Groupe de Recherche
Opérationnelle et Informatique
(ROI)

Mémoire de DEA AISIH
Automatique et Informatique des Systèmes Industriels et Humains

Le Problème du Voyageur de Commerce Multicolore

Mahdi KHEMAKHEM

Juillet 2003

Responsables : Frédéric SEMET
Gilbert LAPORTE
Roberto WOLFLER-CALVO

LAMIH – UMR CNRS 8530
Université de Valenciennes et du Hainaut Cambrésis
Le Mont Houy – 59313 VALENCIENNES – Cedex 9 – France

Remerciements

Ce n'est pas par tradition que cette page figure au préambule de ce mémoire, mais c'est plutôt un devoir moral et une reconnaissance sincère qui me pousse à le faire.

Je serais en effet ingrat si je n'exprime pas ma reconnaissance et ma gratitude à tous ceux qui, de près ou de loin, directement ou indirectement, m'ont facilité ma tâche et m'ont permis de mener à bien ce travail.

C'est au sein du laboratoire LAMIH et plus particulièrement le groupe ROI qu'ont été réunies toutes les bonnes conditions qui m'ont permis de réaliser ce travail.

Je tiens tout particulièrement à exprimer toute ma gratitude et mes vifs remerciements à Mr. Frédéric SEMET, directeur de l'équipe ROI, qui m'a accordé sa confiance en me proposant ce sujet de DEA. Je le pris de croire à ma respectueuse estime et ma sincère reconnaissance pour ses conseils qui m'ont été très précieux et indispensables.

Je ne saurais oublier de remercier Mr Gilbert LAPORTE et Mr Roberto WOLFLER-CALVO qui n'ont pas hésité à participer à ce travail avec leurs expériences et leurs savoir faire dans ce domaine de recherche. Qu'ils me soient permis de les exprimer l'assurance de ma gratitude et de mon profond respect.

Messieurs les membres de la commission de jury, me font l'honneur d'avoir accepter d'évaluer ce travail, je tiens à les remercier chaleureusement.

Mahdi KHEMAKHEM

Tables des matières

Chapitre 1 : Introduction

Dans ce chapitre nous allons exposer et définir le problème de voyageur de commerce multicolore, puis nous présenterons ces intérêts et les domaines d'application qui peuvent être appliquer à ce problème et nous finirons par un bref historique sur le problème de voyageur de commerce.

1.1 Exposition du problème:

Le but de ce projet de recherche est de résoudre le Problème du Voyageur de Commerce Multicolore (PVCM). Ce problème est une extension du Problème du Voyageur de Commerce (PVC), ou Travelling Salesman Problem (TSP) selon la terminologie anglaise, à *n* types de sommets.

Pour introduire le PVCM nous commençons par définir le PVC. Il consiste à déterminer le cycle le plus court passant exactement une fois par chacun des sommets d'un graphe et qui possède la longueur minimale. Ce cycle est appelé cycle Hamiltonien. Le PVC peut être considérer sous deux formes, l'une est symétrique PVCS et l'autre est asymétrique PVCA. Pour notre cas nous avons pris la dernière forme pour généraliser le problème. Nous prendrons en compte les notations, du PVCA, suivantes :

Soit un graphe $G=(V,A)$ tel que $V=\{v_1,\ldots,v_n\}$ l'ensemble des sommets, $n=|V|$ le nombre des sommets du graphe G et C la matrice de distance de taille nxn, tel que c_{ij} définit la distance entre les deux sommets v_i et v_j. Pour le PVCA, G doit être asymétrique, dans ce cas nous définirons $A=\{(v_i,v_j)$ tels que $v_i,v_j \in V\}$ l'ensemble des arcs du graphe G. Le faite d'avoir un arc $(v_i,v_j) \in A$ signifie que nous pouvons aller de v_i à v_j mais pas forcément de v_j à v_i et nous pouvons avoir $c_{ij} \neq c_{ji}$.

Le PVCM, qui est une extension du PVCA, consiste à construire un cycle hamiltonien de longueur minimale en tenant compte de contraintes d'espacement à chaque type de sommets. Tout type est caractérisé par une couleur. Dans un cycle et entre deux sommets v_i à v_j de même couleur il doit y exister un nombre de sommets de couleurs différentes. Ce nombre est compris entre un minimum et un maximum fixé auparavant.

Nous ajoutons les notations suivantes pour le PVCM. Soit $W_i \subset V$ tels que $i \in \{1,\ldots,H\}$ des sous ensembles de V contenant les sommets de même couleur, H est le nombre de couleurs. $C_h=\{(v_i,v_j)$ tels que $v_i,v_j \in W_h\}$. a_h et b_h sont respectivement le nombre minimum et maximum

d'arcs qui peuvent connecter deux sommets successifs de couleur *h*. En d'autres termes, entre ces deux sommets nous devons trouver au minimum $(a_h - 1)$ et au maximum $(b_h - 1)$ sommets de couleurs différentes. En général, nous ne pouvons pas trouver deux sommets adjacents de même couleur.

Pour illustrer la définition de ce problème, nous considérons un exemple de trois sommets à visiter, le premier (v_1) doit être visiter 3 fois, le deuxième (v_2) 5 fois et le troisième (v_3) 4 fois, $a_1 = 2, b_1 = 6, a_2 = 2, b_2 = 4, a_3 = 2, b_3 = 4.$

Cet exemple peut être représenté sous la forme d'un PVCM de la façon suivante. Nous choisissons une couleur pour chaque sommet et nous reproduisons chaque sommet autant de fois qu'il devait être visité (voir figure 1.2).

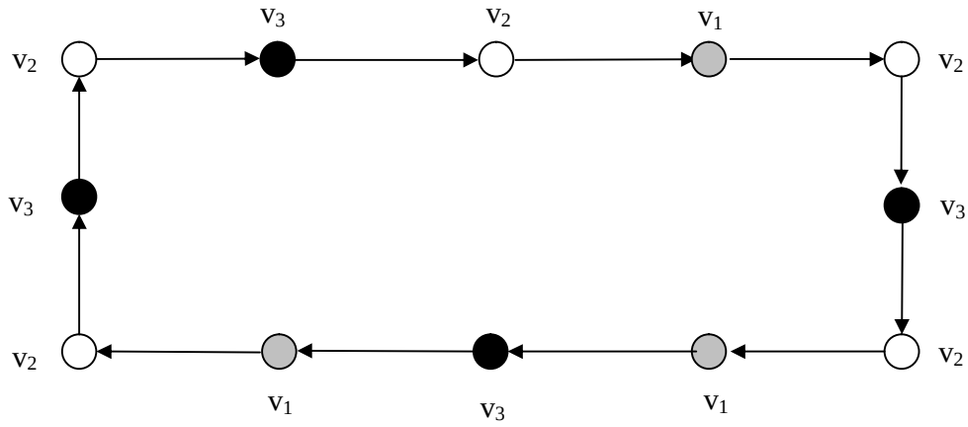


Figure 1.1 Présentation d'un PVCM sous forme colorée

1.2 Intérêt et domaines d'application :

De nos jours la sécurité est devenue une obligation et utilise les dernières technologies et les compétences de la gestion. Les vigiles appelés « agent de sécurité ou de surveillance », sont de plus en plus employés, ceci afin de faire face et de palier à un manque de sécurité flagrant au vue des actes de vandalisme et de malveillance commis auprès des commerçants, entreprises et même des particuliers [10]. Les compagnies de sécurité offrent une large variété de services pour satisfaire aux besoins de leurs clients, tels que la surveillance des banques, des centres commerciales…

La ronde du veilleur de nuit est une application du PVCM [2]. Un veilleur doit faire sa tournée en optimisant la longueur de son parcours et par suite la durée de sa tournée et en tenant compte du degré d'importance attribué à chaque local qu'il doit visiter n_h fois par nuit.

L'ensemble des sommets de même couleur correspond à un même local. La cardinalité d'un tel ensemble désigne le nombre des visites à effectuer pour cet endroit.
Lorsque nous reproduisons le cycle présenté par la figure 1.1 sous sa forme initiale, nous obtenons celui présenté par la figure 1.2.

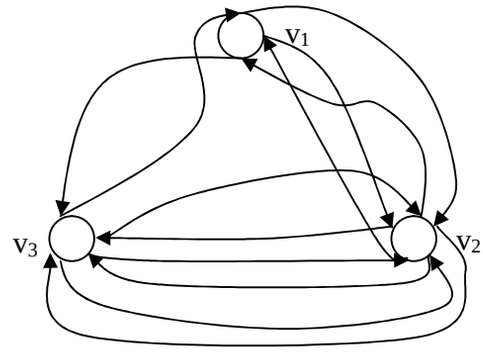


Figure 1.2 : Présentation d'un PVCM sous forme initiale

La détermination anticipée de la ronde du veilleur de nuit, permet au veilleur d'établir sa tournée normalement et d'éliminer les failles de la sécurité qui peuvent survenir pendant la nuit. En effet, la fixation des limites a_h et b_h dépend directement de l'importance et du taux de sécurité attribué au sommet de type *h*.

1.3 Etat de l'art :

Puisque le PVCM est une extension du PVC, il est nécessaire de faire un bref historique et de rappeler quelques travaux qui ont apporté des bonnes solutions pour ce problème.

Les premières approches mathématiques exposées pour le problème du voyageur de commerce ont été traitées au 19^{éme} siècle par les mathématiciens. Il a été décrit pour la première fois par Karl Meneger depuis l'année 1930. Ensuite, il a été étudié par les statisticiens Mahalanobis (1940), Jessen (1942), Gosh (1948) et Julia Robenson (1949). Durant les années 50, un nombre considérable de solutions optimales a été atteint grâce à différentes méthodes comme la formulation de George B. Dantzig, Fulkerson et Johnson en 1954 qui a pu résoudre jusqu'à 49 villes. En 1956 Merill M.Flood a publié quelques heuristiques générant des bonnes solutions et en 1957, Barachet a mis en oeuvre une méthode graphique pour la résolution du PVC.

