSP

$ATSP$ peut être modélisé comme un problème de programmation linéaire en nombres entiers sous contraintes.
À chaque ville correspond un entier i, compris entre 1 et n. Pour chaque couple de villes (i, j),

on définit par c_{ij} le coût de passage de la ville i à la ville j et la variable binaire :

$$x_{ij} = \begin{cases} 1 \text{ si le sommet } j \text{ est le successeur immédiat de } i \text{ dans la tournée}; \\ 0, \text{ sinon.} \end{cases}$$

La formulation du programme mathématique associé au problème du voyageur de commerce asymétrique implique des contraintes d'affectation (assignment constraints) ainsi que les contraintes d'élimination des sous-tours (subtour elimination constraints : SECs), sans oublier les restrictions sur les variables de décision binaires (voir [20, 45, 50]).
Une formulation simple du problème est présentée ci-dessous : [49]

$$Minimiser \quad z = \sum_i \sum_j c_{ij} x_{ij} \tag{3.1}$$

sous les contraintes :

$$\sum_{i \in V} x_{ij} = 1 \quad , \qquad \forall j \in V, \tag{3.2}$$

$$\sum_{j \in V} x_{ij} = 1 \quad , \qquad \forall i \in V, \tag{3.3}$$

$$0 \leq x_{ij} \leq 1 \quad , \qquad \forall i, j \in V, \tag{3.4}$$

$$x_{ij} \in \{0, 1\} \quad , \qquad \forall i, j \in V, \tag{3.5}$$

$$\{(i,j) : x_{ij} \quad = \quad 1, \quad i, j \in V \backslash \{1\}\} \; ne\ contient\ pas\ de\ sous - tours. \tag{3.6}$$

Dans cette formulation, les contraintes d'affectation (3.2) et (3.3) assurent que chaque sommet compte exactement un arc entrant et un arc sortant, quand à la contrainte (3.6), elle a pour fonction de briser les sous-tours existant dans l'ensemble $\{2, \ldots, n\}$, prise avec les contraintes (3.2) et (3.3), elle permet également d'éliminer les sous-tours contenant le sommet 1, ainsi la contrainte d'élimination de sous-tours pour $i = 1$ devient redondante.
On remarque donc, que la formulation concernant les contraintes d'élimination des sous-tours est très faible. Dantzig, Fulkerson & Johnson (1954) [20] ont proposé une autre formulation plus forte des contraintes de sous-tours, le modèle $ATSP$ proposé est le suivant :

$$Minimiser \quad z = \sum_i \sum_j c_{ij} x_{ij} \tag{3.7}$$

sous les contraintes :

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V, \tag{3.8}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V, \tag{3.9}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset V \ t.q. \ 2 \leq |S| \leq n-2, \tag{3.10}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in V, \ \ \forall j \in V, \ \ i \neq j \tag{3.11}$$

L'objectif est de minimiser le coût total de transport lié aux arcs empruntés par le voyageur. Les contraintes (3.8) et (3.9) consistent à assurer que le voyageur arrive et reparte une seule fois de chaque sommet (ville), puis les contraintes (3.10) dites contraintes d'élimination de sous-tours (SEC), elles consistent à éviter de revenir au point de départ avant la réalisation de la tournée Hamiltonienne. Enfin, le respect des variables binaires est assuré par la contrainte (3.11). 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 voir impossible.
On définit la notation introduite par Grötschel et Padberg 1985 [39] et désignons par

$$P_T^n = conv\{x \in \{0,1\}^{A_n} | satisfaisant (3.8) - (3.11)\}$$

le polytope associé au voyageur de commerce asymétrique, à savoir, l'envelope convexe associé aux vecteurs d'incidence pour tous les $(n-1)!$ tours dans le digraphe complet $D_n = (V, A_n)$.

D'autres formulations pour les contraintes de sous-tours ont été proposées, afin de réduire cette combinatoire. Notamment, Miller, Tucker & Zemlin (1960) [48] ont proposé une formulation en n^2 contraintes, et ont ainsi réécrit la contrainte d'élimination des sous-tours (3.10).
Cette formulation, présentée ci-dessous, permet de grandement réduire le nombre de contraintes de sous-tours nécéssaires mais demande l'ajout de variables continues. La formulation la plus ancienne pour $ATSP$ est dûe à $Miller$ et al. [6]. Elle a été proposée à l'origine pour le problème de tournées de véhicules (VRP), où le nombre de sommets associés à chaque route est limité. Ils utilisent des variables u_i pour définir l'ordre dans lequel chaque sommet i est visité dans la tournée. Les contraintes d'élimination des sous-tours définies par la formulation de MTZ sont énoncées comme suit

$$\begin{cases} u_i - u_j + (n+1)x_{ij} \leq n-2 & (i,j \in \{2,\ldots,n\}, i \neq j) \\ 1 \leq u_i \leq n-1 & (i \in \{2,\ldots,n\}) \end{cases}$$

Cette modélisation, bien que plus compacte, n'est pas tellement plus efficace. Il est possible d'utiliser ces formulations seulement pour des petits problèmes. La multitude de tours possibles rend ce problème complexe et tous les évaluer demanderait un temps considérable. Une étude sur la complexité est présentée par Laporte (1992) [44].
On peut trouver dans la littérature d'autre formulations du problème du voyageur de commerce asymétrique, pour plus de détails voir [49].

3.1.3 Méthodes de résolution

Les algorithmes de résolutions du TSP (resp. $ATSP$) peuvent être répartis en deux classes :

- Les algorithmes exacts (déterministes) permettent de trouver la solution optimale, mais leur complexité est d'ordre exponentiel. Ces algorithmes demandent beaucoup de place mémoire et ils sont très gourmands en temps de calcul.
- Les algorithmes approchés (heuristiques) permettent de trouver en un temps raisonnable une solution approchée de la solution optimale. Ils sont utilisés généralement pour résoudre des problèmes concrets de grandes tailles. Nous présenterons quelques unes de ces heuristiques.

1.2.3.1 Les méthodes exactes

L'intérêt des méthodes exactes est d'apporter l'assurance d'obtenir une solution optimale. Pour cela, elles doivent parcourir l'ensemble de l'espace de recherche, ou au moins avoir l'assurance de n'écarter aucune solution ayant le potentiel d'être meilleure que la solution optimale trouvée par l'algorithme.

Les méthodes de résolution les plus répandues sont certainement celles construisant un arbre de recherche afin d'explorer l'ensemble de l'espace de recherche. Ces méthodes sont appelées $Branch\ \&\ *, '*'$ pouvant être remplacé par $Bound$, Cut, $Price$ ou $Cut\ \&\ Price$ (ces méthodes sont également appelées méthodes de séparation). Elles utilisent des bornes pour éliminer de la recherche des ensembles de solutions ayant une certaine structure. D'autre méthodes sont spécifiques à un problème donné, ou moins générales, comme la programmation dynamique et la méthode des plans sécants.

Méthode de Branch & Bound

Les problèmes de voyageur de commerce asymétrique sont souvent résolus à l'optimalité avec l'algorithme de *Branch & Bound* qui considère la relaxation du problème d'affectation (*Assignment problem* AP) (Fischetti et al., 2002)[28].
Au lieu d'avoir une tournée unique à travers toutes les villes (nœuds), une solution optimale de AP se compose généralement de plus d'un tour, ce qu'on appelle les sous-cycles, l'algorithme de $B\&B$ est construit à partir des quatres étapes suivantes :

La règle de branchement elle permet de donner la façon dont le problème actuel devrait être subdivisé en sous-problèmes. L'idée est donc d'opérer une séparation. Cette séparation nécessite le choix d'une variable de séparation. Le choix de cette variable est heuristique, différents choix sont possibles et de ce dernier peut dépendre l'efficacité de la méthode de résolution. Une règle de branchement efficace pour le ATSP avec AP relaxation a été introduite dans Carpaneto et Toth (1980)[15].

La stratégie de parcours dans l'arbre permet de déterminer le prochain sous-problème à traiter. Deux stratégies sont largement utilisés, la stratégie de parcours en profondeur d'abord (DFS) dont le principe est de commencer par résoudre le sous-problème généré en dernier, la stratégie de parcours meilleur d'abord (BFS) qui consiste à résoudre en premier lieu le sous-problème jugé meilleur, généralement, le sous-problème ayant la plus faible borne AP.

La stratégie de la borne supérieure détermine comment les tournées devraient être construites dans le processus de $B\&B$. Une méthode de construction d'une tournée réalisable pour ATSP à partir des solutions AP est la procédure de réajustement donnée par Karp et Steele (1990) [42].

La stratégie de la borne inférieure détermine comment construire la borne inférieure associée à la valeur d'une solution pour un sous-problème. La valeur de la solution AP du sous-problème est généralement considérée comme une borne inférieure.

Un état de l'art concernant l'algorithme de $B\&B$ pour le problème ATSP peut être trouvé dans Miller et Pekny (1991) [47], Carpaneto et al. (1995) [14] et plus tard Turkensteen(2007) [43].

Les algorithmes donnés par Miller et Carpaneto appliquent un réajustement pour obtenir des bornes supérieures, et utilisent les bornes inférieures AP, et branchent sur le plus petit cycle dans la solution AP en cours. La stratégie de parcours des deux algorithmes est BFS (meilleur d'abord), cela signifie que pour de nombreuses instances du ATSP, des solutions sont obtenues dans des délais très courts. D'autre part, une liste des sous-problèmes doit être maintenue afin de déterminer la plus prometteuse. En conséquence, les algorithmes BFS ont tendance à manquer de mémoire lorsque l'arbre de parcours s'agrandit.
Les bornes supérieures de l'algorithme de $B\&B$ pour ATSP ont été améliorées par Turkensteen

(2007) par un réajustement itératif qui est une procédure de construction permettant d'obtenir de bonnes solutions réalisables pour ATSP. La procédure de réajustement itératif permet de réduire le nombre de sous-problèmes dans l'arbre $B\&B$, de sorte que les temps de calcul nécessaires soient plus petits.

Méthodes des plans sécants (Cutting plane)

C'est une alternative à la méthode de séparation et d'évaluation utilisée également pour résoudre des problèmes en nombres entiers. L'idée fondamentale est d'ajouter des contraintes adéquates appelées *coupes* à un problème linéaire jusqu'à ce que la solution réalisable de base optimale prenne des valeurs en nombres entiers. Une coupe relativement à une solution partielle courante satisfait les critères suivants :

1. Chaque solution réalisable en nombres entiers est réalisable pour la coupe, et
2. La solution partielle courante n'est pas réalisable pour la coupe.

Il existe différentes manières de générer les coupes. On cite, les coupes de Gomory, produit des coupes de n'importe quel tableau de la programmation linéaire. Ceci a l'avantage de résoudre n'importe quel problème mais son inconvénient est qu'elle est très lente. La deuxième approche est d'employer la structure du problème pour produire de très bonnes coupes. Elle a besoin d'une analyse au cas par cas, mais peut fournir des techniques de résolution très efficaces.

Méthode de Branch & Cut

Padberg et Rinaldi [51] ont amélioré l'idée du $B\&B$ basé sur la Programmation Linéaire (PL) en décrivant une méthode utilisant des inégalités renforçant la relaxation par PL (inégalités valides pour l'enveloppe convexe des solutions entières). Ils ont appelé cette technique *Branch & Cut* ($B\&C$) (méthode de recherche par séparation et évaluation renforcée efficacement par l'adjonction de coupes). Depuis, beaucoup d'implémentations ont été réalisées en utilisant cette approche.
Cette méthode nécessite la transformation du problème d'optimisation en problème de programmation linéaire en nombres entiers. Malheureusement, il est généralement difficile, voir impossible, d'énumérer toutes les inégalités résultant du problème d'optimisation, même si on se restreint à celles décrivant l'enveloppe convexe au niveau de la région optimale.
Une fois l'éventuelle transformation du problème effectuée, on utilise le fait que le problème de programmation linéaire en variables réelles est plus facile à résoudre que celui en variables entières.
L'algorithme de $B\&C$ utilise cette spécificité afin d'accélérer la recherche.
Le principe de base est le même que pour le $B\&B$, mais au lieu de $'brancher'$ sur une variable

pour descendre dans l'arbre de recherche, on peut rechercher également des ′plans de coupes′ qui permettent de restreindre l'espace des solutions réalisables. Au départ de l'algorithme, on se restreint à un petit ensemble d'inégalités du problème que l'on résout en nombres réels. Puis, si la solution optimale obtenue pour le problème en nombres réels est :

Entière : on coupe la branche étudiée, et on met à jour éventuellement la valeur de la solution optimale.

Non entière : on applique un algorithme de séparation et/ou des heuristiques pour extraire un ensemble d'inégalités violées. Si cet ensemble est vide, une solution entière avec ce coût est recherchée, sinon il faut mettre à jour l'ensemble des inégalités et continuer l'algorithme. Le but est d'exclure de manière ′intelligente′ la solution optimale non entière de l'espace de recherche, à l'aide de plans de coupes.
Si le problème résolu ne possède pas de solution réalisable, on coupe la branche.

Les méthodes de génération de plans de coupes peuvent être réalisées de manières diverses comme par combinaison linéaire des inégalités du problème de PL [33], en se servant de l'algorithme du simplexe défini par Dantzig, ou de manière spécifique au problème.
Quand la génération de plan de coupe échoue, et que le sous-problème ne peut être élagué selon la valeur d'un objectif, il faut alors ′brancher′. L'opération de branchement est réalisée en spécifiant un ensemble d'hyperplans qui divisent le sous-problème courant de telle sorte que la solution optimale réelle courante ne soit réalisable dans aucun des sous-problèmes de relaxation PL générés (par exemple, si x_i, pour la solution optimale réelle, vaut 2.7, on pourra ′brancher′ en ajoutant la contrainte $x \leq 2$ pour une branche et $x \geq 3$ pour l'autre).

