Introduction de contraintes structurelles pour la résolution du problème du voyageur de commerce

Nicolas Isoart, Jean-Charles Régin

Introduction de contraintes structurelles pour la résolution du problème du voyageur de commerce

Nicolas Isoart* **Jean-Charles Régin**
Université Côte d'Azur, CNRS, I3S, France
nicolas.isoart@gmail.com jcregin@gmail.com

Résumé

Plusieurs modèles basés sur la programmation par contraintes ont été proposés pour résoudre le problème du voyageur de commerce (TSP). Les plus efficaces, telle que la weighted circuit constraint (WCC), s'appuient principalement sur la relaxation lagrangienne du TSP, basée sur la recherche d'arbres recouvrants ou plus précisément de "one-trees". Le défaut de cette méthode est qu'elle n'inclut pas assez de contraintes structurelles et se base presque exclusivement sur les coûts des arêtes. L'objectif de cet article est de corriger ce défaut. Aussi, nous cherchons des motifs empêchant l'existence d'un cycle hamiltonien dans un graphe ou, à l'inverse, des motifs imposant que certaines arêtes soient dans l'ensemble de solutions du TSP. Nous proposons un propagateur basé sur la recherche de k-cutsets pour la contrainte de cycle hamiltonien. Sa combinaison avec la contrainte WCC permet d'obtenir, pour la résolution du TSP, des gains d'un ordre de magnitude pour le nombre de backtracks ainsi qu'une forte réduction du temps de calcul.

Abstract

Several models based on constraint programming have been proposed to solve the travelling salesman problem (TSP). The most efficient, such as weighted circuit constraint (WCC), mainly rely on the Lagrangian relaxation of TSP, based on the search for trees covering or more precisely "one-trees". The weakness of this method is that it does not include enough structural constraints and is based almost exclusively on edge costs. The purpose of this article is to correct this defect. Also, we are looking for reasons preventing the existence of a Hamiltonian cycle in a graph or, conversely, reasons requiring that certain edges be in the TSP solution set. We propose a propagator based on the research of k-cutsets for the Hamiltonian cycle constraint. Its combination with the WCC constraint allows us to obtain, for the resolution of the TSP, gains of an order of magnitude for the number of backtracks as well as a strong reduction of the computation time.

*Papier doctorant : Nicolas Isoart est auteur principal.

1 Introduction

Le problème du voyageur de commerce ou TSP (travelling salesman problem) est un problème NP-Difficile ou NP-Complet selon les variantes. Il a de nombreuses applications et a été motivé par des problèmes concrets, tels que le parcours des bus scolaires, la logistique, le routage, etc. A peu près tous les types de méthodes de résolution (MIP, SAT, CP, algorithme évolutionnaire, etc.) ont été testés pour le résoudre. Lorsque le graphe est Euclidien, le programme le plus efficace est le logiciel ad hoc Concorde [1]. Malheureusement, il ne peut pas prendre en compte des contraintes additionnelles. Or, en pratique, elles sont bien souvent nécessaires pour un grand nombre de problèmes, comme le Pickup & Delivery, des problèmes de transport à la demande (Dial and Ride), les vendanges et moissonnages automatiques, etc. La résolution du TSP est difficile puisqu'il s'agit de trouver un unique cycle passant par tous les sommets d'un graphe, tel que la somme des coûts des arêtes qui le composent soit minimale. On peut assez facilement modéliser le fait que chaque sommet appartienne à un cycle. En effet, il suffit que chaque sommet ait au moins deux voisins distincts, autrement dit, chaque sommet doit être l'extrémité d'au moins deux arêtes. On obtient ce résultat en modélisant notre problème comme un problème d'affectation, qui se résout en temps polynomial. Malheureusement, ce modèle n'est pas suffisant pour obtenir un seul et unique cycle dans le graphe. En effet, l'affectation correspond à un recouvrement des sommets par un ensemble de cycles disjoints. Pour ce modèle, on obtient des solutions où chaque sommet appartient à un cycle, mais pas à un seul et unique cycle. Habituellement, pour parvenir

à ce résultat on impose que le sous-graphe engendré par les arêtes sélectionnées soit connexe. C'est cette combinaison des deux aspects qui rend le problème aussi difficile. Contrairement à l'approche précédente, on peut construire un modèle basé sur la notion de sous-graphe connexe. C'est exactement l'idée de Held et Karp [6, 7] qui ont représenté cette notion par un 1-tree, qui est un arbre recouvrant particulier. La notion de cycle est alors introduite en imposant que chaque nœud ait un degré 2 dans le 1-tree. Puis, ils ont utilisé une relaxation lagrangienne en relâchant la contrainte de degré. Plus précisément, ils résolvent une succession de problèmes du minimum spanning tree (polynomial) en ajoutant la contrainte de degré à la fonction objectif tel que si un nœud a un degré > 2, on diminue le coût des arêtes voisines, et si le degré est < 2 on augmente le coût des arêtes voisines pour obtenir une convergence vers la solution optimale.

Comme mentionné dans [3], la relaxation Lagrangienne proposée par Held et Karp semble être la meilleure : "The best approach regarding the number of instances solved and quality of the bound is the Held and Karp's filtering" [3]. D'ailleurs, la *weighted circuit constraint* (WCC) est essentiellement basée sur cette relaxation.

Nous proposons dans ce papier d'améliorer la WCC en lui ajoutant des méthodes de résolutions de cycles hamiltoniens (=TSP sans les coûts). Pour cela, nous nous inspirons des travaux de Cohen et Coudert sur la structure des cycles hamiltoniens effectués pour le FHCP Challenge [5]. La figure 1 présente un exemple où la structure du graphe est importante pour la recherche de cycle hamiltonien. Il n'existe pas de cycle hamiltonien dans ce graphe, car il est impossible de trouver un cycle visitant tous les sommets ne passant qu'une seule fois par le nœud C. On dit d'un tel graphe qu'il est 1-connexe : il existe un sommet dans le graphe tel que sa suppression le déconnecte. On peut donc définir une nouvelle contrainte structurelle : si un graphe est 1-connexe, alors il ne contient pas de cycle hamiltonien. Dans la pratique, nous n'allons que très rarement avoir un graphe 1-connexe, alors nous proposons d'étendre l'idée de cette contrainte aux arêtes et d'étudier les graphes k-arete-connexe pour $k > 1$. Nous étudierons particulièrement les valeurs $k = 2$ et $k = 3$.