Depuis 1954, le souci des chercheurs était d'aller le plus loin que possible au niveau du nombre de villes à considérer dans ce problème. Le tableau suivant illustre les tailles maximales des instances résolues du PVC sous sa forme symétrique [7]:

Année	Auteurs	Nombre de villes
1954	*G.Dantzig, R.Fulkerson et S.Jhonson*	*49*
1971	*M. Held et R.M. Karp*	*64*
1975	*P.M.Camerini, L.Fratta et F.Maffioli*	*100*
1977	*M.Grotschel*	*120*
1980	*H.Crowder et M.W.Padberg*	*318*
1987	*M.Padberg et G.Rinaldi*	*532*
1987	*M. Grotschel et O.Holland*	*666*
1987	*M.Padberg et G.Rinaldi*	*2392*
1994	*D.Appelgate, R.Bixby. V.Chvatal et W.Cook*	*7397*
1998	*D.Appelgate, R.Bixby. V.Chvatal et W.Cook*	*13509*
2001	*D.Appelgate, R.Bixby. V.Chvatal et W.Cook*	*15112*

Tableau 1.1 : Taille maximale des instances résolues

Les solutions trouvées durant toutes ces années sont réalisées en utilisant différents modèles et différentes techniques. Nous pouvons citer les plus importantes tels que la formulation de Dantzig - Fulkerson et Johnson en 1954 sous ces deux formes symétrique et asymétrique en prenant compte de la contrainte d'élimination des sous tours. Elle a été résolue par les algorithmes de « Branch and Bound » et « Branch and Cut » en utilisant des techniques de relaxation et de calcul d'une borne inférieur. En 1960, Miller - Tucker et Zemlin ont proposé une autre formulation en reformulant la contrainte d'élimination des sous-tour de façon à ce qu'elle génère un nombre polynomial de contraintes [5]. Leur contrainte a été améliorée par Desrochers et Laporte en 1991 [1].

En plus des formulations et des programmes linéaires, le PVC a été résolu par des heuristiques générant des bonnes solutions proches de l'optimum. Nous pouvons citer celle du plus proche voisin utilisée pour les deux forme du PVC introduite Rosenkrantz, Stearns et Lewis [14]. Nous pouvons ajouter les algorithmes d'insertion de la classe des heuristiques constructives étudiés par Rosenkrantz, Stearns et Lewis [14], Stewart [15] et Norback et Love [16,17]. L'algorithme d'amélioration r-opt de Lin en 1965 [7] pour le cas symétrique et celui de Lin-Kernighan en 1973 [7], La dernière a été implémenté par Mark et Morton en 1993 [8]. En 1992 Gendreau, Hertz et Laporte ont publié GENIUS qui compte parmi les meilleures heuristiques pour le PVC symétrique [9].

L'intérêt accordé par les scientifiques tout au long de ces années au problème du voyageur de commerce est motivé par son importance dans la vie courante vu son large domaine d'application.

En ce qui concerne le problème du voyageur multicolore, il est à signaler le manque d'études qui s'y intéressent puisqu'il apparaît peu dans la littérature. Le travail le plus proche et similaire au PVCM est le « Problème du Voyageur de Commerce avec sommets Noir et Blanc » PVCNB ou « Black and White Travelling Salesman Problem » BWTSP selon la terminologie anglaise. Ce problème est résolu par des heuristiques par Bourgeois, Laporte et Semet [3]. Il consiste à établir un cycle hamiltonien en tenant compte seulement de deux couleurs de sommets. En effet, chaque sommet est caractérisé soit par la couleur blanche ou la couleur noir. Le PVCNB a été résolu par une méthode exacte par Ghiani, Laporte et Semet [19]. La taille maximale d'instance résolue par cette méthode est de 100 sommets. .

Dans les chapitres suivants, nous présenterons la façon dont nous avons adapté les modèles et techniques de résolutions du PVC, déjà établis dans la littérature, au problème du voyageur de commerce multicolore.

Chapitre 2 : Formulation du PVCM et résolution exacte

La transformation du PVC vers le PVCM n'est pas évidente. Nous allons montrer dans ce chapitre la façon que nous avons adoptée pour avoir une formulation sous forme d'un programme linéaire du PVCM afin de le résoudre de façon exacte. Nous commencerons par introduire le programme linéaire du PVCA, ensuite nous présenterons la formulation du PVCM tout en argumentant nos choix et dégageant les problèmes rencontrés durant cette phase.

2.1 Formulation du problème du voyageur de commerce

Pour commencer, nous avons considéré la formulation du problème du voyageur de commerce, sous forme asymétrique. Cette formulation a été proposée pour la première fois par Dantzig, Fulkerson et Jonhson en 1954 [4], nous pouvons la décrire de la façon suivante : Soit x_{ij} une variable binaire telle que x_{ij}=1 si l'arc (*i*,*j*) appartient à la solution du PVCA et x_{ij}=0 sinon. Donc on veut :

$$\text{Minimiser} \sum_{(i,j)\in A} c_{ij} x_{ij} \qquad (1)$$

Sous les contraintes :

$$\sum_{v_j \in V} x_{ij} = 1 \qquad \forall v_i \in V \text{ et } v_i \neq v_j \qquad (2)$$

$$\sum_{v_j \in V} x_{ji} = 1 \qquad \forall v_i \in V \text{ et } v_i \neq v_j \qquad (3)$$

$$\sum_{v_i, v_j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset V, 3 \leq |S| \leq (n-3) \qquad (4)$$

$$x_{ij} \in \{0,1\} \qquad \forall v_i, v_j \in V \qquad (5)$$

Les contraintes (2) et (3) assurent qu'un sommet n'est visité qu'une fois par cycle : on y arrive une et une seule fois (contrainte (2)) et on en part une et une seule fois (contrainte (3)).
La contrainte (5), appelée contrainte d'intégrité, détermine le domaine des variables x_{ij}.
La contrainte (4) est une contrainte d'élimination des sous tours avec *S* est un sous-ensemble de sommet de *V*. Par exemple, la forme du cycle présenté par (figure 2.1) n'est pas permise par cette contrainte. Ce cycle n'est pas hamiltonien, donc il ne peut pas être une solution du PVC.

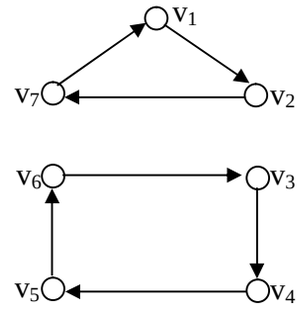


Figure 2.1 Exemple de sous tours

Grotschel et Patberg [11], ont prouvés que cette contrainte est une facette du polytope de l'enveloppe convexe définit par les contraintes de (2) à (5).

Si la cardinalité de *S* est égale à 2, la somme $\sum_{v_i,v_j \in S} x_{ij}$ vaut au maximum 1 donc la contrainte est inutile. Cette contrainte génère un nombre exponentiel de contraintes selon le nombre des sommets à résoudre.

En 1960, Miller, Tucker et Zemlin (M.T.Z) ont proposé une contrainte d'élimination des sous tours qui génère un nombre polynomial de contraintes, elle peut être exprimée comme il suit [5] :

$$u_i - u_j + (n-1)x_{ij} \leq (n-2) \qquad \forall v_i, v_j \in V - \{v_1\} \qquad (6)$$

$$1 \leq u_i \leq n-1 \qquad \forall v_i \in V - \{v_1\} \qquad (7)$$

où u_i désigne l'ordre du sommet i dans le cycle.

Cette contrainte assure qu'il y à uniquement une solution qui correspond au cycle réalisable. Si nous appliquons cette contrainte au sous-cycle composé par $\{v_3,v_4,v_5,v_6\}$ de l'exemple précédent (figure 2.1), nous obtenons les contraintes qui mènent à une contradiction :

$$(u_3 - u_4) + (n-1)x_{34} \leq (n-2)$$
$$(u_4 - u_5) + (n-1)x_{45} \leq (n-2)$$
$$(u_5 - u_6) + (n-1)x_{56} \leq (n-2)$$
$$(u_6 - u_3) + (n-1)x_{63} \leq (n-2)$$

En somment ces contraintes nous obtenons $(n-1) \leq (n-2)$!!!

Ce sous tour n'est pas permis par cette contrainte, mais elle représente une faiblesse. Elle n'est en effet pas une facette pour le polytope de l'enveloppe convexe des solutions réalisables définit par les contraintes (2), (3), (6), (7) et (5). Pour cela, cette contrainte a été amélioré par Laporte et

Desrochers en 1990 [1]. Leur proposition est la contrainte suivante et qui représente une inégalité valide pour le PVC :

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq (n-2) \qquad \forall v_i, v_j \in V - \{v_1\} \qquad (8)$$

➲ *Preuve [1] :*

Cette contrainte est obtenue en utilisant la technique de *lifting* [12,13] sur la contrainte (6). On considère la contrainte d'élimination des sous tours de M.T.Z [5] et on lifte la variable x_{ji}.

$u_i - u_j + (n-1)x_{ij} + a_{ji}x_{ji} \leq (n-2)$ avec $a_{ji} = 0$ (*)

Le processus de *lifting* prend la plus grande valeur possible de a_{ji} pour que (*) demeure une inégalité valide.

Deux cas qui peuvent se présenter : $x_{ji} = 0$ et $x_{ji} = 1$.

Cas1 : $x_{ji} = 0$, (*) est satisfaite pour n'importe quel a_{ji}.

Cas2 : $x_{ji} = 1 \Rightarrow x_{ij} = 0$(pour $n \succ 2$)

$\Rightarrow u_i - u_j = 1$

donc :

$1 + a_{ji} \leq n - 2$

$a_{ji} \leq n - 3$

Puisque a_{ji} prend la plus grande valeur on prend a_{ji} =n-3, d'où la contrainte valide (6) du PVC. □

Pour la suite nous allons prendre en considération la formulation du PVCA composée par les contraintes (1), (2), (3), (8), (7) et (5).

$$\textit{Minimiser} \sum_{(i,j)\in A} c_{ij}x_{ij} \qquad (1)$$

Sous les contraintes :

$$\sum_{v_j \in V} x_{ij} = 1 \qquad \forall v_i \in V \text{ et } v_i \neq v_j \qquad (2)$$

$$\sum_{v_j \in V} x_{ji} = 1 \qquad \forall v_i \in V \text{ et } v_i \neq v_j \qquad (3)$$

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq (n-2) \qquad \forall v_i, v_j \in V - \{v_1\} \qquad (8)$$

$$1 \leq u_i \leq n-1 \qquad \forall v_i \in V - \{v_1\} \qquad (7)$$

$$x_{ij} \in \{0,1\} \qquad \forall v_i, v_j \in V \qquad (5)$$

2.2 Généralisation du PVC au PVCM

La généralisation du PVC vers le PVCM n'est pas évidente vue l'existence de la contrainte des couleurs. Nous avons rencontré beaucoup de problèmes lors de la construction de la formulation du programme linéaire en nombre entier du PVCM.