1.2.3.2 Les méthodes approchées

Les méthodes de résolution approchées sont généralement utilisées là où les méthodes exactes échouent. En effet, une résolution exacte nécessite de parcourir l'ensemble de l'espace de recherche, ce qui devient irréalisable lorsque l'on veut résoudre de gros problèmes. Dans ce cas, une exécution partielle de l'algorithme exact permet rarement d'obtenir une solution de bonne qualité. Des méthodes de résolution approchée ont été mise au point afin de procurer rapidement des solutions de bonne qualité mais non optimales. Les heuristiques peuvent être réparties en trois classes : les heuristiques de construction, les heuristiques de réparation, les heuristiques d'amélioration et les algorithmes composés (qui combinent les deux premières). Les heuristiques de construction élaborent graduellement la tournée en ajoutant une ville (nœud) à chaque étape. Elles s'arrêtent dès que la solution est trouvée et n'essayent pas de l'améliorer. Les heuristiques d'amélioration consistent, une fois qu'une tournée est générée par une heuristique de construction, à l'améliorer pour obtenir une tournée de qualité meilleure. Les algorithmes de recherche locale 2-opt et 3-opt sont des exemples les plus communément utilisés. Dans cette catégorie, il

y a également l'algorithme de la recherche tabou, le recuit simulé et les algorithmes génétiques. L'heuristique de colonie de fourmis combine les caractéristiques des deux classes.

Heuristiques de construction

Il existe de nombreuses procédures de construction heuristiques d'une solution au TSP. Parmi les plus courantes, on peut citer :

1. **L'heuristique du plus proche voisin.**
   Cet algorithme glouton construit le cycle en faisant croître une chaîne. On part d'un sommet arbitraire à partir duquel on va au sommet voisin le plus proche, puis de celui-là à son plus proche voisin non visité, etc ..., jusqu'à ce que tous les sommets aient été parcourus, où l'on revient au départ. Cet algorithme (valable pour les graphes complets) est en $O(n^2)$, et les solutions qu'il fournit peuvent être arbitrairement mauvaises. En effet, cette procédure commence généralement par faire de très bons choix en sélectionnant des arêtes de poids faible, mais, vers la fin, la chaîne doit ensuite aller visiter des sommets qui ont été "oubliés", et des distances importantes sont alors rajoutées à la chaîne.
2. **Les algorithmes d'insertion.**
   Ces algorithmes fonctionnent de la manière suivante : on part d'un tour réduit à quelques sommets, puis on sélectionne un sommet hors du tour que l'on insère entre deux sommets du tour (à la place qui fait augmenter le moins possible la longueur totale de celui-ci). On insère ainsi tous les sommets jusqu'à obtenir un circuit de longueur n. Le choix du sommet à insérer peut être fait suivant de nombreux critères :
   - *plus proche (resp. plus lointaine) insertion*
     on peut insérer le sommet dont la distance minimale à un sommet du tour est minimale (resp. maximale),
   - *insertion au hasard*
     on peut insérer un sommet choisi au hasard,
   - *insertion de coût minimal (resp. maximal) :*
     on peut insérer le sommet qui fera le moins (resp. le plus) augmenter la longueur du tour partiel.
3. **Heuristique de Christofides** :
   C'est l'heuristique de construction de tournée la plus connue depuis 1976, l'algorithme associé est particulièrement élégant et fait usage de résultats classiques de la théorie des graphes.
   Principe :
   (a) Trouver un arbre couvrant minimal T,

(b) Ajouter un couplage de poids minimun M sur les sommets de degré impair de l'arbre couvrant T,

(c) Rajouter au graphe G les arêtes appartenant à la fois à T et à M. Trouver alors un chemin eulérien dans ce graphe.

(d) Raccourcir la tournée si elle passe plusieurs fois par un même sommet.

L'étape 1 peut être réalisée en $O(n^2)$. Le couplage de poids minimum peut être

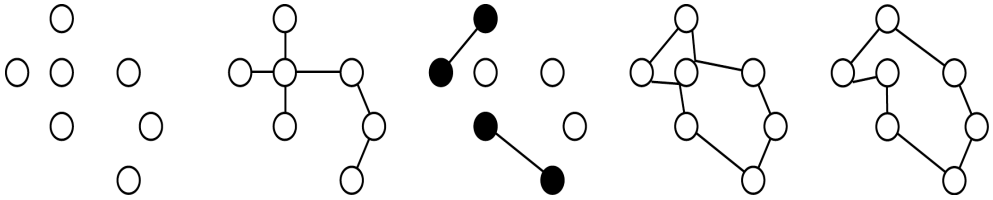


FIG. 3.3 – Heuristique de Christofides

trouvé en $o(n^3)$ et la dernière étape peut être effectuée en un temps linéaire.

Ce qui est remarquable c'est la garantie de cet algorithme. En effet, il a été montré, que toute solution obtenue avec cette algorithme ne peut pas être éloignée à plus de 50% de l'optimale.

Heuristiques d'amélioration

2-opt heuristique : (Croes, 1958)

À partir d'un cycle hamiltonien, cet algorithme retire 2 arêtes du cycle, et calcule un autre cycle en remplaçant ces arêtes par celles appropriées.

On remplace deux arêtes non consécutives $[V_i, V_{i+1}]$ et $[V_j, V_{j+1}]$ du cycle par $[V_i, V_j]$ et $[V_{i+1}, V_{j+1}]$, à condition que le coût du cycle diminue.

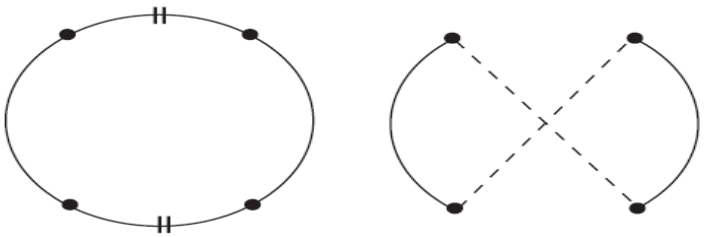


FIG. 3.4 – Heuristique 2-opt

3-opt heuristique : (Bock, 1958)

$3-Opt$ est plus complexe car il y a plusieurs façons de reconstruire le cycle après avoir enlevé trois arêtes. Cependant, il est meilleur en moyenne que $2-Opt$.

Dans le cas de $3-Opt$, on supprime trois arêtes non consécutives $[V_i, V_{i+1}]$, $[V_j, V_{j+1}]$

et $[V_k, V_{k+1}]$. Il existe alors plusieurs façons de reconstruire le cycle. Evidemment, nous garderons les solutions que si le coût du cycle diminue.

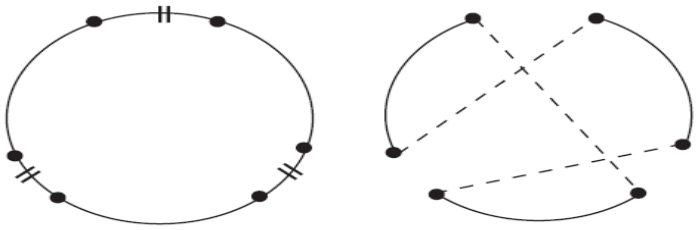


FIG. 3.5 – Heuristique 3-opt

k-Opt : Le voisinage le plus connu du problème du voyageur de commerce est appelé le $k - Opt$, il consiste à enlever k arêtes et à reformer un cycle en ajoutant k autres arêtes. L'énumération complète de ce type de voisinage est en $O(n^k)$.

3.2 Algorithmes de résolution du problème du voyageur de commerce par génération d'inégalités valides

L'utilisation d'algorithmes de génération d'inégalités valides pour la résolution de problèmes d'optimisation en nombres entiers remonte aux travaux de Dantzig, Fulkerson et Johnson [23] sur le voyageur de commerce. La méthode, devenue depuis classique, consiste, à partir d'une formulation du problème où seules sont présentes les contraintes sur les degrés des sommets, à résoudre ce problème à l'optimum et, si la solution obtenue n'est pas un cycle hamiltonien, à ajouter une inégalité valide permettant d'éliminer cette solution. La méthode de base utilise comme inégalités valides les inégalités de Gomory et les inégalités de sous-tours. Beaucoup d'autres classes d'inégalités valides ont été développées depuis, par exemple les inégalités de "peigne" [33]. Applegate et al.[1] ont développé une nouvelle méthode de séparation d'inégalités valides pour le voyageur de commerce sortant délibérément de l'utilisation de classes d'inégalités valides prédéfinies. L'approche utilisée consiste à générer une inégalité valide coupante, la meilleure existante, sur un sous-espace du voyageur de commerce initial ; puis à rendre cette inégalité, par diverses méthodes, valide pour le problème de départ tout en lui conservant son caractère coupant. La méthode s'est avérée très efficace et ce en particulier sur des problèmes de grande taille jugés difficiles.

3.2.1 Séparation des inégalités valides

Le *problème de séparation* pour une famille de contraintes consiste à déterminer pour un point x donné, si x vérifie les contraintes de cette famille, et sinon à trouver une contraintes

violée par x.
Dans ce qui suit, on présente certaines familles d'inégalités, on discutera également du problème de séparation associé.

3.2.2 Inégalités symétriques

Puisque le STSP représente un cas particulier du ATSP, alors toute inégalité $\overline{\alpha}y = \sum_{e\in E}\overline{\alpha}_e y_e \leq \alpha_0$ (défini pour le STS polytope associé au graphe complet non orienté $G_E = (V, E)$) a une équivalence ATS évidente qui est $\alpha x = \sum_{(i,i)\in A}\alpha_{ij}x_i \leq \alpha_0$ obtenue en considérant $\alpha_{ij} = \alpha_{ji} = \overline{\alpha}_e$ pour chaque $e = [i, j] \in E$. Inversement, toute inégalité ATS $\alpha x \leq \alpha_0$ étant symétrique (dans le sens où $\alpha_{ij} = \alpha_{ji}$ pour tout $(i, j) \in A$, $i < j$) a un STS équivalent $\overline{\alpha}y = \sum_{e\in E}\overline{\alpha}_e y_e \leq \alpha_0$ obtenue on posant $\alpha_e = \alpha_{ij}(= \alpha_{ji})$ pour tout $e = [i, j] \in E$. Il peut facilement être démontré que les transformations citées ci-dessus entre le STS et les inégalités ATS symétriques préservent la validité (mais pas nécessairement la propriété de définition de facette). En conséquence, le polytope ATS hérite de la version non orientée toutes les classes d'inégalités valides du STS. Notons cependant que les inégalités symétriques ne sont qu'une petite fraction de l'ensemble des (facettes) ATS inégalités.
On présente ci-dessous, deux familles d'inégalités symétriques les plus élémentaires, à savoir, les inégalités de peigne et les inégalités d'arbres de cliques.

Inégalités de peigne (Comb inequalities)

Cette famille contient toutes les contraintes traduisant un problème de parité. Les contraintes de peigne ont été introduites par Chvátal (1973) [18] et Grötschel et Padberg (1979)[40] pour le problème du voyageur de commerce. Elles jouent un rôle très important dans la résolution du TSP par *Coupes et Branchements*. Un peigne est défini par un manche H et des dents $T_1, \ldots, T_s$ ayant les propriétés suivantes

(i) $H, T_1, T_2, \ldots, T_s \subseteq V$,

(ii) $T_j \backslash H \neq \emptyset \ \forall 1 \leq j \leq s$,

(iii) $T_j \cap H \neq \emptyset \ \forall 1 \leq j \leq s$,

(iv) $T_i \cap T_j = \emptyset \ \forall 1 \leq i < j \leq s$,

(v) $s \geq 3$ et impair.

On peut définir alors la contrainte suivante

$$x(A(H)) + \sum_{i=1}^{s} x(A(T_i)) \leq |H| + \sum_{i=1}^{s} |T_i - 1| - \frac{s+1}{2}$$

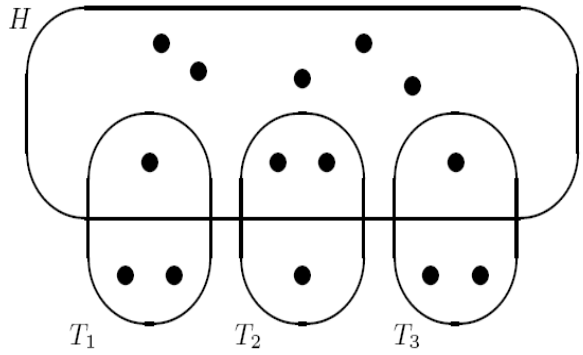


FIG. 3.6 – peigne

Si, de plus, $|T_j \cap H| = 1$, pour tout j, alors les inégalités sont réduites aux inégalités classiques du 2-couplage de Edmonds[23].
Les inégalités de peignes forment des facettes du P_T^n quand $n \geq 6$ [27].

Il n'existe pas d'algorithme polynomial permettant la séparation des inégalités de peigne, mais toutefois, on peut considérer certaines méthodes heuristiques.

Inégalités d'arbres de cliques

Soit l'arbre de cliques C dans G muni des manches $H_1, \ldots, H_r$, $r \geq 2$ et d'un ensemble de dents $T_1, \ldots, T_s$, l'inégalité d'arbre de clique est sous la forme

$$\sum_{i=1}^{r} x(A(H_i)) + \sum_{j=1}^{s} x(A(T_j)) \leq \sum_{i=1}^{r} |H_i| + \sum_{j=1}^{s} (|T_j| - t_j) - \frac{s+1}{2} = s(C)$$

Où t_j représente le nombre de manches qui ont une intersection non vide avec la dent T_j, $j = 1, \ldots, s$, $s(C)$ s'appelle taille de C.
Ces inégalités ont été introduites par Grötschel et Pulleyblank en 1986 [41] de plus, elles définissent des facettes du P_T^n pour $n \geq 6$ [26].