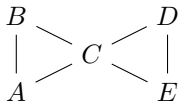

FIGURE 1 – Graphe papillon.

Cet article est organisé de la manière suivante : tout d'abord, on rappelle certains concepts de la théorie des graphes. Ensuite, nous définissons formellement les contraintes structurelles utilisées dans notre méthode de résolution du TSP. Puis, nous définissons une nouvelle structure de données appelée cycled spanning tree, elle est utilisée pour définir un nouvel algorithme visant à exploiter les contraintes structurelles. La dernière partie vérifie expérimentalement l'intérêt de notre méthode. Pour finir, nous concluons.

2 Préliminaires

2.1 Définitions

Les définitions traitant la théorie des graphes sont issues de [10].

Un **graphe orienté** $G = (X, U)$est composé d'un **ensemble de sommets** X et d'un **ensemble d'arcs** U, où chaque arc (u, v) est une paire ordonnée de sommets distincts. On notera $X(G)$ l'ensemble des sommets de G et $U(G)$ l'ensemble des arcs de G. Le **coût** d'un arc est la valeur associée à l'arc. Un **graphe non orienté** est un graphe orienté tel que pour chaque arc $(u, v) \in U$, $(u, v) = (v, u)$. Si $G_1 = (X_1, U_1)$ et $G_2 = (X_2, U_2)$ sont des graphes, G_1 est un **sous-graphe** de G_2 si $V_1 \subseteq V_2$ et $U_1 \subseteq U_2$. Un **chemin** allant du nœud v_1 au nœud v_k dans G est une liste de nœuds $[v_1, ..., v_k]$ telle que (v_i, v_{i+1}) est une arête pour $i \in [1..k-1]$. Le chemin est **élémentaire** si tous ses nœuds sont distincts. Un chemin est un **cycle** si $k > 1$ et $v_1 = v_k$. Un cycle est **hamiltonien** si $[v_1, ..., v_{k-1}]$ est un chemin élémentaire et contient tous les sommets de U. La **longueur** d'un chemin p, noté $length(p)$, est la somme des coûts des arêtes contenues dans p. Pour un graphe G, une solution au **problème du voyageur de commerce** ou **travelling salesman problem (TSP)** dans G est un cycle hamiltonien $HC \in G$ minimisant $length(HC)$. Un graphe non orienté G est **connecté ou connexe** s'il existe un chemin entre chaque paire de sommets, sinon il est **déconnecté**. Les sous-graphes maximaux connectés de G sont ses **composantes connexes**. Un **arbre** est un graphe connexe et sans cycle. Un arbre $T = (X', U')$ est un **arbre couvrant** de $G = (X, U)$ si $X' = X$ et $U' \subseteq U$. Les arêtes de U' sont les **arêtes de l'arbre** T et les arêtes de $U - U'$ sont les **arêtes hors de l'arbre** T. Un **isthme** est une arête telle que sa suppression augmente le nombre de composantes connexes. Une partition des sommets de $G = (X, U)$ telle que $S \subseteq X$ et $T = X - S$ est une **coupe ou cut**. L'ensemble des arêtes $(u, v) \in U$ ayant $u \in S$ et $v \in T$ est la **cutset** de la coupe (S, T). Une **k-cutset** est une cutset de taille k.

2.2 HCWME : Le problème du cycle hamiltonien avec des arêtes obligatoires

Les algorithmes de résolution de TSP tendent à :

• Éliminer des arêtes ne pouvant pas appartenir à la solution optimale.

• Définir des arêtes appartenant à la solution optimale, appelées arêtes obligatoires.

Du fait que chacune des solutions du TSP soit un cycle hamiltonien (HC), l'ensemble des solutions du TSP est un sous-ensemble des solutions du problème du cycle hamiltonien (HCP).

Propriété 1. *Soit $G = (X, U)$. Si $a \in U$ appartient à toutes les solutions de HCP(G), alors a appartient nécessairement à toutes les solutions de TSP(G).*

Propriété 2. *Soit $G = (X, U)$. Si HCP(G) n'a pas de solution, alors TSP(G) n'a pas de solution.*

Afin d'utiliser les propriétés 1 et 2, nous formulons le problème Hamiltonian Cycle With Mandatory Edges (HCWME) tel que :
DONNÉES : Un graphe $G = (X, U)$ et un ensemble d'arêtes obligatoires $M \subseteq U$.
PROBLÈME : Existe-il un cycle hamiltonien dans G passant par toutes les arêtes de M ?

Ce problème nous permet d'exploiter le fait qu'un solveur de contraintes définit des arêtes obligatoires. La propriété 1 permet de définir de nouvelles arêtes obligatoires à partir d'existantes. Aussi, la propriété 2 permet de détecter si étant donné un graphe, il existe une solution du TSP dans celui-ci.

2.3 Complexité du problème HCWME

Soit $G = (X, U)$. Le problème du $HCWME(G, M)$ se réduit au $HCP(G)$. Si on résout $HCWME(G, M)$ avec $M = \emptyset$ alors on résout $(HC(G))$. On vérifie en temps polynomial si une solution donnée $G' = (X', U')$ résout le problème $HCWME(G, M)$. Pour cela, il faut que X' soit égal à X et que $M \subseteq U'$: vérification polynomiale. Puis, il faut vérifier que G' forme un cycle hamiltonien : il doit être connexe et chaque sommet doit avoir exactement 2 voisins, vérification polynomiale [10]. Comme le problème du cycle hamiltonien est un problème NP-Complet, le problème $HCWME(G, M)$ est NP-Complet.