Nous avons pris comme point de départ la dernière formulation du PVCA vue dans la section précédente. Nous avons introduit des modifications sur quelques contraintes aussi bien nous avons ajouté des contraintes spécifiques au PVCM.

Le PVCM est caractérisé par les couleurs, chaque couleur représente un type de sommets bien déterminer. Chaque ensemble de sommets, de même couleur, doit être considéré comme un seul PVC. Pour cela, il faut appliquer la formulation du PVC sur la totalité des sommets, puis sur chaque ensemble de sommet de même couleur.

La formulation doit prendre en compte le cycle entier aussi bien que chaque cycle de sommets de même couleur tout seul. En d'autre terme, elle doit permettre de résoudre plusieurs problème en un seul. Nous pouvons schématiser les différents cycles à résoudre, pour un exemple de 3 couleurs et de 3 visites chacune, comme suit (figure 2.2):

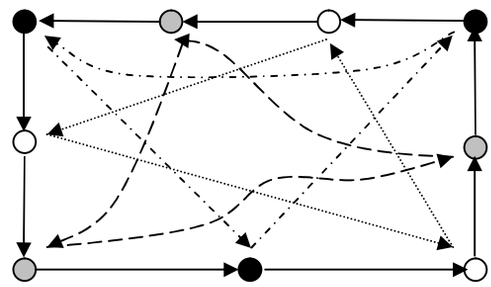


Figure 2.2 : Cycles du PVCM à résoudre

Pour les contraintes d'affectations il faut les appliquer sur tous les sommets du problème aussi bien pour chaque ensemble de sommets de même couleur. Pour cela il faut bien introduire une variable binaire supplémentaire y_{ij}^{h} qui est égale à 1 si les deux sommets *i* et *j*, de même couleur *h*, sont successifs et 0 sinon. La contrainte d'élimination des sous tours d'un PVC classique est destinée pour éliminer les sous tour entre tous les sommets du cycle. Pour la contrainte d'élimination des sous tours du PVCM, appliqué sur chaque ensemble de sommets de même couleur, doit tenir compte des limites a et b . En effet, la différence entre les ordres de deux sommets, de même couleurs et successives, doit être comprise entre le minimum a et le maximum b .

2.3 Formulation du PVCM

2.3.1 Première extension de la formulation classique du PVC

La première formulation est établie suivant les principes généraux du PVCM et selon les premières idées intuitives. Cette formulation présente des problèmes, qui ont été dévoilés lors de l'expérimentation sur un problème simple, que nous les préciserons à la fin de ce paragraphe.
Nous considérons les notations suivantes :

- x_{ij}=1 si l'arc *(i,j)* est utilisé, sinon x_{ij}=0.
- y_{ij}^h a le même sens que x_{ij}, mais elle existe sauf pour les sommets de même couleur.
- La variable u_i représente la position d'un sommet v_i dans le cycle.
- v_0^h : c'est le premier sommet de chaque ensemble de sommets de même couleurs.
- C_{ij} : le coût lorsque on passe du sommets v_i au sommets v_j. Si v_i et v_j sont de même couleurs, $c_{ij}=\infty$ pour que la valeur de la relaxation en continu dans l'algorithme de séparation et d'évaluation ne prenne pas une solution qui contient des sommets adjacents de même couleur. De cette façon nous réduisons le champ de recherche de l'arbre de Branch and Bound.

La formulation établit est la suivante :

$$Minimiser \sum_{(i,j)\in A} c_{ij} x_{ij} \qquad (1)$$

s.c

$$\sum_{v_j \in V} x_{ij} = 1 \qquad \forall v_i \in V \text{ et } v_i \neq v_j \qquad (2)$$

$$\sum_{v_j \in V} x_{ji} = 1 \qquad \forall v_i \in V \text{ et } v_i \neq v_j \qquad (3)$$

$$\sum_{v_j \in W_h} y_{ij}^h = 1 \qquad \forall h \in \{1,...,H\}, \forall v_i \in W_h \text{ et } v_i \neq v_j \qquad (9)$$

$$\sum_{v_j \in W_h} y_{ji}^h = 1 \qquad \forall h \in \{1,...,H\}, \forall v_i \in W_h \text{ et } v_i \neq v_j \qquad (10)$$

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq (n-2) \qquad \forall v_i, v_j \in V - \{v_0\} \text{ et } v_i \neq v_j \qquad (8)$$

$$u_i - u_j + (n-a_h)\, y_{ij}^h + (n-2a_h-b_h)\, y_{ji}^h \leq (n-2a_h) \qquad \forall h \in \{1,...,H\}, \forall v_i, v_j \in W_h - \{v_0^h\} \text{ et } v_i \neq v_j \qquad (11)$$

$$x_{ij} \in \{0,1\} \qquad \forall (v_i, v_j) \in A \qquad (5)$$

$$y_{ij} \in \{0,1\} \qquad \forall (v_i, v_j) \in C_h \qquad (12)$$

$$1 \le u_i \le n-1 \qquad \forall v_i \in V - \{v_0\} \qquad (7)$$

Les contraintes (2) et (3) assurent qu'un sommet n'est visité qu'une fois par cycle : on y arrive une et une seule fois (contrainte (2)) et on en part une et une seule fois (contrainte (3)).
Les contraintes (9) et (10) assurent qu'un sommet n'est visité qu'une fois par cycle. Ces deux contraintes sont appliquées sur les éléments de chaque ensemble de sommets de même couleur.
La contrainte (8) représente la contrainte d'élimination de sous-tour pour le PVC améliorée par Desrochers et Laporte [1].
La contrainte (11) représente une contrainte d'élimination des sous tours entre les éléments de chaque ensemble de sommets de même couleurs. Elle détermine encore la limite du nombre des sommets entre deux sommets de même couleur et successifs.
Les contraintes (5) (12) et (7) représentent les contraintes d'intégrité.

Montrons que la contrainte (11) est valide :
On prend un exemple de deux couleurs et de trois sommets pour chaque couleur et $a_h = b_h = 2 \quad \forall h \in \{0,1\}$.

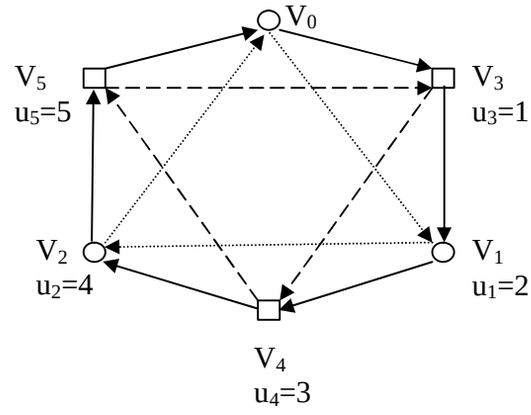


Figure 2.3 : Instance du PVCM à 2 couleurs et 6 sommets

Le premier sommet de chaque couleur n'est pas prit en considération dans la contrainte (11). Dans notre exemple les deux sommets v_0 pour la première couleur et v_3 pour la deuxième couleur.

Donc : $u_i - u_j + a y_{ij}^h + b y_{ji}^h \le n - 2a_h$

* *Si* $y_{ij}^h = 1$ et $y_{ji}^h = 0$

$u_i - u_j + a \le n - 2a_h$

$u_i - u_j \le n - 2a_h - a$

on a $-b_h \le u_i - u_j \le -a_h$

$\Rightarrow a = n - a_h$

* *Si* $y_{ij}^h = 0$ et $y_{ji}^h = 1$

$u_i - u_j + b \le n - 2a_h$

$u_i - u_j \le n - 2a_h - b$

on a $b_h \ge u_i - u_j \ge a_h$

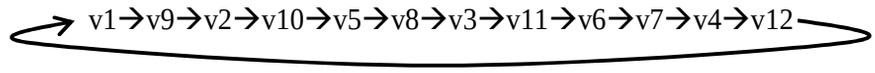
$\Rightarrow b = n - 2a_h - b_h$

Donc : $u_i - u_j + (n - a_h) y_{ij}^h + (n - 2a_h - b_h) y_{ji}^h \le n - 2a_h$ □

Nous avons testé un exemple de 4 couleurs et de 3 sommets pour chacune et qui a les caractéristiques suivantes :

- Couleur1 : sommets (v1,v2,v3) a_1=4, b_1=4.
- Couleur2 : sommets (v4,v5,v6) a_2=4, b_2=4.
- Couleur3 : sommets (v7,v8,v9) a_3=4, b_3=4.
- Couleur4 : sommets (v10,v11,v12) a_4=4, b_4=4.

La solution donnée par le logiciel d'optimisation CPLEX en utilisant la première formulation est la suivante :



Nous remarquons que les sommets v1 et v4, qui sont les premiers sommets (respectivement) de la couleur1 et de la couleur2, ne respectent pas les limites (respectivement) a_1=4, b_1=4 et a_2=4, b_2=4.

Le problème c'est que la contrainte (11) ne peut pas être appliquée au premier élément de chaque ensemble de sommets, de même couleur, pour assurer l'élimination des sous tours. Par contre chaque sommet, même le sommet de départ v_1 doit figurer dans cette contrainte pour imposer la limite du nombre des sommets maximum et minimum entre deux sommets successifs de même

couleur. Pour l'exemple cette contrainte n'est pas appliquée pour les sommets v_1, v_4, v_7 et v_{10}. Les deux derniers sommets ont respecté leurs limites par chance.

2.3.2 Modification du modèle

L'idée de la deuxième formulation est d'ajouter un sommet fictif V_{vir}.. Il intervient dans toutes les contraintes sauf les contraintes d'élimination des sous tours. Nous considérons ce sommet comme départ pour tous les cycles. Il n'est pas pris en considération dans la solution. Son coût par rapport au autre sommets est nul ($c_{i\ vir} = 0, c_{vir\ j} = 0 \quad \forall v_i, v_j \in V$). Nous pouvons l'appeler encore sommet virtuel.