Les inégalités des arbres de cliques sont séparées heuristiquement, le principe est le suivant. Supposons que l'on possède un ensemble $S = \{H_1, T_1, \ldots, T_{2k+1}\}$ définissant une inégalité de peigne non violée. En commençant par la dent qui possède le plus grand cocycle on essaye de construire un ou plusieurs manches, en partant d'un sommet de la dent vérifiant certaines conditions. Pour chaque manche construit, on cherche des dents pour compléter l'arbre de clique. On répète ceci pour chaque dent du peigne initial dont le cocyle est supérieur à une certaine valeur.

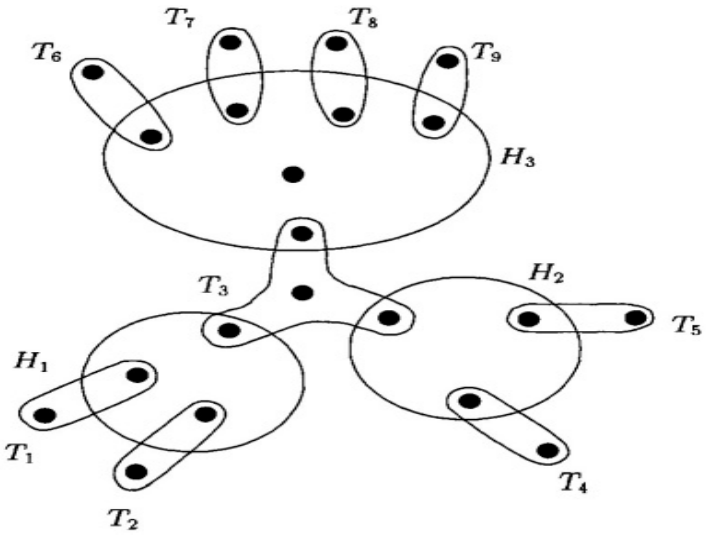


FIG. 3.7 – Arbre de clique muni de r=3 manches et de s=9 dents

3.2.3 Inégalités asymétriques

Nous allons à présent, présenter les principales catégories d'inégalités ATS asymétriques.

Définition de la procédure de *lifting*

Connaissant une contrainte valide sur un graphe, *lifter* cette contrainte consiste à en déduire une contrainte valide sur le graphe obtenu en ajoutant des arêtes (resp. arcs) et/ou des sommets. Cette procédure s'avère très utile dans l'étude polyédrale des problème rencontrés.

Inégalités D_k^+ et D_k^-

L'inégalité D_k^+ est due à Grötschel et Padberg [39], elle est de la forme :

$$x_{i_1 i_k} + \sum_{h=2}^{k} x_{i_h i_{h-1}} + 2\sum_{h=2}^{k-1} x_{i_1 i_h} + \sum_{h=3}^{k-1} x(\{i_2, \ldots, i_{h-1}\}, i_h) \leq k-1$$

où $(i_1, \ldots, i_k)$ sont des séquences de $k \in \{3, \ldots, n-1\}$ nœuds distincts.

Les inégalités D_k^+ sont des facettes induites de P. Comme le prouve Fischetti [9], elles sont obtenues par lifting des inégalités de cycle $\sum_{(i,j)\in C} x_{ij} \leq k-1$ associées au circuit $C = \{(i_1, i_k), (i_k, i_{k-1}), \ldots, (i_2, i_1)\}$.
Le problème de séparation pour la classe des inégalités D_k^+ considéré pour une séquence de nœuds $(i_1, \ldots, i_k)$, $1 \leq k \leq n-1$, dont le degré de violation

$$x^*_{i_1 i_k} + \sum_{h=2}^{k} x^*_{i_h i_{h-1}} + 2\sum_{h=2}^{k-1} x^*_{i_1 i_h} + \sum_{h=3}^{k-1} x^*(\{i_2, \ldots, i_{h-1}\}, i_h) - k + 1 \qquad (3.12)$$

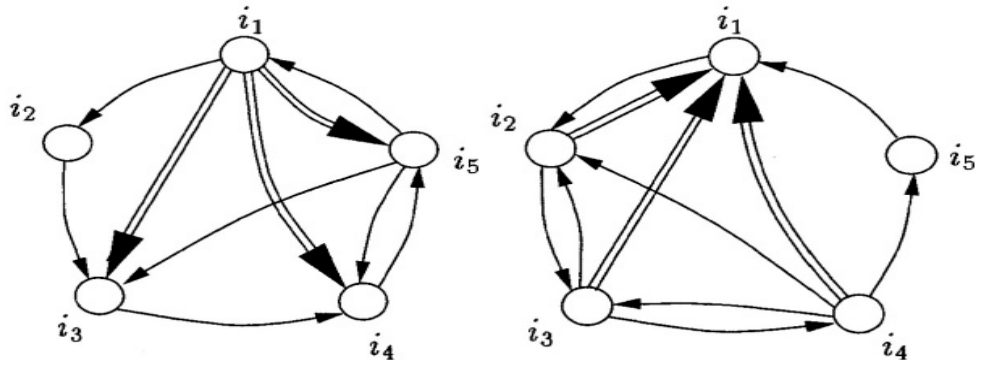


FIG. 3.8 – Graphe associé à l'inégalitée D_k^+ et D_k^- pour k=5

est aussi grand que possible.

Ce problème d'optimisation combinatoire peut être résolu par le schéma d'énumération suivant.

On commence par une séquence de nœuds vide, puis, on prolonge itérativement la séquence en cours de toutes les manières possibles et on évalue le degré de violation pour l'inégalité D_k^+ correspondante. Le processus peut être vu comme les décisions prisent sur l'arbre de branchement. Le nœud racine de l'arbre (c'est-à-dire au niveau zéro) représente la séquence vide. Chaque nœud situé au niveau k $(1 \leq k \leq n-1)$ correspond à une séquence de type $(i_1, i_2, \ldots, i_k)$, lorsque $k \leq n-2$, chacun de ces nœuds génère $n-k$ nœuds fils, un pour chaque extension possible de la séquence $(i_1, i_2, \ldots, i_k, i_{k+1})$. Une énumération exhaustive de tous les nœuds de l'arbre est clairement impossible, même pour de petites valeurs de n.

D'autre part, un très grand nombre de ces nœuds peuvent être taillés (avec le sous-arbre associé) et cela en calculant la borne supérieure. Soit $(i_1, i_2, \ldots, i_k)$, $1 \leq k \leq n-2$, la séquence associée au nœud de branchement courant, appelé μ, et soit ϕ_{max} le degré de violation maximum trouvé jusqu'à présent dans la procédure d'énumération. On considère n'importe quel nœud μ ayant une séquence de type $(i_1, \ldots, i_k, i_{k+1}, \ldots, i_m)$. À partir de la définition (1.12) , on a :

$$\begin{aligned} \phi(\quad & i_1 \quad , \ldots, i_k, i_{k+1}, \ldots, i_m) \\ \leq\ & \pi(i_1, \ldots, i_k) + x^*_{i_{k+1} i_k} + [x^*(i_1, V) - 1] + \sum_{h=k+1}^{m-1} [x^*(V, i_h) - 1] \\ =\ & \pi(i_1, \ldots, i_k) + x^*_{i_{k+1} i_k} \end{aligned} \tag{1.13}$$

Où

$$\pi(i_1, \ldots, i_k) = \sum_{h=2}^{k} x^*_{i_h i_{h-1}} + \sum_{h=2}^{k} x^*(\{i_1, \ldots, i_{h-1}\}, i_h) - k + 1. \tag{1.14}$$

On remarque que $\pi(i_1, \ldots, i_k)$ ne peut pas dépasser le degré de violation de $SEC($ Contrainte d'élimination de sous-tours) associé à l'ensemble de nœuds $S = \{i_1, \ldots, i_k\}$, d'où l'on a

$\pi(i_1, \ldots, i_k) \leq 0$ chaque fois que toutes les contraintes d'élimination de sous-tours sont satisfaites par x^*.
Selon la contrainte (1.13), les seuls nœuds fils de μ qui doivent être générés sont ceux associés à la séquence $(i_1, \ldots, i_k, i_{k+1})$ tel que

$$x^*_{i_k i_{k+1}} > \phi_{max} - \pi(i_1, \ldots, i_k).$$

Notons que les deux quantités $\phi(i_1, \ldots, i_k)$ et $\pi(i_1, \ldots, i_k)$ peuvent être calculées paramétriquement tout au long de l'arbre de branchement comme suit :

$$\phi(i_1, \ldots, i_k) = \phi(i_1, \ldots, i_{k-1}) + x^*_{i_1 i_k} + x^*_{i_k i_{k-1}} + x^*(\{i_1, \ldots, i_{k-2}\}, i_{k-1}) - 1$$

et

$$\pi(i_1, \ldots, i_k) = \pi(i_1, \ldots, i_{k-1}) + x^*_{i_k i_{k-1}} + x^*(\{i_1, \ldots, i_{k-1}\}, i_k) - 1$$

Où $\phi(i_1) = \pi(i_1) = 0$ pour toute séquence de singleton (i_1). La restriction (1.14) est très efficace en pratique et réduit de façon spectaculaire le nombre de nœuds générés dans l'énumération. Néanmoins, dans certain cas on peut être intéressé à réduire davantage le temps de calcul passé dans la procédure. Pour ce faire, avant d'exécuter l'énumération exacte décrite ci-dessus, on applique la version *tronquée* dans laquelle chaque nœud situé au niveau $k \geq 2$ génère au plus un nœud fils associé à la séquence $(i_1, \ldots, i_k, i_{k+1})$, pour i_{k+1} satisfaisant $i^*_{i_{k+1} i_k} > 0$ et $i^*_{i_1 i_{k+1}} + i^*_{i_{k+1} i_k}$ est aussi grand que possible.

Également abordé dans [17] les inégalités D_k^- :

$$x_{i_k i_1} + \sum_{h=2}^{k} x_{i_{h-1} i_h} + 2 \sum_{h=2}^{k-1} x_{i_h i_1} + \sum_{h=3}^{k-1} x(i_h, \{i_2, \ldots, i_{h-1}\}) \leq k - 1$$

où $(i_1, \ldots, i_k)$ sont des séquences de $k \in \{3, \ldots, n-1\}$ nœuds distinct.
Les inégalités D_k^- sont des facettes induites de P, et sont obtenues par la suppression des inégalités de cycle $\sum_{(i,j) \in C} x_{ij} \leq k-1$ associées aux circuits $C = \{(i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, i_1)\}$. Les inégalités D_k^- peuvent être considérées comme provenant de l'inégalité D_k^+ en échangeant le coefficient des deux arcs (i, j) et (j, i) pour tout $i, j \in V$, $i < j$. C'est une opération tout à fait générale appelée *Transposition* (pour plus de détails voir [39]).
L'algorithme de séparation exacte considéré pour les D_k^+-inégalités peut aussi être utilisée pour les D_k^--inégalités.

Les T_k-inégalités

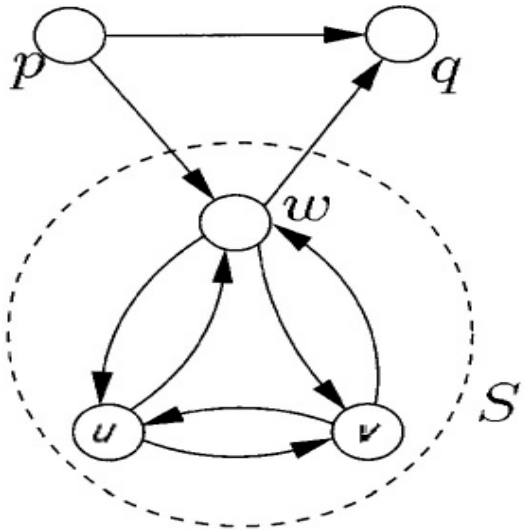
Soit $S \subset V$ un ensemble de sommets de $G = (V, A)$, $2 \leq |S| = k \leq n-2$, et soient $w \in S$, p,$q \in V \setminus S$, alors

$$x(A(S)) + x_{pw} + x_{pq} + x_{wq} \leq k \tag{3.13}$$

appelé T_k**-inégalités** ont été introduites par Grötschel et Padberg [39].
Pour $n \geq 4$, les T_k**-inégalités** définissent des facettes du P_T^n si $2 \leq k \leq n-2$, excepté pour $k = n-3$.
Par exemple, T_3-inégalité pour $S = \{u, v, w\}$ s'écrit

$$x_{uv} + x_{vu} + x_{uw} + x_{wu} + x_{vw} + x_{wv} + x_{pw} + x_{pq} + x_{wq} \leq 3$$

le digraphe correspondant est illustré dans la figure ci-dessus.



Les T_k-inégalités sont séparées heuristiquement. On utilise donc les étapes suivantes correspondant à la procédure de séparation heuristique pour la détection des inégalités violées.
L'idée consiste à essayer de *trouver* les nœuds p, w, q et un ensemble de nœuds S qui seront susceptibles de violer l'inégalité T_k.

- Dans la première étape, on détermine les ensembles de nœuds S pour $|S| \geq 2$ et $x(A(S)) = |S| - 1$. Cela est fait en énumérant les nœuds i, $j \in V$ tel que $x_{ij} + x_{ji} = 1$, réduisant ainsi ces nœuds à un seul nœud et appliquant itérativement cette procédure d'énumération au digraphe réduit résultant.
  Chaque contraction effectuée est stockée dans le digraphe réduit $D_s = (V, A_s)$, où un arc $(i, j) \in A_s$ est défini à chaque fois que les groupes i, j, sont réduits à un nouveau groupe j. L'itération dans laquelle l'arc est généré est sauvegardée sous le poids de l'arc (i, j).
  Notant que le degré sortant de chaque nœud est au plus 1, tandis que plus d'un arc peut entrer dans un nœud. Pour un nœud donné $i \in V$, l'ensemble de nœuds S_k contenant i et satisfaisant $x(A(S_k)) = |S_k| - 1$ peut facilement être détecté.
- Dans la deuxième étape, les trois nœuds prometteurs pour T_k-inégalité sont énumérés. Afin de maintenir un temps de calcul modéré, on ne considère que les nœuds p, w, q, tels que $x_{pw} + x_{pq} + x_{wq}$ est compris entre des valeurs limites. Des tests de calcul montrent

qu'il est plus probable de trouver (dans un délai raisonnable) les T_k-inégalités violées si $1.33 \leq x_{pw} + x_{pq} + x_{wq} \leq 1.75$.
Enfin, il est à vérifier, si une combinaison des triplets (p, w, q) et S_k violent l'inégalité T_k.