Sans perte de généralité, on note $G = (X, U)$, $n = |X|$, $m = |U|$, $M \subseteq U$ l'ensemble des arêtes obligatoires et supposons G connexe pour la suite.

3 Contraintes structurelles

3.1 Présentation de contraintes structurelles

Lorsque ce n'est pas spécifié, une k-cutset est maximale, c'est-à-dire qu'il n'existe pas de sous-ensemble de la k-cutset déconnectant le graphe.

Proposition 1. *Soit une k-cutset dans un graphe, alors tout cycle hamiltonien emprunte un nombre pair et strictement positif d'arêtes de la k-cutset.*

Démonstration. Soit $G = (X, U)$ tel que $S \subset X$ et $T = \{X - S\}$. Pour trouver un cycle hamiltonien dans G il est nécessaire que S et T soient connectés, autrement dit qu'ils aient une cutset de taille strictement positive. Du fait que l'on cherche un cycle, si on emprunte un chemin allant de S à T alors on doit emprunter un chemin disjoint allant de T à S, ce qui donne un nombre pair de chemins. □

Soient $G = (X, U)$, M ⊆ U et $\mathcal{P} = HCMWE(G, M)$. D'après la proposition 1, on définit les propriétés 3, 4, 5, 6 et 7.

Propriété 3. *S'il existe une 2-cutset dans G, alors les deux arêtes notées a_1, a_2 de la cutset deviennent obligatoires : $M \leftarrow M + \{a_1, a_2\}$.*

Propriété 4. *S'il existe dans G une k-cutset avec k impair contenant $k - 1$ arêtes obligatoires, alors l'arête non obligatoire notée a est supprimée car elle ne peut pas appartenir à un cycle hamiltonien : $E \leftarrow E - \{a\}$.*

Propriété 5. *S'il existe dans G une k-cutset avec k pair contenant $k - 1$ arêtes obligatoires, alors l'arête non obligatoire notée a le devient : $M \leftarrow M + \{a\}$.*

Propriété 6. *S'il existe dans G une 1-cutset, alors $\mathcal{P}$ n'a pas de solution.*

Propriété 7. *S'il existe dans G une k-cutset avec k impair contenant k arêtes obligatoires, alors $\mathcal{P}$ n'a pas de solution.*

Définition 1. *On dit que deux problèmes $\mathcal{P}$ et $\mathcal{P}'$ sont équivalents si leurs ensembles de solutions sont en bijection. On note alors $\mathcal{P}' = \mathcal{P}$.*

Corollaire 1. *Soit $\mathcal{P}' = \mathcal{P}$. Si une ou plusieurs des propriétés 3, 4, 5, 6 ou 7 sont appliquées sur $\mathcal{P}$, alors $\mathcal{P}$ et $\mathcal{P}'$ restent équivalents.*

Démonstration. Immédiat par la proposition 1. □

On note $\boldsymbol{a}^* \in U$ une arête obligatoire.

3.2 Exemple

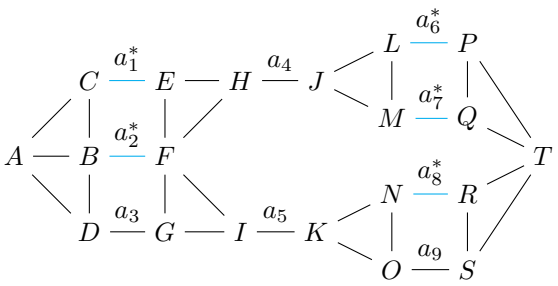

FIGURE 2 – Graphe G_1.

D'après les propriétés 3, 4 et 5, on peut supprimer certaines arêtes de la figure 2 et en rendre obligatoires :

• $\{a_4, a_5\}$ forme une 2-cutset : si on veut relier la partie gauche de H I à la partie droite de J K par un cycle il faut obligatoirement prendre (H,J) et (I,K) donc a_4 et a_5 deviennent obligatoires.

• $\{a_1^*, a_2^*, a_3\}$ forme une 3-cutset avec $\{a_1^*, a_2^*\}$ obligatoires : c'est une cutset de taille impaire avec un nombre pair d'arêtes obligatoires. Alors on peut supprimer a_3, car en la choisissant la cutset deviendrait composée uniquement d'arêtes obligatoires et de taille impaire.

• $\{a_6^*, a_7^*, a_8^*, a_9\}$ forme une 4-cutset avec $\{a_6^*, a_7^*, a_8^*\}$ obligatoires : c'est une cutset de taille paire avec 3 arêtes obligatoires. Si on ne rend pas a_9 obligatoire, alors la cutset n'aura jamais un nombre pair d'arêtes obligatoires, elle est donc nécessaire.

La figure 3 représente l'application des propriétés 3, 4 et 5 sur G_1.

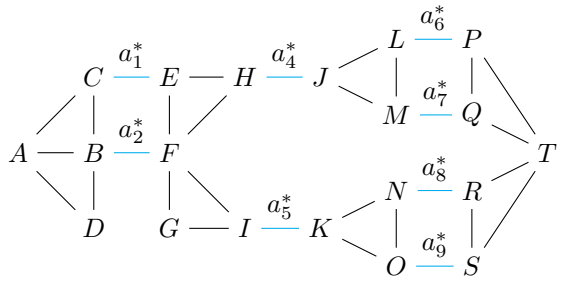

FIGURE 3 – Application des propriétés 3, 4 et 5 sur G_1.

D'après la propriété 7 il ne peut pas exister de cycle hamiltonien dans la figure 4 :

• $\{a_4\}$ est une 1-cutset : il ne peut pas exister de cycle hamiltonien reliant $\{I, J, K\}$ au reste du graphe.