Le fait de prendre en compte ce sommet au début et de l'enlever lorsque l'on récupère la solution entraîne un problème d'optimisation. En effet, lors de la récupération du cycle solution du problème, le sommet virtuel va être insérer entre deux autres sommets V_i et V_j (voir figure 2.4). Le coût C_{ij} , entre ces deux sommets en considérant le sommet virtuel, est nul. Lorsque nous éliminons le sommet virtuel de la solution, le coût entre les deux sommets adjacents va prendre une valeur non nulle. Ce coût doit être le minimum possible, donc il faut prendre en compte cette contrainte dés le début.

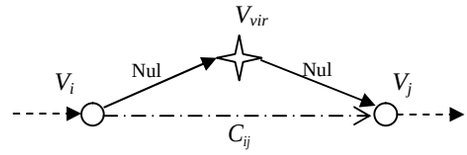


Figure 2.4 : Sommet virtuel

Pour prendre en compte la minimisation du coût entre les deux sommets adjacents au sommet virtuel, nous avons procédé de la façon suivante.

Nous choisissons un sommet quelconque de notre ensemble de sommets V et nous dupliquons un autre sommet de mêmes caractéristiques. Par exemple, nous choisissons le sommet V_0 *et* $V_{0'}$ son duplicata. Ensuite, nous obligeons le fait que le sommet virtuel V_{vir} soit entre ces deux sommets (voir figure 2.5).

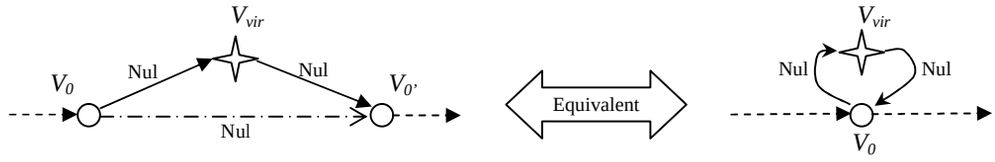


Figure 2.5 : Sommet virtuel et sommet dupliqué

Le sommet $V_{0'}$ intervient que dans les contraintes d'affectations du cycle global et dans la contrainte qui oblige que V_{vir} se situe entre V_0 *et* $V_{0'}$.

On note :

- v_{vir} : désigne le sommet fictif.
- $v_{0'}$: désigne le sommet duplicata de v_0.
- $n'=n+2$: le nombre total des sommets en tenant compte du sommet fictif et du sommet dupliqué.
- $A=\{(v_i,v_j)$ *tels que* $v_i,v_j \hat{I} V \cup \{v_{vir},v_{0'}\}\}$.

Le nouveau model est le suivant:

$$Minimiser \sum_{(i,j)\in A} c_{ij}x_{ij} \qquad (1)$$

sc

$$\sum_{v_j\in V} x_{ij} = 1 \qquad \forall v_i \in V \cup \{v_{vir}, v_{0'}\} \text{ et } v_i \neq v_j \qquad (13)$$

$$\sum_{v_j\in V} x_{ji} = 1 \qquad \forall v_i \in V \cup \{v_{vir}, v_{0'}\} \text{ et } v_i \neq v_j \qquad (14)$$

$$\sum_{v_j\in W_h} y_{ij}^h = 1 \qquad \forall h \in \{1,...,H\}, \forall v_i \in W_h \cup \{v_{vir}\} \text{ et } v_i \neq v_j \qquad (15)$$

$$\sum_{v_j\in W_h} y_{ji}^h = 1 \qquad \forall h \in \{1,...,H\}, \forall v_i \in W_h \cup \{v_{vir}\} \text{ et } v_i \neq v_j \qquad (16)$$

$$u_i - u_j + (n'-1)x_{ij} + (n'-3)x_{ji} \leq (n'-2) \qquad \forall v_i, v_j \in V \cup v_{0'},\ v_i \neq v_j \qquad (17)$$

$$u_i - u_j + (n'-a_h)y_{ij}^h + (n'-2a_h-b_h)y_{ji}^h \leq (n'-2a_h) \qquad \forall h \in \{1,...,H\}, \forall v_i, v_j \in W_h \text{ et } v_i \neq v_j \qquad (18)$$

$$x_{0vir} + x_{vir0'} = 2 \qquad (19)$$

$$x_{ij} \in \{0,1\} \qquad \forall (v_i, v_j) \in A \qquad (5)$$

$$y_{ij} \in \{0,1\} \qquad \forall (v_i, v_j) \in C_h \qquad (12)$$

$$1 \le u_i \le n' - 1 \qquad \forall v_i \in V \qquad (20)$$

Les contraintes (13) et (14) ont la même signification que les deux contraintes (2) et (3), mais ici elles tiennent compte du sommet fictif et du sommet dupliqué.

Les contraintes (15) et (16) ont la même signification que les deux contraintes (9) et (10), mais ici elles tiennent compte du sommet fictif. Chaque cycle de sommets de même couleur passe obligatoirement par le sommet v_{vir}.

La contrainte (17) a la même signification que la contrainte (8), mais elle est appliquée sur tous les sommets sauf v_{vir}.

La contrainte (18) a la même signification que la contrainte (11), mais elle est appliquée sur tous les sommets de même couleurs sauf v_{vir}.. Cela pour remédier au problème rencontrer par la première formulation.

La contrainte (19) oblige l'insertion de v_{vir} entre v_0 et $v_{0'}$.

Les contraintes (5) (12) et (20) représentent les contraintes d'intégrité.

Cette formulation assure que tous les sommets de chaque couleur sont pris par la contrainte d'élimination des sous tours et respectent les limites a_h et b_h.

Remarque : Le sommet v_{vir} est pris comme sommets de départ pour le cycle global et pour chaque cycle de sommets de même couleur.

Cette deuxième formulation n'est pas correcte car elle ne peut pas résoudre le problème de 4 couleurs et de 3 sommets pour chacune. Le résultat est que ce problème est irréalisable malgré qu'on puisse obtenir une solution réalisable manuellement. Nous avons conclu qu'il y à une contradiction dans une ou plusieurs contraintes. Cette contradiction est provoquée par la contrainte (18).

Pour montrer cette contradiction prenons, pour faciliter la compréhension, l'exemple de deux couleurs et 3 sommets pour chacune et nous ajoutons le sommet virtuel et le sommet dupliqué.

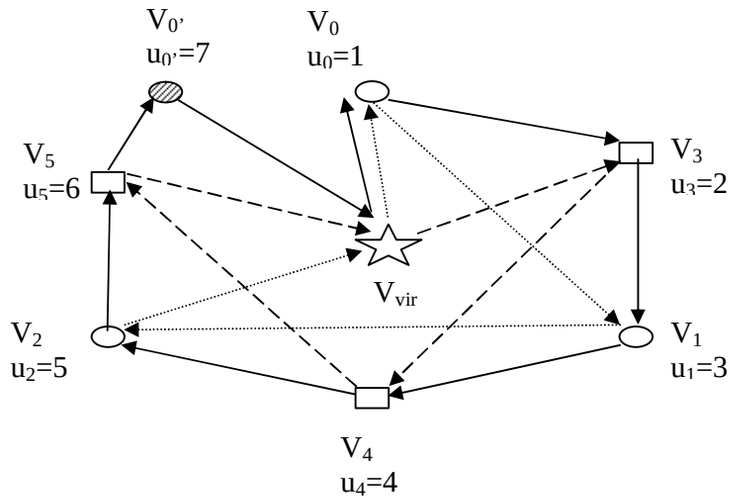


Figure 2.6 : Les cycles pris en considération par le deuxième modèle du PVCM

La contrainte (18) est la suivante :

$$u_i - u_j + (n' - a_h)\, y_{ij}^h + (n' - 2a_h - b_h)\, y_{ji}^h \leq (n' - 2a_h)$$

Nous prenons le cas où : $y_{ij}^h = y_{ji}^h = 0$ et $y_{j\,vir}^h = y_{vir\,i}^h = 1$

D'une part la contrainte (18) donne : $u_i - u_j \leq n' - 2a_h$ (I)

D'autre part nous avons : $u_i - u_j \geq n' - (a_h + 2)$

$$u_i - u_j \geq n' - a_h - 2 \qquad \text{(II)}$$

(I) et (II) donnent une contradiction car $n - 2a_h \geq n - a_h - 2 \quad \forall\, h$ □

2.3.3 Formulation complète du PVCM

Pour remédier à la contradiction rencontrée avec la deuxième formulation, nous avons proposé la formulation suivante :

$$Minimiser \sum_{(i,j)\in A} c_{ij} x_{ij} \qquad (1)$$

sc

$$\sum_{v_j \in V} x_{ij} = 1 \qquad \forall v_i \in V \cup \{v_{vir}, v_{0'}\} \text{ et } v_i \neq v_j \qquad (13)$$

$$\sum_{v_j \in V} x_{ji} = 1 \qquad \forall v_i \in V \cup \{v_{vir}, v_{0'}\} \text{ et } v_i \neq v_j \qquad (14)$$

$$\sum_{v_j \in W_h} y_{ij}^h = 1 \qquad \forall h \in \{1,...,H\}, \forall v_i \in W_h \cup \{v_{vir}\} \text{ et } v_i \neq v_j \tag{15}$$

$$\sum_{v_j \in W_h} y_{ji}^h = 1 \qquad \forall h \in \{1,...,H\}, \forall v_i \in W_h \cup \{v_{vir}\} \text{ et } v_i \neq v_j \tag{16}$$

$$u_i - u_j + (n'-1)x_{ij} + (n'-3)x_{ji} \leq (n'-2) \qquad \forall v_i, v_j \in V \cup v_{0'},\ v_i \neq v_j \tag{17}$$

$$u_i - u_j + (n'-1)\,y_{ij}^h + (n'-a_h - b_h - 1)\,y_{ji}^h \leq (n'-a_h - 1) \quad \forall h \in \{1,...,H\}, \forall v_i, v_j \in W_h \text{ et } v_i \neq v_j \tag{21}$$

$$u_i - u_j + (2n'-a_h - 2)\,y_{vir}^h + (2n'-a_h - 2)\,y_{j\,\mathrm{vir}}^h \leq (3n'-2a_h + b_h - 3) \quad \forall h \in \{1,...,H\}, \forall v_i, v_j \in W_h \text{ et } v_i \neq v_j \tag{22}$$

$$x_{0vir} + x_{vir0'} = 2 \tag{19}$$

$$x_{ij} \in \{0,1\} \qquad \forall (v_i, v_j) \in A \tag{5}$$

$$y_{ij} \in \{0,1\} \qquad \forall (v_i, v_j) \in C_h \tag{12}$$

$$1 \leq u_i \leq n'-1 \qquad \forall v_i \in V \tag{20}$$

La contrainte (21) a la même signification que la contrainte (18) mais nous avons fait quelques changements pour enlever la contradiction rencontrer par la deuxième formulation et nous avons ajouté la contrainte (22).