Les T_k inégalités peuvent être généralisées en reliant la source p et le puits q au peigne, comme suit.

C2-inégalités

Soient un manche $H \subset V$ et un ensemble de dents $T_1, \ldots, T_s \subset V$ satisfaisant : **(a)** $|H \cap T_i| \geq 1$, **(b)** $|T_i \setminus H| \geq 1$, pour tout $i = 1, \ldots, s$, **(c)** $|T_i \cap T_j| = \emptyset$, $1 \leq i < j \leq s$, et **(d)** $s \geq 3$ et impair. Pour chaque paire de sommets distincts p et q dans $(V \setminus H) \setminus (\bigcup_{i=1}^{s} T_i)$, l'inégalité suivante

$$x(A(H)) + \sum_{i=1}^{s} x(A(T_i)) + \sum_{v \in H} (x_{pv} + x_{vq}) + x_{pq} \leq |H| + \sum_{i=1}^{s} (|T_i| - 1) - \frac{s+1}{2} + 1 (= s(C) + 1)$$

est appelée $C2$-inégalité, de plus elle est valide pour le P_T^n.

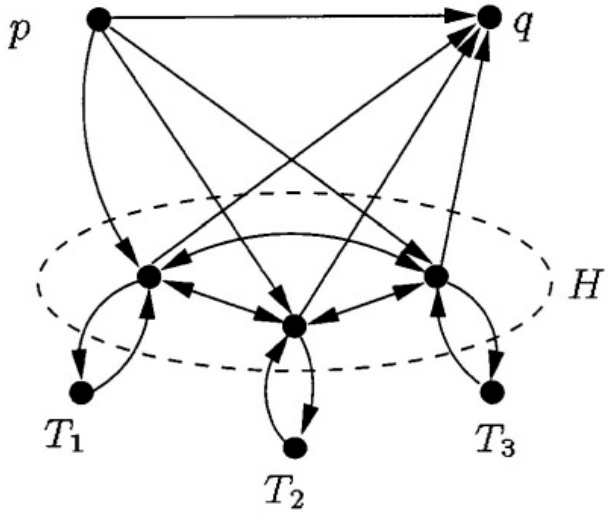


FIG. 3.9 – Graphe associé à l'inégalité C2

C3-inégalités

Soient $i_1,\ i_2,\ i_3$ trois sommets différents et soient $W_1,\ W_2$ deux sous-ensembles de V tels que **(a)** $W_1 \cap W_2 = \emptyset$, **(b)** $W_1 \cap \{i_1,\ i_2,\ i_3\} = \{i_1\}$, **(c)** $W_2 \cap \{i_1,\ i_2,\ i_3\} = \{i_2\}$, **(d)** $|W_j| \geq 2$,

$j = 1,\ 2$. Alors

$$x(A(W_1)) + x(A(W_2)) + \sum_{j \in W_2} x_{i_1 j} + x_{i_2 i_1} + x_{i_3 i_1} + x_{i_3 i_2} \leq |W_1| + |W_2| - 1$$

est appelée $C3$-inégalité.

Les arcs ayant un coefficient positif dans la $C3$-inégalité sont représentés dans la figure qui suit

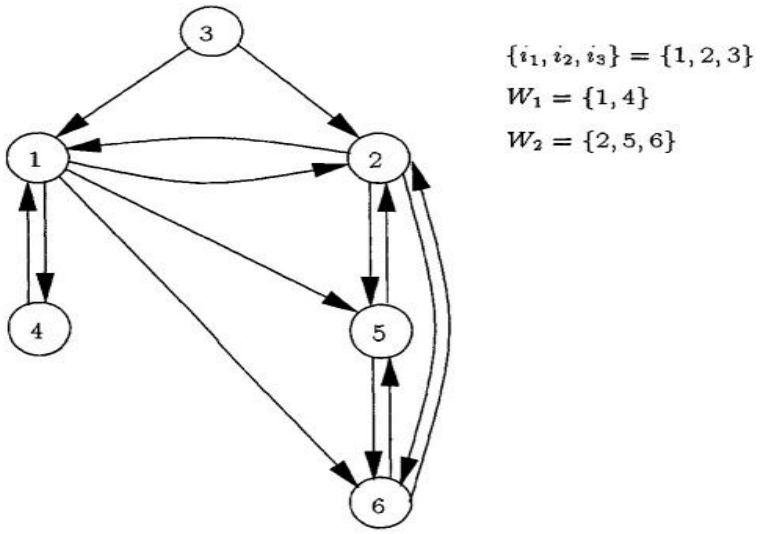


FIG. 3.10 – Graphe associé à l'inégalité C3

Inégaliés Odd CAT

Une classe importante d'inégalités, qui se révèle être très utile dans la pratique, a été introduite par Balas [7]. Pour ce faire, on a besoin de définir la notion de *compatibilité* afin de décrire les inégalités associées à cette classe.

Deux arcs distincts (i, j) et (u, v) sont dit *incompatibles* si $i = u$ ou $j = v$, ou $i = v$ et $j = u$, compatible sinon.

Closed Alternating Trail (CAT) est une séquence $T = \{a_1, \ldots, a_t\}$ de t arcs distincts telle que, pour $k = 1, \ldots, t$, l'arc a_k est incompatible avec les arcs a_{k-1} et a_{k+1}, et compatible avec tous les autres arcs dans T (avec $a_0 := a_t$ et $a_{t+1} := a_1$).

Soient $\delta^+(v)$ et $\delta^-(v)$ l'ensemble des arcs sortant et entrant du nœud $v \in V$ respectivement. Étant donné un T CAT, le nœud v est appelé *nœud source* si $|\delta^+(v) \bigcap T| = 2$ et *nœud puits* si $|\delta^-(v) \bigcap T| = 2$. Notant qu'un nœud peut jouer le rôle du nœud source et puits à la fois. Soit Q l'ensemble des arcs $(i, j) \in A \backslash T$ tel que i est le nœud puits. Pour tout CAT de longueur t impaire, l'inégalité odd CAT

$$\sum_{(i,j) \in T \bigcup Q} x_{ij} \leq \frac{|T| - 1}{2}$$

est valide pour P_T^n. La figure qui suit illustre quelques exemples des odd closed alternating trails.

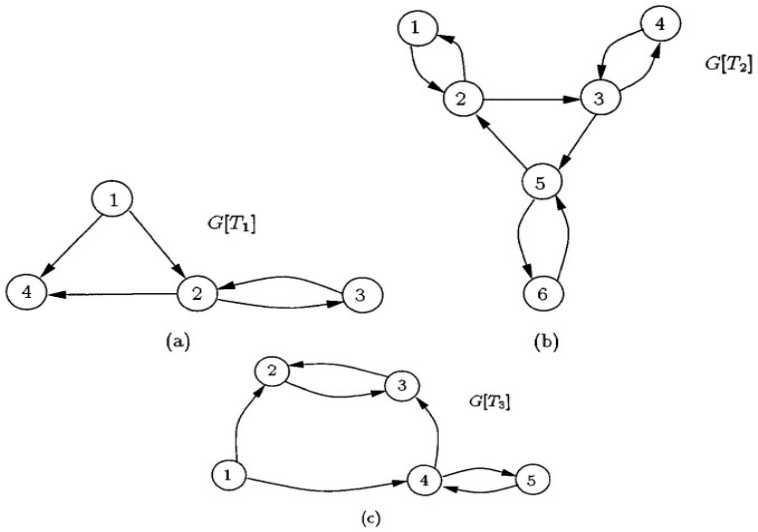


FIG. 3.11 – Quelques Odd closed alternating trails

Balas [7] a prouvé que les odd CAT inégalités définissent des facettes du P_T^n (à l'exception du cas où $n = 6$).

La prochaine étape, consiste à décrire l'algorithme de séparation heuristique pour la famille d'inégalités *odd CAT*. L'algorithme basé sur les inégalités *odd CAT* correspondant au cycle impair sur le graphe d'*incompatibilité* auxiliaire. De plus, l'algorithme de séparation peut être vu comme étant une version spécifique du schéma proposé par *Caprara* et *Fischetti* (1996) [13] pour la séparation d'une sous-classe des coupes de *Chvátal-Gomory* pour les problèmes de programmation en nombres entiers.

Étant donné un point x^*, on met en place un graphe non orienté $\tilde{G} = (\tilde{N}, \tilde{E})$ muni d'arêtes pondérées tel qu'un nœud $v_a \in \tilde{N}$ correspond à l'arc $a \in A$ pour $x_a^* > 0$ et une arête $e = [v_a, v_b] \in \tilde{E}$ représente chaque paire d'arc a, b incompatible dont le poids est défini par $w_e = 1 - (x_a^* + x_b^*)$. Soit $\tilde{\delta}(v)$ l'ensemble des arêtes de $\tilde{E}$ incidentes au nœud $v \in \tilde{N}$. Un cycle $\tilde{C}$ de $\tilde{G}$ est un sous-ensemble d'arêtes de $\tilde{E}$ induisant un sous-graphe connexe de $\tilde{G}$ tel que $|\tilde{C} \cap \tilde{\delta}(v)|$ est pair pour tout $v \in \tilde{N}$. Le cycle $\tilde{C}$ est appelé

(i) Impair si $|\tilde{C}|$ est impair,

(ii) Simple si $|\tilde{C} \cap \delta(\tilde{v})| \in \{0, 2\}$ pour tout $v \in \tilde{N}$,

(iii) Sans-cordes si le sous-graphe de $\tilde{G}$ induit par les nœuds couverts par $\tilde{C}$ n'a pas d'arêtes autres que celles dans $\tilde{C}$.

Par construction, chaque cycle impair simple ou sans-cordes $\tilde{C}$ dans $\tilde{G}$ correspond à un T *CAT* impair, où $a \in T$ si et seulement si v_a est couvert par $\tilde{C}$.

De plus, le poids total de $\tilde{C}$ est

$$\begin{aligned} w(\tilde{C}) &= \sum_{e\in\tilde{C}} w_e = \sum |v_a, v_b| \in \tilde{C}(1 - x_a^* - x_b^*) \\ &= |H| - 2\sum_{a\in T} x_a^* \end{aligned}$$

Donc $\frac{1-w(\tilde{C})}{2}$ donne une borne inférieure sur le degré de violation de l'inégalité CAT correspondante, calculé comme suit :

$$\phi(T) = (2\sum_{a\in T\cup Q} x_a^* - |T| + 1)/2$$

L'algorithme de séparation heuristique, propose de calculer, pour tout $e \in \tilde{E}$, un cycle impair de poids minimum $\tilde{C}_e$ empruntant l'arête e. Si $\tilde{C}_e$ est simple ou sans-cordes, alors il correspond à un CAT impair appelé T. Si de plus, la borne inférieure $(1-w(\tilde{C}_e))/2$ dépasse le seuil $\theta = \frac{-1}{2}$, alors l'inégalité correspondante est violée. Par conséquent, on évalue son degré de violation actuel, $\phi(T)$, et on stocke l'inégalité si $\phi(T) > 0$.
Pour éviter de détecter deux fois la même inégalité, on enlève l'arête e de $\tilde{G}$ après le calcul de chaque $\tilde{C}_e$. Pour augmenter les chances de trouver les cycles impairs qui sont simples et sans-cordes, on s'assure que le poids de toutes les arêtes soit strictement positif.
Le point clé de l'algorithme est le calcul dans $\tilde{G}$ des poids minimum du cycle impair passant par une arête donnée. En supposant que les poids des arêtes sont tous positifs, ce problème est connu pour être résolu polynomialement, car il peut être transformé en un problème du plus court chemin.

Les inégalités SOP

Les contraintes de priorité pour $ATSP$ sont connues sous le nom de *Sequential Ordering Problem* (SOP). On résume ci-après les classes d'inégalités communément utilisées.

- **Predecessor/ Successor inégalités**
  Balas, *Fischetti* et *Pulleyblank* (1995) [9] ont introduit des classes d'inégalités qui peuvent être vues comme un renforcement de l'inégalité de sous-tours (écrites sous la forme de coupe)
  $$x(\delta^+(w)) \geq 1 \qquad \textit{pour tout } W \subset V$$
  - Soient $S \subseteq V\backslash\{1\}$,$\bar{S} = V\backslash S$, alors *predecessor* inégalités (π-inégalité) s'écrit
  $$x(S\backslash\pi(S) : \bar{S}\backslash\pi(S)) \geq 1$$
  - Soient $S \subseteq V\backslash\{1\}$, $\bar{S} = V\backslash S$ alors *successor* inégalités (σ-inégalité) s'écrit
  $$x(\bar{S}\backslash\sigma(S) : S\backslash\sigma(S)) \geq 1$$

On ne sait pas si les π-inégalités (resp. σ-inégalité) peuvent être séparées en temps polynomial, mais *Balas* et *al.*[9] ont décrit un algorithme en temps polynomial pour une plus petite classe d'inégalités, appelé π-inégalité (resp. σ-inégalité) faible. Ces inégalités sont obtenues par substitution de l'ensemble de nœud S par un seul nœud j.

Procédures de séparation exacte pour les inégalités π-inégalité faible.

Entrée : $G = (V, A)$, $P = (V, R)$ et la solution fractionnaire $\bar{x}$ du LP.

Sortie : L'inégalité π violée ou une réponse stipulant qu'une telle inégalité n'existe pas.