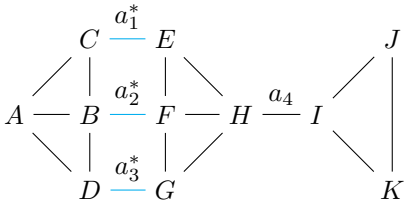

FIGURE 4 – Graphe G_2.

• $\{a_1^*, a_2^*, a_3^*\}$ forme une 3-cutset où les trois arêtes sont obligatoires : on veut un cycle hamiltonien passant par ces trois arêtes mais il ne peut pas en exister.

Nous allons maintenant montrer comment appliquer les propriétés 3, 4, 5, 6 ou 7. Etant donné que ces propriétés sont basées sur la taille des cutsets, on peut légitimement se demander combien de cutsets un graphe peut posséder.

Propriété 8. *Le nombre de cutset d'un graphe de n sommets est* 2^n.

Démonstration. Soit $G = (X, U)$. N'importe quelle partie $S \subseteq X$ forme une coupe (S, T) où $T = X - S$ avec une cutset. La taille de l'ensemble des parties de $S \subseteq X$ est 2^n, il y a donc 2^n cutsets. □

Dans le cas du graphe non orienté, on remarque qu'une coupe (S, T) a la même cutset qu'une coupe (T, S) : on peut aller par les mêmes chemins de S à T et de T à S. Le nombre de cutsets différentes est alors $2^n/2 = 2^{n-1}$.

Dans le but d'intégrer cette idée à un solveur de contraintes, on va donc ne chercher que certaines k-cutsets et appliquer les propriétés 3, 4, 5, 6 ou 7. Si on est dans le cas d'un graphe très dense, alors on a peu de chance de satisfaire les propriétés 3, 4, 5, 6 ou 7 pour une petite valeur de k. Il n'apparaît non plus pas très raisonnable d'appliquer ces propriétés avec une valeur de k élevée pour au moins deux raisons :

• La complexité des algorithmes de k-cutset augmente avec k parce que ce sont des algorithmes d'énumérations [12].

• La relation entre la taille de la cutset et le nombre d'arêtes obligatoires est forte. Plus k augmente et moins on a de chance de satisfaire une des propriétés 3, 4, 5, 6 ou 7. En conséquence, nous proposons d'étudier $k = 1, 2$ et 3 avec les comportements suivants :

• Les 1-cutsets : provoquer une erreur.

• Les 2-cutsets : rendre obligatoire les arêtes de la 2-cutset.

• Les 3-cutsets : considérer uniquement les 3-cutsets avec au moins 2 arêtes obligatoires. Si elle contient une

arête non obligatoire, alors il faut la supprimer, sinon une erreur est provoquée.

4 L'algorithme

Nous devons être capables de trouver les k-cutsets avec $k = 1, 2, 3$ et deux arêtes obligatoires pour $k = 3$. Si on décompose le problème, trouver les 1-cutset, qui sont en fait des isthmes, peut être fait avec l'algorithme de Tarjan [9] en $O(m + n)$; trouver des 2-cutset peut être fait avec l'algorithme de Tsin [11] en $O(m + n)$. L'avantage de l'algorithme de Tsin est qu'il permet aussi de trouver les isthmes, on peut alors gérer $k = 1, 2$ uniquement avec ce dernier. Il nous faut maintenant gérer le cas de $k = 3$ avec au moins deux arêtes obligatoires.

D'après la définition de coupe, si on supprime une arête d'une k-cutset, alors elle devient une $(k-1)$-cutset. On peut alors proposer un premier algorithme simple :

Pour chaque arête obligatoire $a^* \in M$, on cherche les 2-cutsets de $G - \{a^*\}$. De cette façon, chacune des 2-cutset trouvées forme une 3-cutset avec au moins une arête obligatoire. Il suffit alors de conserver uniquement les 3-cutsets avec 2 arêtes obligatoires.

On peut réduire le nombre d'arêtes obligatoires considérées. Pour cela, nous allons construire une structure particulière, nommée CST.

4.1 CST : Cycled Spanning Tree

Soit $G = (X, U)$. Un CST est un sous-graphe connexe de G tel que chacune de ses arêtes appartient à un cycle dans le CST.

Une manière de construire un CST est de calculer un arbre couvrant T, puis d'ajouter certaines arêtes à T jusqu'à ce que toutes les arêtes, celles de T et celles hors de T, appartiennent à un cycle du CST. Ainsi, on introduit l'idée de Cycled Spanning Tree.

Propriété 9. *Soit T un arbre couvrant de G. Toute arête qui n'est pas dans T appartient à un cycle composé de cette arête et uniquement d'arêtes de T.*

Démonstration. Par définition de l'arbre couvrant, $\forall i, j \in X$, il existe un chemin élémentaire $p = [i, ..., j]$ dans T. Choisissons une arête dans $G - T$, notée (i, j). Ajouter (i, j) à p donne $p = [i, ..., j, i]$, par définition p est maintenant un cycle où (i, j) est la seule arête $\notin T$. □

Définition 2. *Soit $G' = (X', U')$ un sous-graphe de G. On dit que G'* **supporte** *les arêtes de $G" = (X, U - U')$ si chacune des arêtes de $G"$ forment un cycle avec uniquement des arêtes de G'.*

Cela signifie qu'un arbre couvrant supporte toutes les arêtes qui ne sont pas dans T. Mais qu'en est-il des arêtes de T ?

Dans ce cas il nous faut d'autres arêtes pour les supporter. On va donc ajouter des arêtes à notre arbre couvrant. On prend une arête $a_T \in T$ puis on cherche une autre arête $\notin T$ permettant d'introduire dans T un cycle contenant a_T. Puis on contracte toutes les arêtes du cycle dans T contenant a_T. On répète jusqu'à avoir contracté toutes les arêtes de T. À la fin on aura ajouté au plus $n - 1$ arêtes, puisqu'on peut considérer individuellement chaque arête de T et que pour chaque arête, l'ajout d'une arête suffit. La nouvelle structure obtenue est le CST, elle contient au plus $2n - 2$ arêtes.