Montrons que la contrainte (21) est valide :

$u_i - u_j + a y_{ij}^h + b y_{ji}^h \leq n' - (a_h + 1)$

* *Si* $y_{ij}^h = 1$ et $y_{ji}^h = 0$

$u_i - u_j + a \leq n' - (a_h + 1)$

$u_i - u_j \leq n' - a_h - 1 - a$

on a $-b_h \leq u_i - u_j \leq -a_h$

$\Rightarrow a = n' - 1$

* *Si* $y_{ij}^h = 0$ et $y_{ji}^h = 1$

$u_i - u_j + b \le n' - (a_h + 1)$

$u_i - u_j \le n' - a_h - 1 - b$

on a $b_h \ge u_i - u_j \ge a_h$

$\Rightarrow b = n' - a_h - b_h - 1$

Donc : $u_i - u_j + (n-1)\, y_{ij}^h + (n - a_h - b_h - 1)\, y_{ji}^h \le n - a_h - 1.$ □

Montrons que la contrainte (22) est valide :

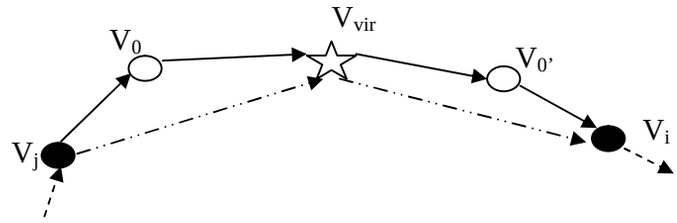


Figure 2.7 Sommets adjacents au sommet virtuel

$n' + u_i - u_j \le a(\, y_{vir\ i}^h + y_{j\ vir}^h) + b$

si $y_{j\ vir}^h = y_{vir\ i}^h = 1$: $a_h + 1 \le n' + u_i - u_j \le b_h + 1$

sinon : $a_h + 1 \le n' + u_i - u_j \le 2n' - a_h + b_h - 1$

$$\left.\begin{matrix} 2a + b = 1 + b_h \\ a + b = 2n' - a_h + b_h - 1 \end{matrix}\right\} \Rightarrow \left\{\begin{matrix} a = -2n' + a_h + 2 \\ b = 4n' - 2a_h + b_h - 3 \end{matrix}\right\}$$

$n' + u_i - u_j \le (-2n' + a_h + 2)(y_{vir\ i}^h + y_{j\ vir}^h) + (4n' - 2a_h + b_h - 3)$

Vérification :

$y_{j\ vir}^h = y_{vir\ i}^h = 1 \Rightarrow n' + u_i - u_j \le b_h + 1$ Θ

Soit $y_{j\ vir}^h = 1$, *soit* $y_{vir\ i}^h = 1 \Rightarrow n' + u_i - u_j \le 2n' - a_h + b_h - 1$ Θ

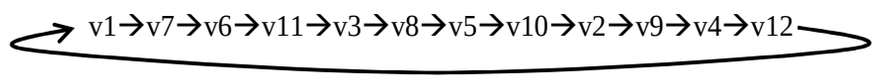
$y_{j\ vir}^h = y_{vir\ i}^h = 0 \Rightarrow n' + u_i - u_j \le 4n' - 2a_h + b_h - 3$

il faut que $4n' - 2a_h + b_h - 3 \ge 2n' - a_h - 1 + b_h \Leftrightarrow 2n' - a_h - 2 \ge 0$ Θ

$n' + u_i - u_j \le (-2n' + a_h + 2)(\, y_{vir\ i}^h + y_{j\ vir}^h) + (4n' - 2a_h + b_h - 3) \Leftrightarrow$

$u_i - u_j + (2n' - a_h - 2)\, y_{vir\ i}^h + (2n' - a_h - 2)\, y_{j\ vir}^h \le (3n' - 2a_h + b_h - 3)$ □

Lors du test de la formulation avec CPLEX sur l'exemple de 4 couleurs et de 3 sommets pour chacune traité précédemment, la solution trouvée est la suivante :



Nous remarquons que tous les sommets respectent-leur limites a et b .

La dernière modélisation du PVCM a été testée sur différentes instances de problèmes. Les résultats numériques sont détaillés dans le quatrième chapitre.

Ceci termine cette partie, nous passons maintenant au chapitre suivant où nous présenterons des heuristiques que nous avons développé pour la résolution du PVCM.

Chapitre 3 : Heuristiques pour le PVCM

Ayant décrit le modèle sous forme de programme linéaire en nombres entiers du PVCM, nous décrivons, dans ce chapitre, les algorithmes non-exacts que nous avons développé. Ces algorithmes s'inscrivent dans le cadre des heuristiques qui permettent de fournir une réponse approchée à ce problème. Nous commençons par exposer quelques heuristiques, déjà implémentées pour résoudre le PVC classique, que nous utiliserons comme sous-routines pour nos algorithmes. Ensuite, nous explicitons nos propres algorithmes de résolution du PVCM en justifiant les choix que nous avons adopté pour les mettre en oeuvre. Ces heuristiques sont développées pour le cas du PVCM non-orienté.

3.1 Méthodes heuristiques pour la résolution du PVC

3.1.1 Heuristique du plus proche voisin

Cette méthode est classée parmi les méthodes constructives gloutonnes. Elle a été proposée par Rosenkrantz , Stearns et Lewis [14]. Son principe est le suivant :

On choisit, arbitrairement, un nœud comme point de départ du cycle[1]. Puis, on détermine le nœud le plus proche de celui-ci et on l'insère dans le cycle. Tant qu'ils existent d'autres nœuds à insérer, on recommence la dernière étape tout en considérant le dernier nœud inséré. A la fin on relie le dernier nœud inséré au premier. Le complexité de cet algorithme est de l'ordre de $O(n^2)$.

Le pseudo code de cette heuristique peut être le suivant:

```
Sélectionner un nœud arbitraire s.
cycle ← s.
Tant que (ils existent des nœuds à insérer)
        distance ← infini
        s ← dernier nœud inséré.
        Pour (s' allant de 1 au nombre total des nœuds)
                Si ( distance(s,s')<distance  &&  s' n'est pas inséré)
                        Si ( nombre des sommets insérés dans cycle >1)
                                retirer s'.
                        insérer s' après s.
                        distance = distance(s,s').
                Fin Si
        Fin Pour
Fin Tant que
```

3.1.2 Les algorithmes d'insertion

Ces algorithmes appartiennent à la classe des heuristiques gloutonnes constructives. Ils ont été étudiés par Rosenkrantz, Stearns et Lewis [14], Stewart [15] et Norback et Love [16,17].

Ils opèrent de la manière suivante :

[1] Cycle dans le cas du PVC non-orienté

On part d'un cycle réduit à quelques nœuds, par exemple un cycle de deux nœuds, puis on sélectionne un nœud hors du cycle que l'on insère entre deux nœuds du cycle. Ainsi, on insère tous les nœuds jusqu'à obtenir un cycle qui contient tous les nœuds. Le choix du nœud à insérer peut être fait suivant nombreux critères :

- *Plus proche (resp. plus lointaine) insertion :* On peut insérer le nœud dont la distance minimale à un nœud du tour est minimale (resp. maximal).
- *Insertion au hasard :* On peut insérer un nœud choisi au hasard.
- *Insertion de coût minimal (resp. maximal) :* On peut insérer le nœud qui fera le moins (resp. le plus) augmenter la longueur du cycle partiel.

3.1.3 Méthode d'optimisation locale *k-opt*

Une fois qu'une solution initiale est obtenue par une heuristique gloutonne ou autre, celle-ci peut généralement être améliorer par une série de transformations locales effectuées par la méthode *k-opt*. Cette dernière a été proposée par Lin [18] et consiste à supprimer *k* arrêtes du cycle, ce qui a pour effet de couper celui-ci en *k* chaînes[2] disjointes et à recomposer un autre cycle en reconnectant ces chaînes d'une autre manière.

Ces transformations nécessitent parfois le changement du sens de parcours de certaines chaînes. En particulier, *2-opt* parcourt toujours une des deux chaînes à rebours et *3-opt* comporte quatre possibilités de transformation dont une seule ne conduit à parcourir aucune chaîne au sens inverse. Changer l'orientation d'une chaîne ne pose pas de problèmes dans le cas du PVC symétrique pur, mais, dans le cas de PVC asymétrique, la longueur du chemin[3] dépend du sens du parcours.

3.1.4 Méthode composite GENIUS

Une hybridation d'algorithme de construction et d'insertion GENI (GENeralized Insertion) et d'une post-optimisation US (Unstringing, Stringing) produit une excellente heuristique nommée GENIUS introduite par (Gendreau, Hertz et Laporte) [9]. Décrivons la procédure de construction et d'insertion GENI et ensuite la post-optimisation US.

GENI : La construction se fait à partir de trois nœuds choisis aléatoirement à partir desquels sont insérés d'autres nœuds pour former un cycle hamiltonien. L'insertion d'un nœud *x* se fait seulement entre deux nœuds de son voisinage $V_p(x)$ défini comme étant les *p* nœuds déjà

[2] Séquences de nœuds dans le cas d'un cycle (PVC non-orienté).
[3] Séquences de sommets dans le cas d'un circuit (PVC orienté).

insérés les plus proches de *x*. Le paramètre *p*, oscillant généralement entre 3 et 20 selon la taille du problème, permet de limiter la recherche et ainsi de contrôler la complexité de l'algorithme. Supposons une insertion du nœud *x* entre les nœuds $i, j \in V_p(x)$ et que $k \in V_p(i+1)$ soit sur la chaîne de *j* à *i* et que $\ell \in V_p(j+1)$ soit sur la chaîne de *i* à *j*. L'insertion peut se produire de deux façons (voir figure 3.1 et 3.2) :

- ➲ Insertion de type I : Soient $k \neq i$ et $k \neq j$. L'insertion de x dans un cycle entraîne l'élimination des arêtes *(i,i+1)* , *(j,j+1)* et *(k,k+1)* qui sont remplacés par *(i,x), (x,j), (i+1,k)* et *(j+1,k+1).*
- ➲ Insertion de type II : Soient $k \neq j$ et $k \neq j+1$, $\ell \neq i$ et $\ell \neq i+1$. L'insertion de x dans un cycle entraîne l'élimination des arêtes *(i,i+1), (ℓ-1,ℓ), (j,j+1)* et *(k-1,k)* qui sont remplacés par *(i,x), (x,j), (ℓ,j+1), (k-1,ℓ-1)* et *(i+1,k)*..