Pour tout $j \in V \setminus \{n\}$ avec $\pi(j) \neq \emptyset$ faire :

1. Construire un digraphe $\acute{D} = (\acute{V}, \acute{A})$ en supprimant tous les sommets dans $\pi(j)$ et tous les arcs incidents, c'est-à-dire,

   $$\acute{V}=V\backslash\pi(j), \qquad \acute{A}=\{(i,j) \in A | i,j \in \acute{V}\}.$$

2. Associer un poids à chaque arc $\bar{c}_{ij} = \bar{x}_{ij}$ $\forall (i,j) \in \acute{A}$.
3. Calculer une coupe (j,n)-coupe minimum $\delta^-(S)$ dans $\acute{D}$.
4. Si $\bar{c}(\delta^-(S)) < 1$ alors j et S viole l'inégalité π
   sinon aucune π-inégalité faible n'est violée en considérant j.

Le même principe de séparation est appliqué aux inégalités σ et (π, σ).

3.2.4 Autres inégalités valides pour le polytope ATSP(G)

Inégalités de sous-tours (SEC's)

Dans l'algorithme de résolution il est impératif qu'aucune contrainte de sous-tours ne soit violée avant de passer à d'autres algorithmes de séparation ou à d'autres phases du "Branch and Cut". Cette inégalité aussi appelée inégalité de coupe est formulée comme suit

$$\sum_{(i,j)\in A: i,\, j\in S} x_{ij} \leq |S| - 1$$

Les problèmes de séparation associés aux contraintes d'élimination des sous-tours sont résolus par des procédures développées pour le TSP. Cette démarche est motivée par le fait que chaque inégalité du TSP $by \leq b_0$ pourrait être transformée en inégalité symétrique du ATSP $ax \leq a_0$ et cela en substituant y_{ij} avec $x_{ij} + x_{ji}$ et en fixant $a_0 = b_0$, $a_{ij} = a_{ji} = b_{ij}$, et vice versa. Pour cette raison, tous les algorithmes de séparation pour les inégalités du TSP peuvent ainsi être utilisés pour le ATSP.

Supposons qu'on a un point $\bar{x}$. La première étape consiste alors à poser

$$\bar{y}_e = \bar{x}_{ij} + \bar{x}_{ji} \quad \textit{pour tout}\, e = ij \in E$$

et ensuite appliquer l'algorithme de séparation du TSP à $\bar{y}$. Si une inégalité violée est détectée, le problème est transformé en un problème du ATSP homologue. Dans les dernières années plusieurs procédés de séparation exacte et heuristique pour les inégalités du TSP ont été proposés. Ils peuvent tous être utilisés pour le ATSP et ainsi pour le HTSP. Nous allons à présent détailler la procédure utilisée pour la séparation les inégalités de sous-tours.

La séparation des inégalités du sous-tour est relativement simple. Etant donnés un graphe $G = (V, E)$ et un vecteur $x \in \mathcal{R}_+^E$, le problème de recherche d'inégalités de coupes violées par x, se ramène à la recherche d'une coupe minimum dans G, où x représente les capacités des arêtes de G. En utilisant un algorithme polynomial pour le problème de flot maximum [22, 24] et le fameux théorème de flot maximum-coupe minimum (cf. Ford et Fulkerson [29]), on peut trouver une coupe minimum dans G en temps polynomial. Un des plus performants algorithme de flot maximum est celui de Goldberg et Tarjan [32]. Nous utilisons cet algorithme et le code correspondant de Goldberg et Tarjan pour séparer les inégalités de coupes.
Soit $V = \{v_1, \ldots, v_n\}$. Coupe-Min(G, x, v_1, v_i) est une fonction qui utilise l'algorithme de Goldberg et Tarjan [32] pour la recherche d'une coupe minimum dans G séparant v_1 de v_i. En faisant varier $i = 2, \ldots, n$, le minimum des valeur données par Coupe-Min(), nous donne la valeur de la coupe minimum dans G. Ici, on s'arrête dès que l'on trouve une coupe ayant une capacité strictement inférieure à 2. Et donc notre procédure de séparation des inégalités de coupes COUPE(), s'écrit comme suit
COUPE(G, x)
étape 0. $i \leftarrow 2$;
étape 1. Tant que Coupe-Min$(G, x, v_1, v_i) \geq 2$ et $i \leq n$ faire
$i \leftarrow i + 1$;
étape 2. Si $i \leq n$ alors la coupe séparant v_1 de v_i est violée par x, terminer.
sinon il n'existe pas d'inégalités de coupes violées par x, terminer.

4

Le problème du voyageur de commerce asymétrique avec contraintes de saut

4.1 Introduction

Le problème du voyageur de commerce asymétrique ($ATSP$) est un problème classique de l'optimisation combinatoire. Il consiste à trouver le circuit hamiltonien de coût minimal contenu dans un graphe orienté sans boucle $G = (V,\ A)$ où $V = \{1, \ldots, n\}$ et les coûts c_{ij} sont associés à chaque arc $(i,\ j) \in A$. Dans le cas asymétrique, les coûts c_{ij} et c_{ji} peuvent être différents pour tout couple $\{i \in V, j \in V\}$. Ce problème est connu pour être NP-difficile et a été très étudié dans la littérature [20, 48, 21, 35, 36, 16, 57, 56, 46].
Mais qu'en est-il si certaines villes doivent être éloignées d'au moins **s** liens dans le tour ?
Soit S l'ensemble des villes soumises à la condition de saut (i.e., dont la distance en nombre d'arêtes (resp. arcs) entre chacune de ces villes est d'au moins H). Le problème du voyageur de commerce asymétrique avec contraintes de saut ((HTSP), Hop Asymmetric Traveling Salesman Problem) consiste alors à trouver un circuit Hamiltonien de coût minimal qui respecte l'éloignement entre les villes de S.

4.2 Formulations

Soient $G = (V, A)$ un graphe orienté et $S \subset V$ un ensemble de nœuds. Étant données, une fonction C qui associe à chaque arc $(i, j) \in A$ un coût c_{ij} et une borne $1 \leq H \leq |V|$, le problème du voyageur de commerce avec contraintes de saut (HTSP) consiste à trouver un circuit de coût minimal parcourant chaque nœud de V une et une seule fois (un circuit Hamiltonien) de telle sorte que deux nœuds consécutifs de S dans le circuit sont séparés par au moins H arcs (ou saut).
Le HTSP peut être reformulé comme étant la recherche d'un circuit hamiltonien de coût minimum de telle sorte que chaque paire de noeuds de S soit à une distance au plus de $n - H$ dans le circuit.
Un problème voisin est le Black and White TSP (BWTSP) minimisant le coût du cycle hamiltonien où les nœuds consécutifs noirs doivent être séparés par **au plus** un nombre donné de nœuds blancs. À première vue, le BWTSP semble être le même problème que la version du HTSP citée ci-dessus. Cependant, la différence réside dans le fait que dans le HTSP, le saut s'effectue entre toute paire de sommets de S et pas seulement entre les sommets consécutifs noirs.

Dans ce qui suit, on note par $HTSP(G, S, H)$ le polytope associé au HTSP pour un graphe G, où les nœuds de S doivent être à une distance d'au moins H.
On définit à présent les différentes formulations du $HTSP$ et cela sous forme de programmes linéaires en nombres entiers. Les variables de décision notamment utilisées sont des variables de décision indiquant la position de chaque ville dans le tour ou des variables permettant de prendre en compte le nombre de sauts entre les villes dans le tour.

4.2.1 Formulations étendues et compactes

Formulation HTSPxh

On pose $D = \{-(n-1), \ldots, -1\} \cup \{2, \ldots, n\}$. Comme pour la formulation précédente, on dispose des variables $\mathbf{x_{ij}}$, $i, j \in V$. Pour chaque couple de ville (i, j), et pour $d \in D$, la variable $\mathbf{h_{ij}^{d}} = 1$ si le nombre de sauts entre les villes i et j dans le tour est égal à d et 0 sinon.
Le problème (HTSP) est équivalent au programme linéaire mixte suivant :

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

$$\text{s.c.} \sum_{j=1, j\neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{4.1}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{4.2}$$

$$x_{ij} + \sum_{d\in D} h_{ij}^{d} = 1 \qquad i = 1, \ldots, n, j = 2, \ldots, n, j \neq i, \tag{4.3}$$

$$\sum_{d=-(N-1)}^{-1} h_{i1}^{d} = 1 \qquad i = 2, \ldots, n \tag{4.4}$$

$$\sum_{d=2}^{n-1} h_{i1}^{d} = 0 \qquad i = 2, \ldots, n \tag{4.5}$$

$$x_{ij} + \sum_{d\in D} dh_{ij}^{d} = x_{ik} + \sum_{d\in D} dh_{ik}^{d} + x_{kj} + \sum_{d\in D} dh_{kj}^{d} \quad i = 1, \ldots, n, j, k = 2, \ldots, n, i \neq j \neq k \neq i, \tag{4.6}$$

$$x_{ij} = h_{ji}^{-1} \qquad i = 1, \ldots, n, j = 2, \ldots, ni \neq j, \tag{4.7}$$

$$x_{i1} = h_{i1}^{-(n-1)} \qquad i = 2, \ldots, n, \tag{4.8}$$

$$h_{ij}^{d} = h_{ji}^{-d} \qquad i, j = 1, \ldots, n, i \neq j, d = 2, \ldots, n-1 \tag{4.9}$$

$$x_{ij} + \sum_{d=-(s-1)}^{-1} h_{ij}^{d} + \sum_{d=2}^{s-1} h_{ij}^{d} = 0 \qquad i, j \in S, i \neq j \tag{4.10}$$

$$\sum_{d=n-s+1}^{n-1} h_{ij}^{d} = 0 \qquad i, j \in S, i \neq j \tag{4.11}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n, i \neq j. \tag{4.12}$$

$$h_{ij}^{d} \geq 0 \qquad i, j = 1, \ldots, n, i \neq j, d \in D. \tag{4.13}$$

La formulation est composée : de contraintes d'affectations (4.1)-(4.5), de contraintes (4.6) appelées contraintes triangle, de contraintes de symétrie (4.7)-(4.9), de contraintes de saut (4.10) et (4.11) et enfin de contraintes d'intégrité sur les x et de continuité sur les y.
Soit l'enveloppe convexe $P_1(G, S, H)$ définie par

$$P_1(G, S, H) = conv\{(x, y) \in \{0, 1\}^A : (x, y)\, satisfait\, (4.1) - (4.13)\}$$

4.2.2 Formulations naturelles et étendues non compactes

Formulation naturelle, HTSPx

Soit $\mathbf{x_{ij}}$, pour tout (i,j)$\in A$, une variable binaire indiquant si l'arc (i,j) est pris dans la solution. Le modèle mathématique (HTSPx) est le suivant :

$$\text{Min.} \sum_{\forall (i,j)\in A} c_{ij}x_{ij} \tag{4.14}$$

$$\text{s.c.} \sum_{j,j\neq i} x_{ij} = 1 \qquad \forall i \in V, \tag{4.15}$$

$$\sum_{i,i\neq j} x_{ij} = 1 \qquad \forall j \in V, \tag{4.16}$$

$$x(\delta^+(W)) \geq 1 \qquad \forall W \subset V, \emptyset \neq W \neq V, \tag{4.17}$$

$$x(\delta^+(W)) \geq |W \cap S| - \left\lfloor \frac{|W \setminus S|}{H-1} \right\rfloor \qquad \forall W \subset V, \emptyset \neq W \neq V, \tag{4.18}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A. \tag{4.19}$$

(4.15)-(4.17), (4.19) représente la formulation classique pour trouver un chemin Hamiltonien asymétrique de 1 à n. Les inégalités (4.15) et (4.16) garantissent qu'il existe exactement un arc entrant et un arc sortant de chaque nœud de V, tandis que les inégalités (4.17) brisent les sous-tours. L'inégalité (4.18) permet de modéliser la longueur minimale entre les nœuds de S. À savoir, soit $k = |W \cap S|$ et $u_1, u_2, \ldots, u_k$ des nœuds de $S \cap W$ et soit T la solution associée au HTSP. On Suppose sans perte de généralités, que ces nœuds apparaissent dans cet ordre dans le cycle. Cela implique que les chemins reliant toute paire de nœuds u_i et u_{i+1} pour $i = 1, \ldots, k-1$, et u_k et u_1 sont disjoints. On note par $\mathcal{P}$ cet ensemble de k chemins. Chacun de ces chemins doit avoir une longueur d'au moins H, de ce fait, il doit contenir au moins $H-1$ nœuds qui n'appartiennent pas à S. Le nombre de ces nœuds est donné par $|W \setminus S|$. Alors

$$\left\lfloor \frac{|W \setminus S|}{H-1} \right\rfloor$$

représente la longueur maximale des chemins de $\mathcal{P}$ qui peuvent être compris complètement dans le graphe induit par l'ensemble W. Il en résulte qu'au moins

$$|W \cap S| - \left\lfloor \frac{|W \setminus S|}{H-1} \right\rfloor$$

chemins de $\mathcal{P}$ empruntent des nœuds autre que ceux de W, d'où l'inégalité (4.18) induit le fait qu'au moins ce nombre d'arcs doivent quitter l'ensemble W.

Cette formulation a un nombre exponentiel d'inégalités et donc, la relaxation linéaire ne peut être calculée que s'il existe une procédure de séparation des inégalités (4.17). Il est bien connu que les inegalités (4.17) peuvent être séparées en temps polynomial par le calcul des flots

maximaux.
Afin d'étudier la complexité du problème de séparation des inégalités (4.18), une proposition à été donnée, en établissant la fonction f définie sur l'ensemble des sous-ensembles de V par $f(W) = x(\delta^+(W)) - |W \cap S| + \frac{|W \setminus S|}{H-1}$ qui est sous-modulaire. Comme la minimisation d'une fonction sous-modulaire peut être effectuée en temps polynomial, en conséquence, la séparation de la classe d'inégalités (4.18) peut être également effectuée en temps polynomial.