Sans perte de généralité, on admet G connexe et sans isthme, il existe alors nécessairement un CST dans G.

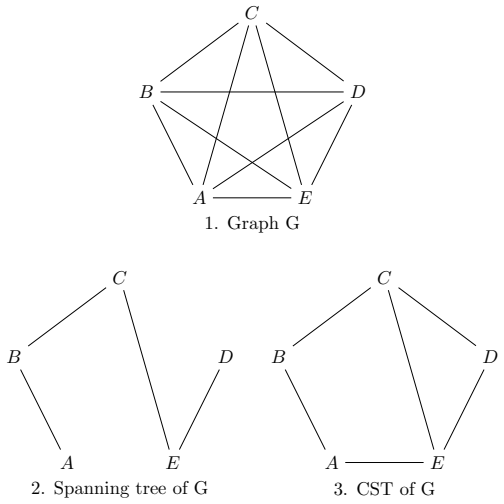

FIGURE 5 – Exemple de construction d'un CST.

Corollaire 2. *Soit $k > 1$. S'il existe une k-cutset dans G, alors au moins deux arêtes de la cutset sont dans le CST.*

Démonstration. Par construction, le CST est connexe et couvre le graphe par des cycles. Comme un cycle à une coupe ≥ 2, chaque k-cutset dans G avec $k > 1$ ont une coupe de taille ≥ 2. □

Corollaire 3. *S'il existe une 3-cutset contenant au moins deux arêtes obligatoires, alors au moins une arête obligatoire appartient au CST.*

Démonstration. Immédiat par le corollaire 2. □

Via le corollaire 3, on peut améliorer l'algorithme simple en réduisant le nombre d'arêtes obligatoires considérées. L'algorithme devient : pour chaque arête

obligatoire a^* du CST, appelée **arête d'identification**, chercher les 2-cutsets appartenant à $G - \{a^*\}$. Pour chaque 3-cutset trouvée, on obtient soit une 3-cutset avec trois arêtes obligatoires, soit une 3-cutset avec deux arêtes obligatoires ou alors une 3-cutset avec une arête obligatoire. Puis, il suffit de leur appliquer les propriétés 3, 4, 5, 6 et 7.

Puisque les arêtes obligatoires hors du CST ne sont pas considérées comme arêtes d'identifications et qu'on choisit à la construction les arêtes présentes dans le CST, on a intérêt à minimiser son nombre d'arêtes obligatoires.

4.2 Amélioration supplémentaire

L'algorithme proposé dépend fortement du nombre d'arêtes d'identifications. D'après le corollaire 2, si deux arêtes appartiennent à la même 2-cutset et qu'elles sont obligatoires, alors ce sont des arêtes d'identifications. Cependant, lorsqu'on a cherché les 3-cutsets avec une arête d'identification, il n'est pas nécessaire de répéter la recherche pour toutes les arêtes formant une 2-cutset avec celle-ci. Plus précisément, le problème de la recherche des 3-cutsets avec a^* comme arête d'identification a le même ensemble de solutions que le problème de la recherche des 3-cutsets avec chacune des arêtes formant une 2-cutset avec a^*. La figure 6 l'illustre bien puisque la 2-cutset est un chemin.

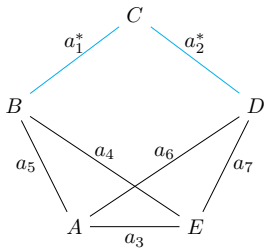

FIGURE 6 – $\{a_1^*,a_2^*\}$ est une 2-cutset. $\{a_1^*,a_4,a_5\}$ et $\{a_1^*,a_6,a_7\}$ sont les 3-cutsets incluant a_1^*. $\{a_2^*,a_4,a_5\}$ et $\{a_2^*,a_6,a_7\}$ sont les 3-cutsets incluant a_2^*. .

Afin de généraliser, on définit la propriété 10.

Propriété 10. *Soient S_1 une k-cutset et S_2 une 2-cutset telles que $k > 1$ et $S_2 \not\subseteq S_1$, si $\exists a \in S_1$ tel que $a \in S_2$ alors $(S_1 \cup S_2) - \{a\}$ forme une k-cutset.*

Démonstration. Soient $S_2 = \{a_1, a_2\}$ une 2-cutset et $a_1 \in S_1$. Supprimer la cutset S_1 du graphe le déconnecte en deux composantes connexes. Par la définition de coupe, $S_2 - \{a_1\} = \{a_2\}$ est un isthme et déconnecter a_2 augmente le nombre de composantes connexe à trois. Si on reconnecte a_1, G est déconnecté en deux composantes connexes, sa cutset est $(S_1 - \{a_1\}) \cup \{a_2\} = (S_1 - \{a_1\}) \cup (S_2 - \{a_1\}) = (S_1 \cup S_2) - \{a_1\}$. Comme S_1 est une k-cutset, il n'existe pas de sous-ensemble de celle-ci déconnectant le graphe autre que la k-cutset elle même. Si $(S_1 \cup S_2) - \{a_1\}$ déconnecte le graphe alors c'est une k-cutset car on supprime et on ajoute une arête dans un ensemble de cardinalité initial k. □

Soient S_1 une 3-cutset, $S_2 = \{a_1, a_2\}$ et S_3 deux 2-cutsets distinctes. La propriété 10 permet d'améliorer notre algorithme car le nombre d'arêtes d'identifications est réduit. On a :

- Si $a_1 \in S_1$, alors $(S_1 - \{a_1\}) \cup \{a_2\}$ est une 3-cutset.
- Si $a_1 \in S_3$, alors $(S_3 - \{a_1\}) \cup \{a_2\}$ est une 2-cutset.

Ainsi, on considère comme arêtes d'identification, les arêtes obligatoires du CST n'appartenant à aucune 2-cutset et pour l'ensemble des 2-cutsets de G, son sous-ensemble qui maximise sa cardinalité tel qu'il n'existe pas de combinaison de celui-ci formant une 2-cutset.