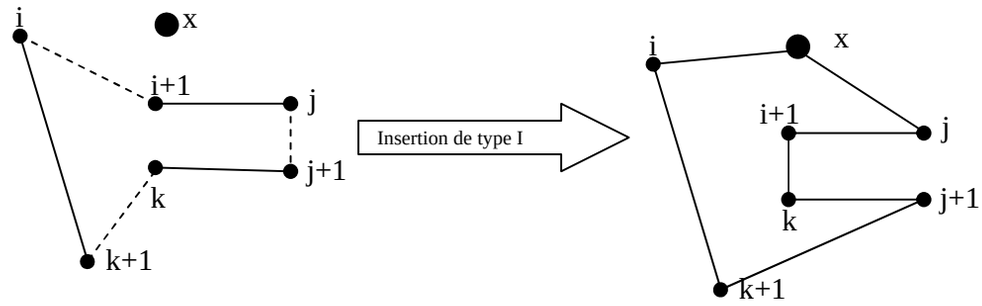


Figure 3.1 : Insertion de type I

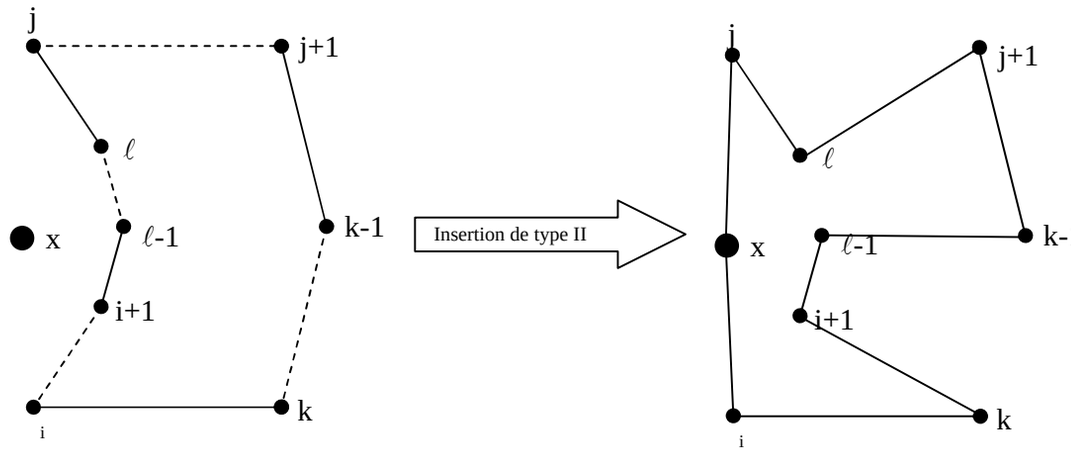


Figure 3.2 : Insertion de type II

On remarque que GENI permet l'insertion entre deux nœuds non-consécutifs pour ensuite ré-optimiser la chaîne entre ces deux nœuds. Ainsi, une insertion standard suivie d'un *3-opt* est équivalent à une insertion de type I.

US : La procédure de post-optimisation US (Unstringing-Stringing) consiste à considérer le retrait de chaque nœud pour tenter leur insertion dans le but de produire une tournée de moindre coût. Notons que le retrait (unstringing) est symétrique aux insertions (stringing) et peut être effectué de deux façons :

- ➲ Retrait de type I : Soient $j \in V_p(i+1)$ et $k \in V_p(i-1)$ un nœud sur le chemin de *i+1* à *j-1*. Le retrait de i entraîne le retrait des arêtes *(i-1,i) ,(i,i+1), (j,j+1)* et *(k,k+1)* qui sont remplacés par *(i-1,k), (i+1,j), (k+1,j+1).*
- ➲ Retrait de type II : Soient $j \in V_p(i+1)$ et $k \in V_p(i-1)$ un nœud sur la chaîne de *j+1* à *i-2* et que $l \in V_p(k+1)$ sur le chaîne de *j* à *k-1*. Le retrait de i entraîne le retrait des arêtes *(i-1,i) ,(i,i+1), (j-1,j) (l,l+1)* et *(k,k+1)* qui sont remplacés par *(i-1,k), (l+1,j-1), (i+1,j), (l,k+1).*

L'algorithme GENIUS applique les procédures d'insertion du nœud retiré *i* au meilleur cycle *k* en considérant à chaque fois les deux types d'insertions. Rappelons qu'il est possible de réduire la complexité en utilisant la notion de *p*-voisinage qui considère seulement les *p* plus proches nœuds.

3.2 Heuristiques pour la résolution du PVCM

Dans cette partie, nous décrivons les algorithmes des heuristiques que nous avons proposé pour résoudre le PVCM non-orienté. Pour bien illustrer le fonctionnement de chaque heuristique, nous prenons une instance du problème, à douze nœuds et trois couleurs, comme référence. Cette instance est détaillée dans ce qui suit :

- Couleur 1 noir : 6 nœuds {v1,v2,v3,v4,v5,v6} $a_1=2$ $b_1=2$.
- Couleur 2 blanche : 2 nœuds {v7,v8} $a_2=4$ $b_2=8$.
- Couleur 3 grise : 4 nœuds {v9,v10,v11,v12} $a_3=2$ $b_3=4$.

3.2.1 Algorithme I

L'idée du premier algorithme pour la résolution du PVCM a été inspirée de l'algorithme appliqué au problème du voyageur de commerce à deux couleurs BWTSP [3]. Il est composé de

deux phases. La première phase consiste à construire, avec les nœuds de couleur *h*, un sous-cycle de cette couleur[4]. Pour chaque couleur, on sélectionne ces nœuds et on applique GENIUS.

La deuxième phase est basée sur l'insertion. Tout d'abord, on tri les sous-cycles, produits par la première phase, en ordre décroissant suivant leurs cardinalités. On sélectionne le premier cycle, puis on commence à insérer les nœuds du cycle suivant tout en essayant de ne pas dépasser la limite b du premier cycle. L'insertion se fait de la manière suivante: on commence à insérer les nœuds entre ceux qui sont adjacents et de même couleurs. Une fois les nœuds du deuxième cycle sont insérés, on prend le cycle qui suit et on procède de la même façon mais en respectant les limite b des couleurs des cycles déjà insérer. S'il n'existe pas des sommets adjacents, on débute par le premier nœud de couleur insérée juste avant et on insère juste après lui, ensuite on passe au suivant. On procède de cette façon jusqu'on n'a rien à insérer.

Nous remarquons que cette heuristique ne tient pas compte de la limite minimum. Elle sera satisfaite à la fin de la construction du cycle globale grâce au tri des sous cycles initiaux dans l'ordre décroissant de leur cardinalité.

Le pseudo code de cet algorithme peut être le suivant :

Algorithme 1

Phase1

Trier les couleurs en ordre décroissant, suivant leur ordre d'apparition, dans un tableau **coul_triées**
Pour (**coul** allant de 1 au nombre total des couleurs triées)
    Appliquer GENIUS sur l'ensemble des nœuds de couleur **coul** et récupérer le cycle.
Fin Pour

Phase2

cycle_total ← cycle de la première couleur
couleurs_inserées ← 1
Pour (**coul** allant de 2 au nombre total des couleurs triées)
    couleurs_inserées ++
    Pour (**som** allant de 1 au nombre de nœuds du cycle de couleur **coul**)
        Pour (i allant de **couleurs_inserées** jusqu'à 1)
            Si **cycle_total** contient des nœuds s et s' adjacents de couleurs **coul_triés**[i]
                inserer **som** entre s et s'
                aller à **fin_insertion**
            Fin Si
        Fin Pour
        insérer **som** après le premier nœud, rencontré dans **cycle_total**, de couleur **coul-1**
        →**fin_insertion**
    Fin Pour
Fin Pour

[4] Un sous-cycle de couleur h est le cycle composé des nœuds de couleur h.

La figure suivante illustre la manière de construire le cycle par cet algorithme quand on l'applique à l'exemple de référence mentionné au début de cette section.

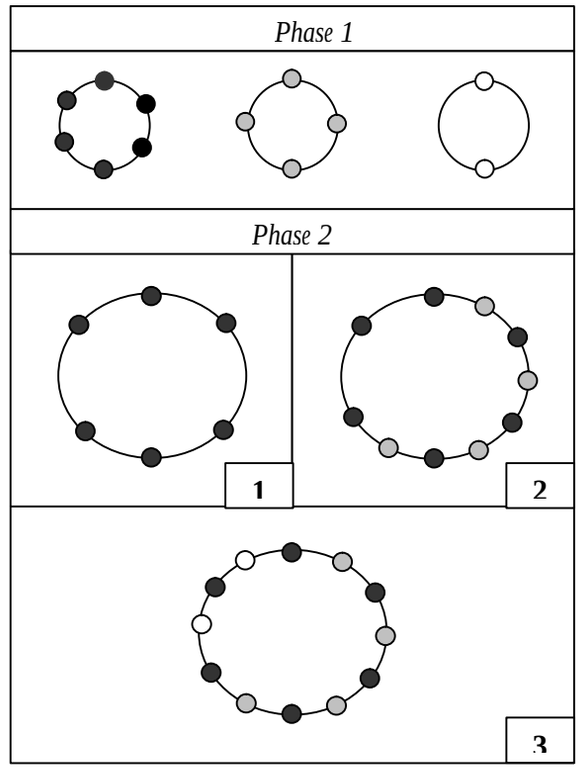


Figure 3.3 : Construction du cycle par l'algorithme 1

Nous remarquons que le cycle généré par cet algorithme n'est pas admissible. Les couleur blanche et grise ne respectent pas leur limite minimum a . Cela peut être expliquer par le fait que cette heuristique possède un aspect glouton et le non-respect des tous les paramètres de chaque couleur lors de l'insertion des nœuds. Cette heuristique peut résoudre seulement des problèmes où toutes les couleurs possèdent la même cardinalité.