Formulation étendue, HTSPxy

Plusieurs formulations compactes (nombre polynomial d'inégalités et de variables) ont été introduites pour le TSP et les problèmes du VRP. Gavish et Graves [31] ont présenté une telle formulation, où les contraintes d'élimination des sous-tours sont modélisées par une unique unité de flot, où un flot de valeur $n-1$, quittant le nœud 1 doit être distribuée à chaque autre nœud avec une quantité de 1 .
L'idée étant d'essayer d'adapter cette formulation d'unique unité de flot pour modéliser la rupture des sous-tours et les inégalités de longueur bornée. De cette façon, on considère que chaque nœud de S sera une source de $s-1$ unités de flot et chaque nœud de $V \setminus S$ peut prendre ou non une unité de cette flot. Ainsi, le flot sortant de chaque nœud de S définira un chemin de longueur au moins $s-1$. Vu que l'on impose qu'il n'y ait pas de flot entrant dans les nœuds de S, les chemins définis par le flot contiennent seulement des nœuds de $V \setminus S$. Le fait que ces flots ne se croisent pas les uns les autres sera garanti par les égalités d'affectation sur les variables x. On considère l'ensemble supplémentaire de variables sur les arcs de A où y_{ij} indique la quantité de flot circulant sur l'arc (i, j). Nous avons alors la formulation suivante

$$
\begin{aligned}
\text{Min.} \sum_{\forall (i,j) \in A} & c_{ij} x_{ij} & & \\
\text{s.t.} \sum_{j} & x_{ij} = 1 & \forall i \in V, & (4.20)\\
\sum_{i} & x_{ij} = 1 & \forall j \in V, & (4.21)\\
& x(E(W)) \leq |W| - 1 & \forall W \subset V, & (4.22)\\
\sum_{j} & y_{ij} = s - 1 & \forall i \in S & (4.23)\\
\sum_{j} & y_{ji} = 0 & \forall i \in S & (4.24)\\
& 0 \leq \sum_{j} y_{ji} - \sum_{j} y_{ij} \leq 1 & \forall i \in V \setminus S & (4.25)\\
& y_{ij} = (s-1) x_{ij} & \forall i \in S, j \in V \setminus S, & (4.26)\\
& y_{ij} \leq (s-2) x_{ij} & \forall i, j \in V \setminus S, & (4.27)\\
& y_{ij} \geq x_{ij} & \forall i, j \in V \setminus S, & (4.28)\\
& x_{ij} \in \{0, 1\} & \forall (i,j) \in A. & (4.29)\\
& y_{ij} \in I\!N & \forall (i,j) \in A. & (4.30)
\end{aligned}
$$

Comme les contraintes d'élimination de sous-tours peuvent être séparées en temps polynomial et toutes les autres inégalités sont en nombre polynomial, la relaxation linéaire de cette formulation peut être calculée en temps polynomial. Il est même possible d'obtenir une formulation compacte en remplaçant les contraintes d'élimination de sous-tours par un ensemble d'inégalités en nombre polynomial impliquant un nouvel ensemble de variables.

4.3 Inégalités valides

On rappelle que le but de notre travail est de déterminer si l'on peut adapter les inégalités valides associées au $ATSP$ à notre problème ($HTSP$). Après l'étude détaillée de ces inégalités dans le chapitre précédent et la définition de notre problème sous ces différentes formulations, on constate que toute inégalité valide pour le problème du voyageur de commerce asymétrique présentée l'est aussi pour le problème du voyageur de commerce asymétrique avec contraintes de saut. Dans cette section, nous avons sélectionné quelques contraintes valides dont nous allons prouver l'efficacité et la validité dans la partie implémentation.

Inégalités DFJ

Inégalités 2-DFJ : Pour $n \geq 5$, l'inégalité qui suit

$$x_{ij} + x_{ji} \leq 1$$

définit une facette du polytope $P(G, S, H)$

Inégalités 3-DFJ :

$$x_{ij} + x_{kj} + x_{ik} + x_{ji} + x_{jk} + x_{ki} \leq 2$$

Ces deux inégalités assurent le fait qu'il ne peut y apparaître deux fois le même nœud dans une tournée, elles permettent ainsi de briser les sous tours pouvant exister.

Inégalités ATSPxycut

Soient i, j, $k \subset V$ et $d \in I\!N$. Soit $h_{ij}^d = 1$ si le nombre de sauts entre les nœuds $i$et j dans le tour est égal à d, 0 sinon. L'inégalité $ATSPxycut$ suivante

$$x_{ij} + x_{jk} + \sum_{d=2}^{n-1} h_{jk}^d + x_{kj} + x_{ki} + \sum_{d=2}^{n-1} h_{ki}^d + x_{ik} \leq 2$$

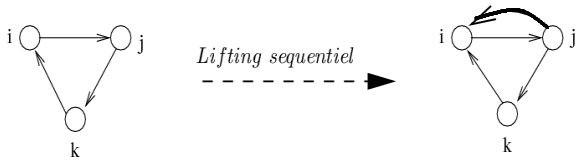
est valide pour le $P_1(G, S, H)$.
Cette inégalité permet de briser les sous tours tous en prenant compte de la notion de saut entre les nœuds et cela à travers l'introduction de la varible h_{ij}^d (voir section 4.2.1).

D_3-Inégalités

On considère le cas particulier $k = 3$ associé aux D_k-inégalités. Pour ce faire, soit un digraphe complet $D = (V, A)$ et soient i, j, $k \in V$, l'inégalité D_3 appelé aussi inégalité *3-cycles liftée* est donnée par :

$$x_{ij} + x_{jk} + x_{ki} + 2x_{ji} \leq 2$$

L'énumération de tous les nœuds possibles $i \neq j \neq k$ est réalisée en $O(n^3)$. D'autres exemples sur les inégalités de *cycles liftées* peuvent être trouvées dans [39].

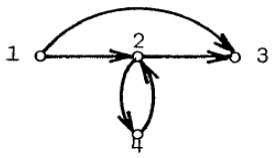


T_2-Inégalités

Un cas particulier des T_k-inégalités consiste à prendre $k = 2$.
Pour $n \geq 4$, soit $\{v, w, p, q\} \subset V$ l'inégalité T_2 est donné par

$$x_{pw} + x_{pq} + x_{wq} + x_{wv} + x_{vw} \leq 2$$

Par exemple, si on prend l'ensemble $\{4, 2, 1, 3\} \subset V$, le graphe associé à l'inégalité T_2 est illustré comme suit



Inégalités E_4

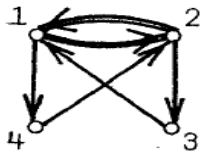
Soit $\{i_1, i_2, i_3, i_4\} \subset V$, $|V| \geq 5$; alors l'inégalité

$$2x_{i_1 i_2} + x_{i_1 i_4} + 2x_{i_2 i_1} + x_{i_2 i_3} + x_{i_3 i_1} + x_{i_4 i_2} \leq 3$$

est appelée E_4-inégalité.
Exemple : l'inégalité E_4 associé à l'ensemble $\{i_1, i_2, i_3, i_4\} \equiv \{1, 2, 3, 4\} \subset V$ est

$$2x_{12} + x_{14} + 2x_{21} + x_{23} + x_{31} + x_{42} \leq 3$$

Le graphe associé est représenté ci-dessous :



4.4 Conclusion

Nous avons présenté dans ce chapitre les différents modèles utilisés pour formaliser le problème du voyageur de commerce asymétrique avec contraintes de saut. Dans la partie résultats nous nous sommes intéressé à l'implémentation de deux formulations l'une compacte et l'autre non compacte à savoir les formulations *HTSPxh* et *HTSPxy*. Nous avons aussi présenté des familles de contraintes valides pour le problème.

5

Résultats expérimentaux

Notre objectif principal dans cette phase expérimentale est de mesurer l'efficacité des contraintes de coupes et des inégalités valides considérées dans la méthode de coupes pour le problème du voyageur de commerce asymétrique avec contraintes de saut (HTSP). On souhaite en particulier voir si les inégalités valides connues pour le problème du voyageur de commerce asymétrique (ATSP) seront aussi efficaces pour HTSP, c'est-à-dire, savoir si elles peuvent suffir pour obtenir une solution optimale, et dans le cas contraire, si ces contraintes peuvent donner une bonne borne inférieure pour entamer la phase de branchement. (Cette phase est généralement très coûteuse).

Nous développons pour cela un algorithme de coupes et branchements pour deux types de formulations du problème à savoir la formulation $HTSPxh$ (compacte) et $HTSPxy$ (non compacte), et nous présentons les résultats numériques obtenus.

Avant de présenter les résultats numériques obtenus lors de la résolution du problème par la méthode du "Branch and Cut", nous allons donner quelques indices permettant d'intuiter ce qu'est une instance difficile du problème du voyageur de commerce.
Le nombre de villes dans une instances n'est sûrement pas la cause principale de la difficulté d'une instance. Plusieurs instances de taille raisonnable ne sont pas encore résolues alors que d'autres de plus grande taille le sont. La taille d'une instance implique un certain besoin de

place mémoire pour résoudre les programmes linéaires, mais ne donne aucun a priori ni sur la taille de l'arbre du "Branch and Cut", ni sur le temps nécessaire à la résolution du problème. Ceci s'explique principalement par les différences de topologie des instances, c'est-à-dire la distribution des villes dans l'espace. Cette distribution influe sur la difficulté à générer des contraintes violées par l'algorithmes des coupes polyédrales, ou à obtenir de bonnes contraintes de branchement.

Nous passons à présent à la description de notre implémentation, nous avons pour cela utilisé un algorithme de coupe et branchement pour résoudre $HTSP$. L'algorithme a été implémenté dans les langages C et C++, en utilisant les logiciels *ABACUS 3.0* ([1], [58], [25]) pour gérer l'arbre de branchements et *CLP* comme solveur linéaire. *ABACUS* supporte l'Open Solver Interface (*Osi*) développé par le projet *COIN-OR* (Computational Infrastructure for Operations Research) qui signifie que chaque solveur supporté par *Osi* peut être utilisé pour résoudre les relaxations.
Les expérimentations ont été effectuées sur une machine avec un processeur Pentium IV à 3 Ghz et 512 Mo de mémoire RAM avec Linux comme système d'exploitation.
Les instances qui nous ont servi de tests sont des graphes de la TSPLIB (graphes complets). Les différentes entrées des tableaux de résultats qui vont suivre sont les suivantes :

- $Problème$: le nom de l'instance,
- $I.V.C.$: Inégalités valides considérées,
- NSC : nombre de sommets considéré dans l'instance prise,
- $|S|$: nombre de nœuds soumis à la condition de saut,
- H : nombre de saut entre chaque nœud de S,
- $N.S.B$: le nombre de sous-problèmes dans l'arbre de branchement,
- Z_{IP} : valeur de la solution entière,
- $CPU(IP)$: temps CPU associé au calcul de la solution entière en secondes,
- Z_{LP} : valeur de la solution du programme linéaire relaxé,
- $CPU(LP)$: temps CPU associé au calcul de la solution du problème linéaire en secondes,
- S_{ab} : coût de la solution avant la phase de branchements,
- S_{opt} : coût de la solution optimale,
- Gap : écart relatif, en pourcentage, entre la solution optimale et la solution avant branchements, c'est-à-dire $\frac{S_{opt}-S_{ab}}{S_{opt}} * 100$.

Le temps limite de résolution a été fixé à deux heures.
Nous avons voulu vérifier l'efficacité des inégalités : *DFJ*, *ATSPcut*, *3-cycles liftée*, T_2, E_4 pour $HTSP$ et cela pour les deux formulations *HTSPxh* et *HTSPxy*. Nous avons donc résolu le problème sans et avec ces inégalités.

5.1 Formulation *HTSPxh*

L'utilisation de la formulation compacte $HTSPxh$ associée au $HTSP$ a permis d'obtenir les résultats expérimentaux suivant :

Problème	I. V. C.	Z_{LP}	CPU (LP)	Z_{IP}	CPU (IP)	N.S.B.	Gap(%)
	HTSPxh	48	1.44	56	2.60	27	0.11
	HTSPxh+DFJ	48	2.14	56	3.70	27	0.10
	HTSPxh+ATSPxycut	48	1.04	56	1.74	11	0.11
br17	*HTSPxh+*D_3	48	1.42	56	2.55	27	0.11
NSC=8	*HTSPxh+*T_2	48	1.31	56	2.52	27	0.11
H=$\|S\|$**=2**	*HTSPxh+*E_4	48	1.42	56	2.65	27	0.11
	HTSPxh+ATSPcut+D_3	48	1.00	56	1.73	11	0.11
	HTSPxh+ATSPcut+T_2	48	1.08	56	1.84	11	0.11
	HTSPxh+ATSPcut+E_4	48	1.07	56	1.80	11	0.11
	HTSPxh	133	2.42	133	4.33	31	0
	HTSPxh+DFJ	133	2.99	133	4.93	31	0
	HTSPxh+ATSPxycut	133	3.79	133	5.83	31	0
br17	*HTSPxh+*D_3	133	2.62	133	4.62	31	0
NSC=8	*HTSPxh+*T_2	133	2.40	133	4.30	31	0
$\|S\| = 3$	*HTSPxh+*E_4	133	2.34	133	4.19	31	0
H=2	*HTSPxh+ATSPcut+*D_3	133	3.95	133	6.18	31	0
	HTSPxh+ATSPcut+T_2	133	3.74	133	5.81	31	0
	HTSPxh+ATSPcut+E_4	133	3.71	133	5.84	31	0
	HTSPxh	732.167	169.93	752	192.46	99	0.04
	HTSPxh+DFJ	735.8	115.28	752	129.49	45	0.016
	HTSPxh+ATSPxycut	752	4.16	752	5.18	1	0
ftv33	*HTSPxh+*D_3	732.167	167.38	752	190.44	99	0.04
NSC=12	*HTSPxh+*T_2	732.167	176.68	752	200.54	99	0.04
$\|S\| = 3$	*HTSPxh+*E_4	732.167	167.97	752	200.82	99	0.04
H=2	*HTSPxh+ATSPcut+*D_3	752	3.94	752	4.54	1	0
	HTSPxh+ATSPcut+T_2	752	3.93	752	4.47	1	0
	HTSPxh+ATSPcut+E_4	752	3.93	752	5.51	1	0

On remarque que pour la plupart des instances, le temps de résolution est moins élevé lorsque les inégalités valides entre en jeu. Cependant, pour l'instance $br17$ avec 8 nœuds, le temps de résolution du problème passe de 2.42 secondes à 3.79 avec les inégalités $ATSPxycut$, $ATSPxycut + D_3$, $ATSPxycut + T_2$ et $ATSPxycut + E_4$ et reste pratiquement inchangé avec les autres inégalités (D_3, T_2 et E_4). De plus, dans le cas de l'instance $ftv33$ avec 12 nœuds on remarque que le gap entre la solution optimale et la solution avant branchements est plus élevé lorsque les inégalités DFJ, $ATSPxycut$ ne sont pas prise en compte, et que ces inégalités permettent bien de couper le polyèdre et ainsi se rapprocher de la solution optimale Z_{IP}. Une autre remarque peut aussi être faite concernant les inégalités $ATSPxycut$, on constate que si ces inégalités sont associées aux autres inégalités tel que D_3, T_2 et E_4, le temps de résolution est légèrement plus petit, reste à confirmer sur de plus grandes instances qui dans notre cas dépasse le temps limite fixé (deux heures) vu la performance de la machine utilisée. On peut en conclure qu'elles permettent dans certains cas d'approcher plus rapidement la solution optimale du problème.