Pour ne pas avoir d'incohérence, il faut chercher toutes les 2-cutsets avant d'effectuer la recherche des 3-cutsets. Si ce n'est pas fait, il se peut qu'il existe une 2-cutset contenant au moins une arête non obligatoire. Cela peut avoir pour effet que lors de la recherche des 3-cutsets une arête soit désignée comme supprimable alors qu'elle est nécessaire à l'existence d'un cycle hamiltonien.

Aussi, la suppression d'une arête dans une 3-cutset peut créer une 2-cutset : il faut attendre la fin de la recherche de toutes les 3-cutsets pour rendre effectives les suppressions. Pour avoir une suppression instantanée, il faudrait effectuer une recherche des 2-cutsets à chaque suppression d'arêtes lors de la recherche des 3-cutsets.

Puis, lorsque l'on a peu d'arêtes obligatoires dans le graphe, il y a de grandes chances qu'on ne puisse pas les prendre dans le CST. En revanche, lorsque ce nombre grandit, elles deviennent nécessaires à l'existence d'un CST et forment des chemins, donc des 2-cutsets ! Alors, lorsque $|M|$ tend vers n, le nombre d'arêtes d'identifications tend vers 1. Du fait que notre complexité dépend fortement de ce nombre, l'impact est fort sur le temps de résolution.

Enfin, le CST a un autre avantage : il est incrémental. En effet, tant qu'aucune arête du CST n'est supprimée, toutes les arêtes hors du CST appartiennent à un cycle composé d'arêtes du CST, il n'est alors pas nécessaire de le reconstruire.

4.3 Implémentation

Nous proposons une implémentation du filtrage k-cutset dans l'algorithme 1. La fonction principale est FILTRAGE(G). Les fonctions CHERCHER2CUTSET(G) et CHERCHER3CUTSET(G) appellent la fonction CHERCHERCUTPAIRS(G,isthmAction,cutpairAction), elle fait

une recherche des 2-cutsets comme proposé dans [11] avec une complexité en $O(n+m)$. Ses arguments sont respectivement un graphe, une fonction spécifiant ce qui doit être fait lorsqu'un isthme est trouvé et une fonction spécifiant ce qui doit être fait lorsqu'une 2-cutset est trouvée. Lors de l'exécution de FILTRAGE(G), CHERCHER2CUTSET(G) trouve donc toutes les 2-cutsets. S'il n'existe aucun isthme, la fonction MERGECUTPAIRS(set) remplit le tableau *id* afin que chaque arête appartenant à la même 2-cutset ait la même valeur dans *id*. Si deux 2-cutsets ont une arête en commun, alors chacune de leurs arêtes ont la même valeur dans *id*. On peut alors exploiter l'amélioration proposée en 4.2 qui réduit la complexité en $O(|M|)$. Un CST est ensuite calculé en $O(n)$ avec la méthode proposé en 4.1, l'ensemble S représente les arêtes obligatoire de ce dernier. Enfin, pour chaque arête a^* de S on considère celles n'appartenant soit à aucune cutset, soit à une cutset dont la recherche de $id[a^*]$ n'a pas été effectuée. Il faut ensuite trouver les 3-cutsets contenant a^*. Pour cela, on considère G' le graphe duquel l'arête a^* est retiré et on cherche les 2-cutsets dans G' avec la fonction CHERCHER3CUTSET(G'). Lors des itérations, si une 3-cutset avec trois arêtes obligatoires est trouvée, alors l'exécution est stoppée (la variable fail est mise à true), sinon, les cutsets avec l'identifiant $id[a^*]$ sont marquées comme traités.

5 Expérimentations

Les algorithmes ont été implémentés en Java 11 dans un solveur de programmation par contraintes développé localement, appelé Talos. Les expérimentations ont été effectuées sur une machine Windows 10 avec un processeur Intel Core i7-7820HQ CPU @ 2.90GHz et 32Go de RAM. Les instances de références sont issues de la TSPLib [8], une librairie de graphes de références pour le TSP. Toutes les instances considérées sont des graphes symétriques.

Nous présentons les résultats sous la forme de tables. Chacune d'entre elles rapporte le temps de résolution en millisecondes. S'il est strictement supérieur à 30 minutes il est spécifié *t.o.* Le nombre de backtracks est noté #bk. Elles comportent une colonne exprimant le ratio du temps de résolution et du nombre de backtracks.

Afin de vérifier l'intérêt de l'ajout de contraintes structurelles à la WCC, nous allons dans un premier temps vérifier que le filtrage k-cutset est efficace en utilisant la stratégie statique "maxCost", qui sélectionne les arêtes selon les coûts décroissants.

La table 1 représente les performances de l'ajout du filtrage k-cutset à la WCC. Choisir une stratégie statique telle que maxCost permet de bien comparer les performances des filtrages en évitant une perturbation due à la stratégie. Ces résultats montrent qu'utiliser un filtrage structurel est très intéressant. Par exemple, le temps de résolution de kroE100 a été réduit d'un