Il faut tenir compte, lors de la construction du cycle, des contraintes d'espacement de toutes les couleurs insérées. Cela va être pris en compte dans les heuristiques qui suivent.

3.2.2 Algorithme II

Cet algorithme est composé de trois phases. La première est une phase de construction d'un cycle composé de tous les nœuds sans tenir compte des contraintes d'espacement des couleurs. Ceci revient donc à construire un cycle de PVC classique. Cette construction est effectuée par la méthode constructive du plus proche voisin.

La deuxième phase est celle de correction séquentielle des contraintes d'espacement pour les différentes couleurs. Son principe est le suivant :

On sélectionne une couleur *h*, puis on fait des déplacements des nœuds de cette couleur de telle façon à respecter leurs limites a_h et b_h. Ensuite, on choisit une autre couleur et on fait les déplacements requit, en suivant le cycle dans un sens, pour satisfaire ces limites tout en conservant celles des nœuds de couleurs déjà corrigées. S'il n'est pas possible de corrigé la couleur courante, suivant un sens, sans conserver les limites des couleurs déjà corrigées, on tente dans le sens contraire de celui choisi initialement. On procède de cette façon jusqu'à ce qu'il n'y ait plus de couleurs à sélectionnées. La sélection des couleurs se fait suivant une séquence de couleur choisie au début. Si une séquence n'aboutit pas à une solution admissible, on considère une autre séquence des couleurs et on tente de nouveau la correction. Cette phase peut être gourmande en temps.

La troisième phase est la procédure d'amélioration. Elle est appliquée sur chaque ensemble de nœuds de même couleur. Elle est définie comme suit : On considère les couleurs une à une. Pour chaque couleur, on sélectionne un nœud et on essaye de le permuter avec un autre de même couleur afin de diminuer la longueur du cycle. Si c'est le cas, on garde la permutation sinon on l'annule. Cette tentative de permutation est appliquée à chaque nœud de couleur *h* avec tous les autres nœuds de même couleur. Si on a considéré tous les nœuds de la couleur sélectionnée, on passe à la suivante jusqu'à la fin des couleurs à sélectionner.

Cet algorithme peut être résumé dans le pseudo-code suivant :

Algorithme 2

Phase1 : Solution initiale

Appliquer l'heuristique du plus proche voisin sur l'ensemble de tous les nœuds (c.f. 3.1.1)

Phase2 : Admissibilité

```
→nouvelle_combinaison
Combine(couleurs) /*nouvel ordre des couleurs*/
Pour (coul allant de 1 au nombre total des couleurs)
        s ← début du cycle.
        Tant que ( couleur(s) != coul)
                s ← suivant (s)/* On sélectionne le premier nœud de couleur courante*/
        Fin Tant que
        compteur ← 1/*pour compter les nœuds insérés de couleur courante*/
        Tant que  (compteur < nombre des nœuds de coul)
                s ← suivant(s).
                pas++/*compteur d'arêtes qui sépare deux nœuds successifs de même couleur */
                Tant que (couleur(s) != coul)
                        s ← suivant(s)/* on sélectionne le nœud de couleur courante qui suive le premier*/
                        pas++
                Fin Tant que
                Si ( pas >beta )
```

```
                    /*reculer le dernier nœud sectionné de (beta-pas) arêtes*/
                    Pour (i allant de 1 à (beta - pas))
                        permuter  les couleurs et les indices entre s et précédent(s)
                        s=precedent(s)
                        Si (il y à une couleur violée)
                            aller à sens_contraire.
                        Fin Si
                    Fin Pour.
                Fin Si
                Si ( pas < alpha)
                    /*avancer le dernier nœud sectionné de (beta-pas) arêtes*/
                    Pour (i allant de 1 à (pas - alpha))
                        permuter les couleurs et les indices entre s et suivant(s).
                        s=suivant(s).
                    Fin Pour
                Fin Si
                compteur++.
                s ← suivant(s)
            Fin Tant que
Fin Pour

→Sens_contraire /*dans le cas où on ne pourrait  pas corriger une couleur dans un sens de déplacement*/
Réinitialiser le cycle généré par la phase 1

Pour (coul allant de 1 au nombre total des couleurs)
        s ← début du cycle.
        Tant que ( couleur(s) != coul)
            s ← précédent (s)
        Fin Tant que
        compteur ← 1.
        Tant que  (compteur < nombre des nœuds de coul)
            s ← précédent (s).
            pas++.
            Tant que (couleur(s) != coul)
                s ← précédent (s) )/* on sélectionne le nœud de couleur courante qui précède le premier*/
                pas++
            Fin Tant que
            Si ( pas >bata )
                Pour (i allant de 1 à (beta - pas))
                    permuter  les couleurs et les indices entre s et précédent(s)
                    s=precedent(s)
                    Si (il y à une couleur violée)
                        aller à nouvelle_combinaison
                    Fin Si
                Fin Pour.
            Fin Si
            Si ( pas < alpha)
                Pour (i allant de 1 à (pas - alpha))
                    permuter les couleurs et les indices entre s et précédent(s).
                    s= précédent (s)
                    Si (il y à une couleur violée)
                        aller à nouvelle_combinaison
                    Fin Si
                Fin Pour
            Fin Si
            compteur++.
            s ← précédent (s)
        Fin Tant que
Fin Pour
```

Phase3 : Amélioration

```
Pour (coul allant de 1 au nombre total des couleurs)
        s← debut(cycle).
        Pour ( i  allant de 1 au (nombre total des nœuds de couleur coul))
                Tant que (couleur(s) != coul)
                        s ← suivant(s).
                Fin Tant que
                s' ← suivant(s)
                Pour (j allant de 1 au  (nombre total des nœuds de couleur coul - 1))
                        Tant que (couleur(s') != coul)
                                s' ← suivant(s').
                        Fin Tant que
                        permuter s et s'.
                        Si (nouvelle longueur du cycle >= ancienne longueur)
                                permuter s et s'
                        Sinon
                                ancienne longueur ← nouvelle longueur.
                        Fin si
                        s' ← suivant(s').
                Fin Pour
                s ← suivant(s).
        Fin Pour
Fin Pour
```

Pour illustrer le déroulement de cet algorithme nous prenons l'instance de référence considérée au début de cette section, et nous appliquons l'algorithme.

Phase 1 :

La première phase génère le cycle illustré par la figure 3.4.

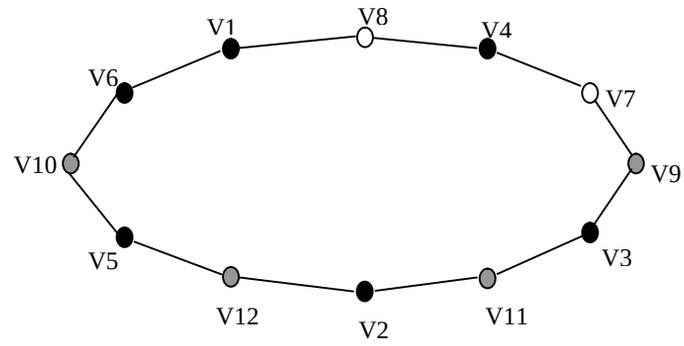


Figure 3.4 : Cycle après la première phase de l'algorithme 2

Nous remarquons que le cycle généré par la première phase n'est pas une solution du PVCM puisque aucune couleur ne respecte ces limites a et b. C'est normal puisque lors de cette phase les contraintes d'espacement sont relaxées.

Phase 2 :

- On combine les couleurs dans un ordre par exemple {noir, blanc, gris}.
- On commence par la première couleur (noire) et du premier nœud *v1* de cette couleur dans le cycle (figure 3.4) et en passe au suivant de même couleur *v4* (dans le sens horaire). On s'aperçoit que ces deux nœuds respectent $a_1 = 2$ et $b_1 = 2$. Donc on passe au nœud de couleur noir qui suive *v4* qui est *v3*. On s'aperçoit qu'il y a 3 arêtes qui relient ces deux nœuds. Donc on permute *v3* avec le nœud qui le précède *v9* et on passe au nœud de couleur noir qui suive *v3* en tenant compte de sa nouvelle place et on vérifie les limites entre les deux. On continu de cette façon jusqu'à ce que tous les nœuds de couleur noire respectent les contraintes d'espacement (voir figure 3.5).

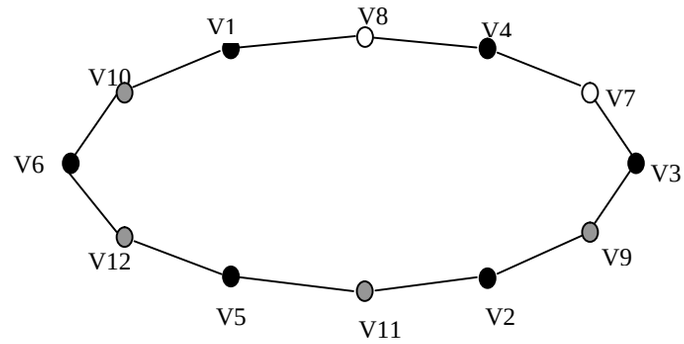


Figure 3.5 : Cycle après la correction de la couleur noire

- On procède de la même façon que les nœuds noirs avec les nœuds blancs, mais il faut vérifier que l'on conserve l'admissibilité par rapport aux contraintes d'espacement pour la couleur noire lors de chaque changement d'un nœud blanc. Si ce n'est pas le cas on essaye une autre fois mais en parcourant le cycle dans son sens inverse. Si on se bloque de nouveau, on initialise le cycle comme il a été généré par la première phase et on essaye avec une autre combinaison de couleur par exemple {noir, gris, blanc}.
- Si les contraintes d'espacement pour la couleur blanche sont satisfaites, on passe à la troisième couleur (grise) et on fait de même que la couleur blanche et en tenant compte du respect des contraintes d'espacement de cette dernière et pour la couleur noire.

Le cycle généré à la fin de cette phase est illustré par la figure 3.6.