5.2 Formulation *HTSPxy*

On représente ci-dessous le schéma général de l'algorithme de coupes et de branchements développé lors de notre implémentation, illustrons ainsi les différentes phases de programmation que ce soit, la déclaration des variables ($ABA\_VARIABLE$), déclaration des contraintes ($ABA\_CONSTRAINT$), présentation du sous-problème associé aux énumérations implicites (ABA_SUB) c'est-à-dire les nœ*uds* dans l'arbre de branchement et enfin la classe (ABA_MASTER) qui représente l'une des classes centrale du programme (elle permet de contrôler le processus d'optimisation et la sauvegarde des structures de données globales pour l'optimisation).

Le principe de l'algorithme consiste à résoudre la relaxation linéaire associée à la formulation $HTSPxy$ pour le problème du voyageur de commerce avec contraintes de saut.
On dit qu'une solution optimale $y \in I\!R^A$ de ce programme linéaire est réalisable pour $HTSP$ si y est entier et que toutes les inégalités de coupe sont satisfaites. Ceci n'est pas toujours le cas. Ici on utilisera la procédure de séparation des contraintes de coupes (voir section 3.2.4) pour déterminer d'éventuelles contraintes valides violées par la solution courante. Cet appel est nécessaire pour assurer la réalisabilité de la solution finale.

Il existe de nombreux solveurs pour les programmes linéaires qui pour la plupart procèdent par énumération implicite avec ajout de plans coupants. Notre choix c'est porté sur l'utilisation du solveur CLP dont seule la partie résolution du LP rentre en considération.Nous avons

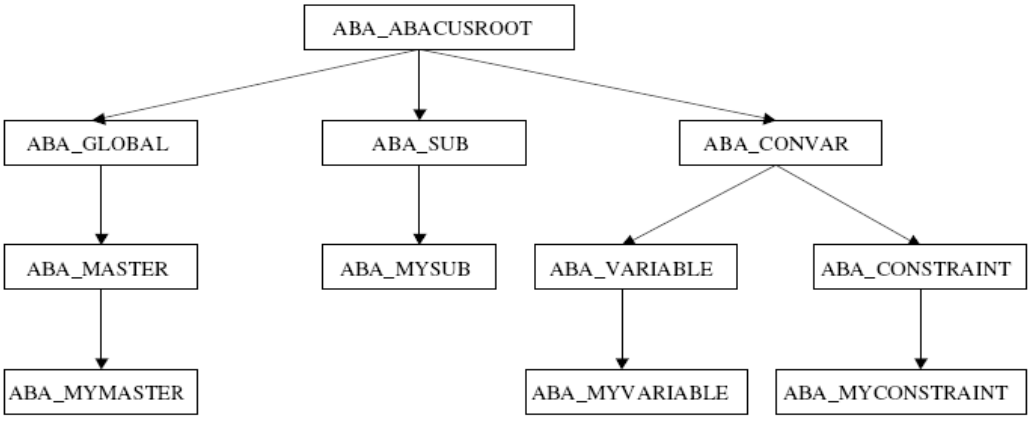


FIG. 5.1 – Classes de problèmes spécifique dans ABACUS

constaté que ce solveur avait du mal à gérer le problème sous cette formulation ($HTSPxy$). La version actuelle d'$ABACUS$ jumelée à CLP n'est pas très stable et nous avons rencontré des erreurs dues à CLP.
Une perspective intéressante serait alors d'utiliser d'autre solveur comme par exemple $CPLEX$ et voir ainsi l'efficacité des inégalités valides introduites pour $HTSPxy$.

5.3 Conclusion

D'après les résultats commentés ci-dessus, on déduit que les inégalités $ATSPxycut$ et DFJ représentes une bonne relaxation du polytope $HTSP$ et que associé aux inégalités E_4, T_2 et D_3 elles permettent d'obtenir la solution dans un temps de calcul.

6

Conclusion

Dans ce travail, nous avons étudié le problème du voyageur de commerce asymétrique avec contraintes de saut et cela d'un point de vue polyédral.

Dans un premier temps, nous avons étudié le problème du voyageur de commerce asymétrique et examiné les différentes classes d'inéglités valides associées.
Nous nous sommes ensuite intéressés au problème étudié à savoir le problème du voyageur de commerce asymétrique avec contraintes de saut ($HTSP$). Pour ce problème, nous avons donné les différentes formulations associées sous forme de programmes linéaires en nombres entiers. Nous avons aussi présenté quelques inégalités valides utilisées lors de la partie implémentation. Nous avons par la suite développé un algorithme de coupe et branchements pour ce problème. Les résultats expérimentaux ont bien mis en évidence l'utilité des inégalités valides introduites (inégalités DFJ, $ATSPcut$, D_3 (*3-cycles liftée*), T_2 et E_4).

À la suite de ce travail, certaines questions restent ouvertes et méritent d'être étudiées.
D'abord sur le plan polyédral, il serait intéressant en premier lieu de décrire des conditions pour que ces inégalités définissent des facettes, voir si d'autres inégalités valides pour le $ATSP$ autres celles citées dans ce document peuvent être adaptées au $HTSP$.
Une question qui peut se poser également est de caractériser les points extrêmes critiques du polytope $P(G, S, H)$. Cela peut donner lieu à de nouvelles facettes pour le polytope associé et

aussi améliorer l'algorithme de coupe et branchements.
Sur le plan algorithmique, nous avons remarqué que le solveur utilisé CLP n'a pas pu obtenir des résultats pour notre problème, il serait donc intéressant d'utiliser d'autre solveur comme par exemple $CPLEX$ et voir ainsi l'efficacité des inégalités valides introduites pour la deuxième formulation utilisée.

7

Annexe

7.1 Problèmes combinatoires importants

7.1.1 Le problème de flot maximum-coupe minimum

Dans cette sous section on considère un graphe $G = (V, E)$ orienté, et soient s et t deux sommets fixés de G. Soit $c : E \to \mathbb{R}^+$ une fonction dite de *capacite* sur les arcs du graphe. On distingue deux sommets s et t qui seront appelés **source** et **puits**.

Définition 7.1. *Soit $R = (V, E, c)$ un réseau, s et t les sommets source et puits. Un* **flot** *f est une application de $V \times V$ dans $\mathbb{R}$ vérifiant :*

1. $f(x, y) = -f(y, x)$, $\forall (x, y) \in E$.
2. $f(e) \leq c(e)$, $\forall e \in E$.
3. $f(v, V) = 0$, $\forall v \in V \setminus \{s, t\}$.

D'une manière concrète, on dispose d'un graphe orienté dont les arêtes sont étiquetées par des nombres. On cherche à étiqueter par des nombres inférieurs ou égaux à C en assurant la nullité de leur somme orientée en tout point n'étant pas la source ou le puits.
La valeur du flot f est alors $|f| = f(s, V)$. On va s'intéresser à la recherche d'un flot f maximisant $|f|$.

La méthode de Ford-Fulkerson

Pour décrire cette méthode, on va utiliser le graphe des écarts, les chemins améliorants et les coupes, qu'on définit d'abord.
Soit $R = (V, E, c)$ un réseau, s et t sont la source et le puits respectivement, f un flot sur le réseau. La capacité résiduelle d'un arc u est notée par $c_f = c(u) - f(u)$.

Définition 7.2. *(***Le réseau des écarts***).On appelle par* **le réseau des écarts** *le graphe* $R_f = (V, E_f, c_f)$ *où* $E_f = \{(x, y) \in V \times V,\ c_f(xy) > 0\}$.
En d'autres mots, pour un arc donné u, si $c_f(u) = 0$*, alors on remplace u par un arc étiqueté par* $c_f(u)$*, sinon on le supprime.*

Définition 7.3. *(***Chemin améliorant***).Un chemin améliorant noté* μ*, est un chemin de s à t dans* R_f*.*

Définition 7.4. *(***Capacité résiduelle***).La capacité résiduelle de* μ *est notée par* $c_f(\mu) = min\{c_f u | u \in \mu\}$.

Définition 7.5. *(***Coupe***).Une* **coupe** *dans* $R = (V, E, c)$ *est une partition de* V *en* $(S, \overline{S})$ *avec* $s \in S$ *et* $t \in \overline{S}$*. La capacité de cette coupe est* $c(S, \overline{S}) = \sum_{i \in S, j \in \overline{S}} c(i, j)$. Le théorème suivant établit le lien entre la coupe et le flot dans un réseau.

Théorème 7.1. (Ford et Fulkerson). *Soit* f *un flot dans un réseau* $R = (V, E, c)$*. Il y a équivalence entre les trois propositions suivantes :*

1. f *est un flot maximum.*
2. *Il n'existe pas de chemin améliorant.*
3. *Il existe une coupe* $(S, \overline{S})$ *telle que* $|f| = c(S, \overline{S})$.

La méthode de Ford-Fulkerson consiste à démarrer d'un flot nul, et à l'améliorer en lui ajoutant f_μ *tant qu'il existe un chemin améliorant* μ*. Elle est d'une complexité algorithmique de* $O(f*|E|)$.
Il existe un autre algorithme dit Algorithme d'Edmonds-Karp, qui consiste à choisir dans R_f *à chaque étape de la méthode de Ford-Fulkerson, un plus court chemin en nombre d'arcs. Cet algorithme se base sur un graphe non orienté, et ne calcule pas les graphes des écarts. En d'autres mots, il ne parcourt que les les arcs de capacité résiduelle strictement positive. Il est d'une complexité de* $O(nm^2)$.

L'algorithme de Gomory-Hu

L'algorithme de Gomory-Hu est aussi une autre méthode de calcul de flot maximum-coupe minimum qui opère sur un graphe en le transformant en un arbre qui va représenter les flots maximums entre chaque deux sommets du graphe G.

Cet algorithme est d'une nature récursive. Il consiste à diviser l'ensemble des sommets du graphe G en deux sous ensembles R et S, et faire de même par la suite sur ces deux derniers ensembles. Initialement, il est appliqué sur tous les sommets du graphe.
Au départ, on sélectionne d'une manière aléatoire deux sommets du graphe, et on applique par exemple l'algorithme de Ford-Fulkerson pour calculer un flot maximum-coupe minimum entre ces deux sommets. Après ce calcul, on obtient deux sous-ensembles de sommets (R et S). Chacun des deux sommets sélectionnés au départ va appartenir à un des deux ensembles (R ou S). On continue encore cette procédure en choisissant à chaque fois deux sommets des deux dernières parties formées, et on calcule de nouveau un flot maximum entre ces deux sommets. Ainsi on aura calculé $n-1$ flots maximums dans ce graphe, et le résultat est l'arbre de Gomory-Hu, qui nous donne le flot maximum (coupe minimum) entre chaque couple de sommets de G. La complexité de cet algorithme est de l'ordre de $O(n^2\sqrt{m})$.

7.1.2 Bibliothèque ABACUS

Du point de vue pratique, la partie *Branch & Cut* du problème du Voyageur de Commerce est implémentée dans ABACUS, une bibliothèque de fonctions écrites en C++ implémentant les bases d'une résolution par la méthode *Branch & Cut*. Cette bibliothèque a été initialement réalisée par Stefan Thienel [58] et en est actuellement à sa version 3.0. Une introduction à ABACUS peut être trouvée dans [60].
Le principe d'ABACUS est d'implémenter la boucle de résolution commune à tout algorithme de type *Branch & Cut* et de définir les spécifications des fonctions que l'utilisateur adapte à son propre problème. Ainsi il existe une classe définissant les variables et une classe définissant les contraintes. L'utilisateur dérive ces classes pour créer ses propres variables et ses propres contraintes. Ainsi une variable est définie par les deux extrémités de l'arête correspondante. Chaque type de contrainte est défini par une classe. La procédure de séparation est aussi définie par l'utilisateur. Ainsi ABACUS permet de définir des procédures correspondant à chaque étape de la méthode *Branch & Cut* sans avoir à programmer le déroulement de la méthode en tant que tel. Un exemple de résolution du problème de Voyageur de Commerce à l'aide d'ABACUS peut être trouvé dans [59].
ABACUS permet l'ajout de classes de contraintes de manière assez aisée. En effet il suffit de créer une classe de contraintes dérivée de la classe-mère ABACUS, dans laquelle le membre de droite est spécifié ainsi que le sens de la contrainte. Les coefficients de la contrainte sont définis par une méthode, de telle sorte qu'il n'est pas nécessaire de stocker tous les coefficients entre tous les couples de sommets (cela utilise trop de mémoire). À chaque classe de contraintes correspond une classe C++. Ainsi les contraintes de degré, d'élimination de sous-tours, de peigne, d'arbres de cliques ainsi que de nombreuses autres sont définies. La librairie ABACUS doit être couplée à un solveur de programmes linéaires pour pouvoir fonctionner. Plusieurs solveurs sont gérés

par ABACUS afin de permettre une utilisation transparente, sans avoir à connaître le mode de fonctionnement de chaque solveur.