Algorithm 1: k-CUTSET($G=(X,U)$,M)

```
global fail
FILTRAGE(G, M) :
  fail ← false
  set ← ∅
  ∀e ∈ U : id[e] ← nil
  CHERCHER2CUTSET(G,M,set)
  if fail then return ;
  CST ← CALCULERCST()
  S ← CST.ARETESOBLIGATOIRES()
  MERGECUTPAIRS(S,set,id)
  for each a* ∈ S do
    if id[a*] = nil or visited[id[a*]] = 0 then
      G' ← (X, U − {a*})
      CHERCHER3CUTSET(G', M)
      if fail then return ;
      if id[a*] ≥ 0 then visited[id[a*]] ← 1;

CHERCHER2CUTSET(G,M,SET) :
  define isthmCutpairs(){ fail ← true}
  define cutpairsFound(M, a1, a2){
      if a1 ∉ M then M ← M ∪ {a1};
      if a2 ∉ M then M ← M ∪ {a2};
      set ← set ∪{a1, a2}
    }
  CHERCHERCUTPAIRS(G,isthmCutpairs,cutpairsFound)

CHERCHER3CUTSET(G,M) :
  define isthm3cutset(){ return }
  define 3cutsetFound(M, a1, a2){
      if a1 ∈ M and a2 ∈ M then fail ← true;
      else if a1 ∈ M then U ← U − {a2};
      else if a2 ∈ M then U ← U − {a1};
    }
  CHERCHERCUTPAIRS(G,isthm3cutset,3cutsetFound)

MERGECUTPAIRS(S,set,id) :
  cpt ← 0
  for each s ∈ set do
    if id[s.a1] ≠ nil and id[s.a2] ≠ nil then
      for each s' ∈ S do
        if id[s'] = id[s.a1] then
          id[s'] ← id[s.a2]
    if id[s.a1] = nil and id[s.a2] = nil then
      id[s.a1] ← id[s.a2] ← cpt
      cpt ← cpt + 1
    if id[s.a1] ≠ nil and id[s.a2] = nil then
      id[s.a2] ← id[s.a1]
    if id[s.a1] = nil and id[s.a2] ≠ nil then
      id[s.a1] ← id[s.a2]
```

facteur 39.24 et le nombre de backtracks d'un facteur 131.96. En effet, le nombre de backtracks est en général réduit d'un gros facteur, ce qui permet d'obtenir une bonne réduction du temps de résolution.

Nous considérons maintenant différentes stratégies, notamment LCFirst maxCost introduite par Fages *et al.* [4]. Elle conserve pour la dernière arête branchée une de ses deux extrémités et sélectionne les arêtes parmi le voisinage du nœud par leurs coûts décroissants. Elle est considérée comme la meilleure stratégie actuelle pour résoudre le TSP en PPC.

	(1) Talos WCC statique maxCost		(2) Talos WCC + k-cutset statique maxCost		Ratios (1) / (2)	
Instance	time(ms)	#bk	time(ms)	#bk	time	#bk
gr96	9997	14988	2438	1512	4.1	9.91
rat99	60	36	73	40	0.82	0.9
kroA100	67044	96414	14829	9444	4.52	10.21
kroB100	178435	294686	12319	7178	14.48	41.05
kroC100	4743	4246	2702	1754	1.76	2.42
kroD100	509	474	496	282	1.03	1.68
kroE100	863352	1609168	22004	12194	39.24	131.96
rd100	29	12	26	4	1.12	3.0
eil101	126	122	124	88	1.02	1.39
lin105	9	2	10	0	0.9	Inf
pr107	338	10	339	10	1.0	1.0
gr120	4493	3790	1811	960	2.48	3.95
pr124	1065	544	919	304	1.16	1.79
bier127	391	398	295	124	1.33	3.21
ch130	13476	11172	5178	2290	2.6	4.88
pr136	t.o	2234970	114673	35584	$\geq$ 15.7	$\geq$ 62.81
gr137	21419	11814	17634	6822	1.21	1.73
pr144	1203	454	1285	418	0.94	1.09
kroA150	1025157	601084	171299	59034	5.98	10.18
kroB150	t.o	1087292	939634	302318	$\geq$ 1.92	$\geq$ 3.6
ch150	8626	5362	4188	1532	2.06	3.5
brg180	2843	1400	3238	1390	0.88	1.01
rat195	1656448	662298	915603	287322	1.81	2.31
d198	343600	166470	170325	54512	2.02	3.05
kroA200	t.o	913968	t.o	402802	NaN	2.27
kroB200	t.o	880048	1415194	345516	$\geq$ 1.27	$\geq$ 2.55
gr202	15187	9636	6585	2316	2.31	4.16

TABLE 1 – Résultats de la WCC avec le filtrage k-cutset et stratégie statique maxCost.

De façon surprenante, la table 2 montre que le filtrage k-cutset n'est pas intéressant pour la stratégie LCFirst maxCost. D'après nos expériences, la stratégie semble très ad hoc par rapport au propagateur de la contrainte WCC et notamment à la relaxation lagrangienne. Elle semble corriger une partie du manque de contraintes structurelles de la WCC. Nous observons des résultats similaires à la table 1 pour d'autres stratégies dynamiques, telle que LCFirst minCost.

Aussi, nous proposons d'utiliser la stratégie minRepCost proposée dans [4] qui est mieux adaptée à notre modèle. Elle consiste à sélectionner les arêtes par leurs coûts de remplacement [2] croissant. La table 3 permet d'observer une réduction conséquente de l'espace de recherche et une réduction moindre du temps de calcul. Par exemple, gr137 a un facteur de gain du temps de

	(1) Talos WCC LCFirst maxCost		(2) Talos WCC + k-cutset LCFirst maxCost		Ratios (1) / (2)	
Instance	time(ms)	#bk	time(ms)	#bk	time	#bk
kroB100	4844	4864	28446	16948	0.17	0.29
kroC100	872	730	996	538	0.88	1.36
kroD100	479	412	633	378	0.76	1.09
ch130	3501	2268	7568	3282	0.46	0.69
pr136	218637	149034	146158	52208	1.5	2.85
gr137	3570	1878	8546	3312	0.42	0.57
pr144	1109	304	1259	358	0.88	0.85
kroB200	240725	91708	1583689	390622	0.15	0.23
gr202	5916	3004	6217	2084	0.95	1.44

TABLE 2 – Résultats de la WCC avec le filtrage k-cutset avec la stratégie LCFirst maxCost.

résolution et du nombre de backtracks respectivement de 3.1 et 6.8.