7.1.3 Variables globales

La bibliothèque ABACUS gère un certain nombre de variables globales qui permettent de gérer le processus itératif et de branchement de la méthode *Branch & Cut*. On distingue deux catégories de variables globales : les variables constitutives de la méthode et les variables d'ajustement de calcul, non nécessaires théoriquement mais permettant un meilleur contrôle du processus de résolution.

On détaille à présent les variables constitutives de la méthode *Branch & Cut*.

- Borne supérieure globale (GUB) : contient la valeur du meilleur cycle hamiltonien trouvé jusqu'à présent
- Borne inférieure globale (GLB) : contient le minimum des valeurs de la fonction objectif obtenues sur l'ensemble des nœuds actifs de l'arbre de résolution. Dès que $GLB > GUB-1$, la valeur GUB est la solution optimale du problème.

Les principales variables permettant un contrôle du processus itératif sont les suivantes :

- Fréquence de "*pricing*" : comme le "*pricing*" nécessite l'analyse de toutes les variables non présentes dans le programme linéaire courant, il est coûteux. On réalise donc cette opération à une fréquence définie au départ et non après chaque résolution de programme linéaire. Ce processus ne s'utilise que lors d'un algorithme de génération de colonne.
- Fréquence d'amélioration de GUB : la borne supérieure du problème correspond au meilleur cycle hamiltonien trouvé jusqu'à présent. Un cycle est recherché avant de résoudre le premier programme linéaire afin d'obtenir une première borne. Cette borne est améliorée à partir de la solution courante du programme linéaire à une certaine fréquence.
- Fréquence et valeur de *"tailing off"*. Le *"tailing off"* consiste à forcer l'étape de branchement même si la séparation permet encore de trouver des contraintes. En effet il arrive que l'étape de séparation aboutisse à la génération de contraintes qui ne font que très peu progresser la valeur de la fonction objectif. Dans ce cas, un branchement est effectué. La fréquence du contrôle de progression de la valeur de la fonction objectif peut être définie par l'utilisateur de même que le pourcentage de progression minimum entre deux vérifications.

7.1.4 Solveur de programmes linéaires

Le solveur de programmes linéaires utilisé dans le code développé est CLP.

7.1.5 Méthodes de séparation

L'étape de séparation se fait à l'aide de la solution fractionnaire courante, stockée sous forme d'un tableau, et du graphe, défini dans une classe particulière. Ces deux données permettent d'appliquer les algorithmes de séparation, qui génèrent les contraintes sous un format adapté, afin de pouvoir les ajouter à l'ensemble des contraintes existantes. Une contrainte non active dans un certain nombre de programmes linéaires relaxés successifs (dont le membre de droite n'est pas atteint) est supprimée.

7.1.6 Méthodes de branchement

La méthode de branchement implémentée par défaut est basée sur la valeur des variables dans le dernier programme linéaire relaxé. L'utilisateur définit un nombre de variables k qu'il souhaite tester pour réaliser le branchement. L'algorithme implémenté dans ABACUS choisit k variables dont la valeur est proche de 0.5, puis teste la valeur de la fonction objectif des deux sous-problèmes générés par un branchement sur cette variable, en résolvant le problème par l'ajout de la contrainte de branchement. La variable e^* la plus prometteuse selon les critères définis par l'utilisateur, est alors sélectionnée pour générer deux sous-problèmes, un dans lequel elle est fixée à 0, l'autre dans lequel elle est fixée à 1.
Cette méthode de branchement a un inconvénient majeur : le problème est déséquilibré du côté $x_{e^*} = 0$, car fixer une variable à 1 fait bien plus avancer la résolution du problème que de la fixer à 0. En effet toute solution réalisable du problème est un vecteur comportant n composantes ayant une valeur de 1 et $O(n^2)$ composantes ayant une valeur à 0. C'est pourquoi d'autres méthodes de branchement ont été imaginées, comme choisir un ensemble de sommets dont le cocycle est proche d'un nombre impair ou choisir un ensemble de variables de valeurs proches de 0. ABACUS permet l'ajout de méthodes de branchement de manière très simple, grâce à la redéfinition de la méthode utilisée par défaut.

Bibliographie

[1] http ://www.informatik.uni-koeln.de/abacus/.

[2] D. Applegate, R. Bixby, V. Chvatal et W. Cook : Computational Combinatorial optimization. M. Junger and D. Naddef, editors, Springer (2001)

[3] D. Applegate, R. Bixby, V. Chvatal et W. Cook : Special session on tsp. In 15th International Symposium on Mathematical Programming. University of Michigan, USA (aug 1994)

[4] N. Ascheuer, M. Fischetti et M. Grötschel : A Polyhedral study of the asymmetric traveling salesman problem with time windows. Wiley (2000)

[5] N. Ascheuer, M. Fischetti et M. Grötschel : Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. Zib (1999)

[6] N. Ascheuer and M. Jünger et G. Reinelt : A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints. Computational Optimization & Application, 17 : 61-84 (2000)

[7] E. Balas : The asymmetric assignment problem and some new facets of the traveling salesman polytope on a directed graph. SIAM Journal on Discrete Mathematics 2, 4, pp : 425-451 (1989)

[8] E. Balas et M. Fischetti : Polyhedral theory for the asymmetric traveling salesman problem. in The Traveling Salesman Problem and its Variations, G. Gutin and A. Punnen ed.s, Kluwer, 117-168, 2002.

[9] E. Balas, M. Fischetti et W. Pulleyblank : The precedence constrained asymmetric traveling salesman polytope. Math, Prog., 68 :241-265 (1995)

[10] N. L. Boland, L. W. Clarke et G. L. Nemhauser, The asymmetric traveling salesman problem with replenishment arcs. Elsevier Science B. V, 2000.

[11] S. Borne, Etude de problèmes de sécurisation de réseaux. *Rapport de DEA*, Université Blaise Pascal, Clermont-Ferrand, 2002.

[12] P. M. Camerini, L. Fratta et F. Maffioli : On improving relaxation methods by modified gradient techniques. Mathematical Programming Study 3 : 26-34, 1974.

[13] A. Caprara, M. Fischetti, 0-1/2 Chvatal-Gomory Cuts. *Mathematical Programming A*, 74, 221-235, 1996.

[14] Carpaneto, G., M. Dell'Amico, and P. Toth, Exact Solution of Large-scale Asymmetric Traveling Salesman Problems, *ACM Transactions on Mathematical Software*, 21, 394–409, 1995.

[15] G. Carpaneto, P. Toth, Some new Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem, Management Science 26, 1980.

[16] R.D. Carr et G. Lancia : Compact vs. exponential-size LP relaxations. Operations Research Letters, 30 :57-65 (2002)

[17] S. Chopra et G. Rinaldi : The Graphical Asymmetric Traveling Salesman Polyhedron. IPCO : 129-145, 1990.

[18] V. Chvatal : Edmonds polytope and weakly hamiltonian graphs. Mathematical Programming, 5 :29-40, 1973.

[19] H. Crowder et M. Padberg : Solving largescale sym metric traveling salesman problems to optimality. Management Science 26, 495509, 1980.

[20] G. Dantzig, D. Fulkerson et S.Johnson : Solution of a large scale traveling salesman problem. Oper. Res, 2 : 393-410, 1954.

[21] M. Desrochers et G. Laporte : Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. Operations Research Letters, 10 : 27-36, 1991.

[22] E. A. Dinits : Algorithm for solution of a problem of maximum ow in a networkwith power estimation. Soviet math. Dokl. 11 : 1277-1280, 1970.

[23] J. Edmonds : Maximum matching and a polyhedron with 0,1-vertices. Journal of Research of the National Bureau of Standards (B) 69 : 9-14, 1965.

[24] J. Edmonds. Submodular function, matroids, and certain polyhedra, in Combi-natorial Structures and their Application. R. K. Guy, E.Milner, and N. Sauer,eds., New York, Gordon and Breach, pp. 69-87, 1970.

[25] M. Elf, C. Gutwenger, M. Jünger, et G. Rinaldi. Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimisation*, I. NCS 2241, pages 157 222, 2001.

[26] M. Fischetti. Clique tree inequalities define facets of the asymmetric travel ing salesman polytope. Technical report, Technical report OR/89/7(R), D niversity of Bologna, 1989.

[27] M. Fischetti. Facets of the asymmetric trling sleman plyto. *Mathmatic of Operations Research*, 16 :2-56 1991.

[28] M. Fischetti, A. Lodi, P. Toth, Exact Methods for the Asymmetric Traveling Salesman Problem, in The Traveling Salesman Problem and its Variations, G. Gutin and A. Punnen ed.s, Kluwer, 169-206, 2002.

[29] L. R. Ford and D. R. Fulkerson : Maximal ow through a network. Can. J. Math.8 399-404, 1956.

[30] M. R. Garey et D. S. Johnson, Computers and Intractability : A Guide to the Theory of NP-completeness. Freeman, San Francisco, 1979.

[31] B. Gavish et S. Graves, The travelling salesman problem and related problems. Working Paper OR-078-78, Operations Research Center, MIT, Cambridge, MA 1978.

[32] A. V. Goldberg and R. E. Tarjan, A New Approach to the Maximum-Flowproblem. Journal of the Association for Computing Machinery, Vol. 35, No. 4, pp. 921-940, October 1988.

[33] R. E. Gomory : An algorithm for integer solutions of linear programs. Recent Advances in Mathematical Programming, 269-302, 1963.

[34] L. Gouveia et P. Pesneau : On extended formulation for the precedence constrainted asymmeric traveling salesman problem. Wiley, 2006.

[35] L. Gouveia et J. Pires : The asymmetric traveling salesman problem and a reformulation of the Miller-Tucker-Zemlin constraints. European Journal of Operational Research, 112 : 134-146, 1999.

[36] L. Gouveia et J. Pires : The asymmetric traveling salesman problem : on generalizations of disaggregated Miller-Tucker-Zemlin constraints. Discrete Applied Mathematics, 112 : 129-145, 2001.

[37] M. Grötschel et O. Holland : Solution of large-scale symmetric travelling salesman problems. Mathematical Programming, 51 : 141-202, 1991.

[38] M. Grötschel and L. Lovász et A. Schrijver : The ellipsoid method and its consequences in combinatorial optimisation. Combinatorica 1 : 70-89, 1981.

[39] M. Grötschel et M. Padberg : Polyhedral theory. Lawler et al. The traveling salesman problem : a guided tour of combinatorial optimization, New York : Wiley, 1985.

[40] M. Grötschel et M. W. Padberg : On the symmetric traveling salesman problem : I and II. Mathematical Programming, 16 : 265-280 and 281-302, 1979.

[41] M. Grötschel and W. Pulleyblank. Clique tree inequalities and the symmetric traveling salesman problem. *Mathematics of Operations Research*, 11 :537-59, 1986.

[42] R. Karp et J. Steele, Probabilistic Analysis of Heuristics, in E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys (eds.), The Traveling Salesman Problem, Wiley, New York, pp. 181–205, 1990.

[43] M. Turkensteen, D. Ghosh, B. Goldengorin et G. Sierksma, Tolerance-based Branch and Bound algorithms for the ATSP, *European Journal of Operational Research* 189 : 775–788, 2007.

[44] G. Laporte, The traveling salesman problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59, 231-247, 1992.

[45] E.L. Lawer, J.K. Lenstra, A. Rinnooy-Kan et D. B. Shmoys : The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization, Wiley, New York, 1985.

[46] L. Létocart, L. Alfandari et S. Borne : Modèles linéaires et quadratiques pour le problème du voyageur de commerce asymétrique. 9éme congrés de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF2008), Clermont-Ferrand, France : pp. 295-296, Février 2008.

[47] Miller, D. and J. Pekny, Exact Solution of Large Asymmetric Traveling Salesman Problems, Science, 251, 754–761, 1991.

[48] C. Miller, A. Tucker et R. Zemlin : Integer programming formulation of traveling salesman problems. *Journal of ACM*, 7 : 326-329, 1960.

[49] T. Öncan, I. Kuban Altinel et G. Laporte : A comparative analysis of several asymmetric traveling salesman problem formulation. *Computers & Operations Research* 36 : 637-654, 2009.

[50] M. Padberg et T. Sung : An analytical comparaison of different formulations of the traveling salesman problem. Mathematical Programming 52 : 315-357, 1992.

[51] M.Padberg et G.Rinaldi : A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM Review, 33(1) : 60-100, 1991.

[52] M.Padberg et G.Rinaldi : Optimization of a 532-city symmetric traveling salesman problem by branch and bound and cut. Operations Research Letters 6 : 1-7, 1987.

[53] M.Padberg et G.Rinaldi : Facet identification for the symmetric traveling salesman polytope. Mathematical Programming, 47 : 219-257, 1990.

[54] M.Padberg et M.R. Rao : Odd minimum cut-sets and b-matchings. Math. of OR, 7 : 67-80, 1982.

[55] G.Paletta et C. Triki : Solving the asymmetric traveling salesman problem with periodic constraints. Wiley, Inc 2004.

[56] S. C. Sarin , H. D. Sherali, et A. Bhootra : New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. Operations Research Letters 33 : 62-70, 2005.

[57] H.D. Sherali et P.J. Driscoll : On tightening the relaxations of Miller-Tucker-Zemlin formulations for asymmetric traveling salesman problems. Operations Research, 4 : 134-146, 2002.

[58] Stefan Thienel. Stefan Thienel. ABACUS - *A Branch-And-CUt System*. PhD thesis, Universität zu Köln, 1995

[59] Stefan Thienel. A Simple TSP-Solver : An ABACUS Tutorial, 1996.

[60] Stefan Thienel. Introduction to abacus - a branch-and-cut system. Technical Report 97.263, Universität zu Köln, 1997.