	(1) Talos WCC LCFirst maxCost		(2) Talos WCC + k-cutset minRepCost		Ratios (1) / (2)	
Instance	time(ms)	#bk	time(ms)	#bk	time	#bk
gr96	1040	964	1093	482	0.95	2.0
rat99	63	44	67	20	0.94	2.2
kroA100	3161	2836	4735	1778	0.67	1.6
kroB100	4844	4864	1248	482	3.88	10.09
kroC100	872	730	320	96	2.72	7.6
kroD100	479	412	161	52	2.98	7.92
kroE100	3362	3354	3671	1422	0.92	2.36
rd100	39	18	30	8	1.3	2.25
eil101	90	68	42	24	2.14	2.83
lin105	10	2	9	0	1.11	Inf
pr107	343	10	342	10	1.0	1.0
gr120	769	540	590	210	1.3	2.57
pr124	981	432	1073	228	0.91	1.89
bier127	317	236	1670	318	0.19	0.74
ch130	3501	2268	2731	820	1.28	2.77
pr136	218637	149034	105412	23926	2.07	6.23
gr137	3570	1878	1150	276	3.1	6.8
pr144	1109	304	642	74	1.73	4.11
kroA150	21560	11510	15164	3234	1.42	3.56
kroB150	394760	217934	821804	176294	0.48	1.24
ch150	4355	1958	4417	1000	0.99	1.96
brg180	816	344	612	226	1.33	1.52
rat195	114889	40060	207379	29938	0.55	1.34
d198	23976	9132	456885	78262	0.05	0.12
kroA200	t.o	722550	t.o	248232	NaN	2.91
kroB200	240725	91708	236642	32338	1.02	2.84
gr202	5916	3004	2916	366	2.03	8.21

TABLE 3 – Résultats de la WCC avec le filtrage k-cutset et la stratégie minRepCost.

L'interaction du filtrage k-cutset avec la relaxation lagrangienne est peu clair, une étude plus approfondie devra être menée pour mieux la comprendre. Néanmoins nous observons que la contrainte WCC est construite autour de la relaxation lagrangienne. Rajouter un filtrage peut alors perturber la convergence de cette dernière et parfois, la ralentir. Cela explique pourquoi le facteur de gain du nombre de backtracks est toujours bien supérieur à celui du temps de résolution.

A titre indicatif, la table 4 compare notre méthode avec [4].

	(1) Choco WCC LCFirst maxCost		(2) Talos WCC + k-cutset minRepCost		Ratios (1) / (2)	
Instance	time(ms)	#bk	time(ms)	#bk	time	#bk
kroB100	11576	4427	1248	482	9.28	9.18
kroC100	2735	1005	320	96	8.55	10.47
kroD100	696	231	161	52	4.32	4.44
ch130	5547	1341	2731	820	2.03	1.64
pr136	554465	138457	105412	23926	5.26	5.79
gr137	10616	1907	1150	276	9.23	6.91
pr144	2262	307	642	74	3.52	4.15
kroA150	59529	11533	15164	3234	3.93	3.57
kroB200	867058	95941	236642	32338	3.66	2.97
gr202	15125	2667	2916	366	5.19	7.29

TABLE 4 – Comparatif de l'état de l'art et de Talos avec l'ajout du filtrage k-cutset à la WCC et la stratégie minRepCost.

6 Conclusion

Nous avons introduit une nouvelle contrainte structurelle dans la WCC basée sur la recherche de k-cutsets dans le graphe. Les résultats expérimentaux montrent l'intérêt de notre approche en pratique. Nous avons observé que le nombre de backtracks est réduit d'un ordre de magnitude en fonction de la stratégie choisie, ce qui nous permet d'obtenir un facteur de gain intéressant du temps de résolution. Les interactions entre cette contrainte et la stratégie de recherche, ainsi qu'entre cette contrainte et le modèle lagrangien de la WCC méritent une étude plus poussée.

7 Remerciment

Ces travaux ont été soutenus par l'Agence Nationale de la Recherche (ANR) à travers le projet MULTIMOD sous la référence ANR-17-CE22-0016.

Références

[1] David L APPLEGATE, Robert E BIXBY, Vasek CHVATAL et William J COOK : *The traveling salesman problem : a computational study*. Princeton university press, 2006.

[2] Pascal BENCHIMOL, Jean-Charles RÉGIN, Louis-Martin ROUSSEAU, Michel RUEHER et Willem-Jan van HOEVE : Improving the held and karp approach with constraint programming. *In* Andrea LODI, Michela MILANO et Paolo TOTH, éditeurs : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 40–44, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[3] Sylvain DUCOMMAN, Hadrien CAMBAZARD et Bernard PENZ : Alternative filtering for the weighted circuit constraint : Comparing lower bounds for the tsp and solving tsptw. *In AAAI*, 2016.

[4] Jean-Guillaume FAGES, Xavier LORCA et Louis-Martin ROUSSEAU : The salesman and the tree : the importance of search in cp. *Constraints*, 21(2): 145–162, 2016.

[5] Michael HAYTHORPE : Fhcp challenge set : The first set of structurally difficult instances of the hamiltonian cycle problem, 2019.

[6] Michael HELD et Richard M. KARP : The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[7] Michael HELD et Richard M. KARP : The traveling-salesman problem and minimum spanning trees : Part ii. *Mathematical Programming*, 1(1):6–25, 1971.

[8] Gerhard REINELT : Tsplib—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[9] Robert E. TARJAN : A note on finding the bridges of a graph. *Inf. Process. Lett.*, 2:160–161, 1974.

[10] Robert E. TARJAN : *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics, 1983.

[11] Yung H. TSIN : Yet another optimal algorithm for 3-edge-connectivity. *Journal of Discrete Algorithms*, 7(1):130 – 146, 2009. Selected papers from the 1st International Workshop on Similarity Search and Applications (SISAP).

[12] Li-Pu YEH, Biing-Feng WANG et Hsin-Hao SU : Efficient algorithms for the problems of enumerating cuts by non-decreasing weights. *Algorithmica*, 56(3):297–312, 2010.