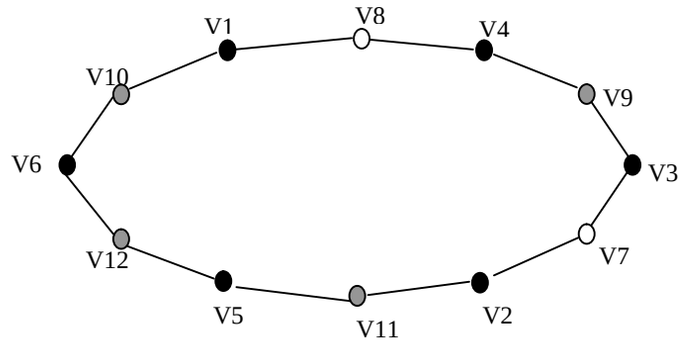


Figure 3.6 : Cycle après la correction de toutes les couleurs

Phase 3 :

- Après avoir corrigé toutes les couleurs, on passe à la phase d'amélioration. On sélectionne par exemple le nœud *v1* de couleur noire et on le permute avec le nœud *v4* de même couleur. Si cette permutation diminue le coût du cycle on conserve cette permutation, sinon on l'annule. On essaye cette permutation du nœud *v1* avec tous les nœuds restants de couleur noire, puis on passe au nœud *v2* et on fait la même chose.
- Lorsqu'on termine l'examen de tous les nœuds de couleur noire, on passe à la couleur blanche et on procède de la même façon, puis la couleur grise.

Le cycle final généré par le deuxième algorithme est illustré par figure 3.7. Nous remarquons qu'il n'y a pas de changement effectué par la phase d'amélioration pour cette instance.

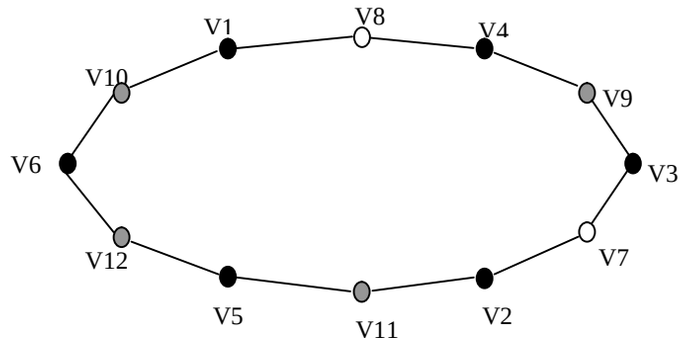


Figure 3.7 : Cycle généré par le deuxième algorithme

3.2.3 Algorithme III:

Cet algorithme a le même principe que le précédent. La différence entre les deux réside dans le fait que celui-ci utilise GENIUS, lors de la première phase, pour créer un cycle

comportant tous les nœuds et qui ne tient pas compte des contraintes d'espacement. En plus, il ne comporte pas une phase d'amélioration. Il est composé seulement de deux phases, la première consiste à construire un cycle total avec GENIUS sans tenir compte des contraintes d'espacement. Ces contraintes seront satisfaites lors de la deuxième phase qui consiste à faire migrer le cycle vers un état qui les respecte.
La représentation en pseudo-code de l'algorithme est comme suit :

Algorithme 3

Phase1 : Solution initiale

Appliquer GENIUS sur l'ensemble de tous les nœuds et récupérer le cycle.

Phase2 : Admissibilité

Voir phase 2, algorithme 2 (c.f 3.2.2).

Nous pouvons classer cet algorithme dans une classe des heuristiques destructives. En effet, après la première phase qui génère un cycle par GENUIS généralement proche de l'optimum d'un PVC classique, on commence à augmenter sa longueur à cause des corrections effectuées par la deuxième phase. Il y a des cas où le cycle obtenu juste après la première phase respecte les conditions d'espacement des couleurs et on n'aura pas besoin de la phase de correction. Nous ne pouvons pas considérer le cycle générer par GENIUS comme une borne inférieure puisque GENIUS ne donne pas toujours la valeur optimale.

3.2.4 Algorithme IV

Dans certains cas, la deuxième phase de l'algorithme précédent dégrade énormément la qualité du cycle généré par GENIUS. D'où, nous avons ajouté une troisième phase qui consiste à améliorer la solution obtenue par le troisième algorithme. Ainsi, la différence entre cet algorithme et celui qui le précède c'est que le présent comporte une phase d'amélioration utilisant la technique *2-opt*.

Pour le PVC classique, la méthode d'amélioration *2-opt* consiste, comme il a été déjà mentionné au début de ce chapitre, à couper le cycle en deux chaînes en enlevant deux arêtes au choix et de le reconnecter en permutant deux extrémités des deux chaînes. Si cette modification fait diminuer la longueur du cycle, on passe à deux autres chaînes, sinon on rétablit le cycle d'avant. Il est connu que lorsqu'on permute les extrémités des deux chaînes d'un PVC, il aura lieu un changement de sens de l'une des deux. Ce changement est autorisé s'il est question d'améliorer le coût du cycle.

Dans le cas du PVCM, si la procédure de 2-opt est appliquer sur la totalité du cycle, il y a un risque de violer les contraintes d'espacement des couleurs. Nous avons donc choisi d'appliquer cette procédure sur chaque sous-cycle associé à la couleur *h*, à condition que sa cardinalité soit en minimum égale à quatre. Cette condition élimine la violation des contraintes rétablis par la deuxième phase. En effet, lorsqu'on applique cette procédure d'amélioration sur chaque sous-cycle de couleur *h* tout seul, les permutations vont être faites sur des nœuds de même couleur d'où on ne risque pas de changer l'ordre des couleurs déjà établit par la deuxième phase.

Cette phase d'amélioration est avantagé par rapport à celle du deuxième algorithme. En effet, la procédure 2-opt entraîne plusieurs permutations de noeuds lors d'une seule étape, par contre la procédure d'amélioration, utilisé au deuxième algorithme, fait une seul permutation lors d'une seule étape.

Dans le cas où la cardinalité d'un sous-cycle serait inférieure à quatre, on a recours à la méthode d'amélioration déjà mentionnée dans le deuxième algorithme

Pour bien illustrer le comportement de cette phase d'amélioration, nous pouvons appliquer la *2-opt* sur le cycle généré par les deux premières phases de l'algorithme de l'exemple de référence de 12 nœuds. (voir figures 3.8 et 3.9). La transformation est effectuée sur le sous-cycle de couleurs noires.

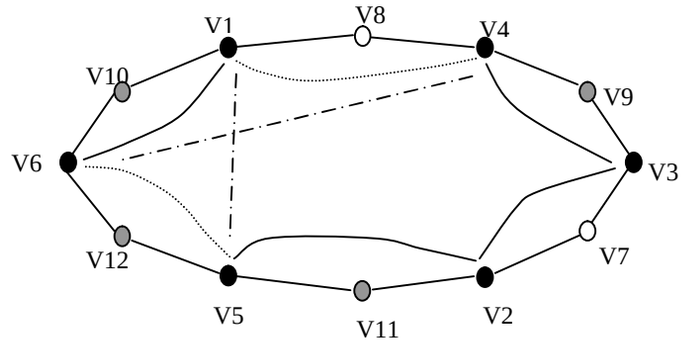


Figure 3.8 : Cycle avant un changement 2-opt sur une couleur

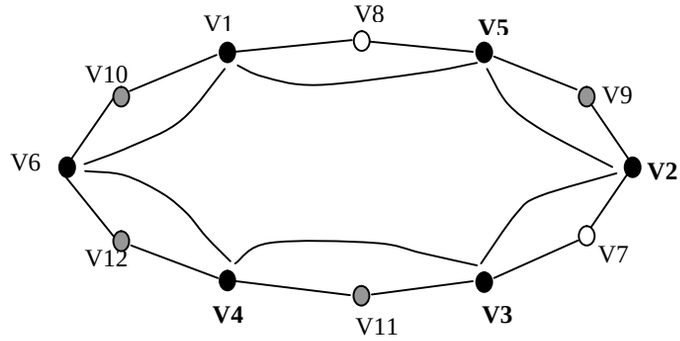


Figure 3.9 : Cycle après un changement 2-opt sur une couleur

La figure 3.8 et 3.9 illustrent respectivement le cycle avant et après les changements effectués par une transformation *2-opt* appliquée sur le sous-cycle de couleur noire.
Les permutations effectuées par *2-opt* sont comme suit :

- Les deux arêtes pointillées (*v1*,*v4*) et (*v5*,*v6*) de la figure 3.8 seront supprimés.
- Les deux arêtes discontinues (*v1*,*v5*) et (*v4*,*v6*) de la figure 3.8 seront ajoutés.
- La chaîne composée par (*v4*,*v3*,*v2*,*v5*) de la figure 3.8 va changer de sens. Ceci peut changer la longueur du cycle total lorsqu'on a entre deux nœuds de même couleur plus qu'un nœud de couleur différente.
- Le sous-cycle noir illustré par la figure 3.9 présente lui-même après la permutation. Les nœuds en gras sont ceux qui ont changé d'endroit.
- Si le coût du cycle total composé par tous les nœuds a diminué, on garde les transformations effectuées, sinon on les annule.

Le pseudo code de cet algorithme est le suivant :

Algorithme 4

Phase1 : Solution initiale

Appliquer GENIUS sur l'ensemble de tous les nœuds et récupérer le cycle.

Phase2 : Admissibilité

La même que la deuxième phase du deuxième algorithme.

Phase3 : Amélioration

Pour (**coul** allant de 1 au nombre total des couleurs)
    Amélioration *2-opt* du cycle de couleur **coul**
Fin Pour

Les algorithmes de résolution approchée du PVCM, développés dans ce chapitre, ont été testés sur des différentes instances. Les détails de ces tests et leurs comparaisons par rapport à la résolution exacte et entre eux-mêmes seront présentés dans le chapitre suivant.

Chapitre 4 : Expériences numériques

Ce chapitre est consacré aux expériences numériques pour le PVCM. Nous comparons les solutions données par les quatre algorithmes, implémentés dans le chapitre précédent, à celles données par la résolution d'un logiciel commercial de la formulation établie au deuxième chapitre.
Le choix des instances du problème pour les tests est délicat. En effet, si les coefficients a et b de chaque couleur sont mal choisis le problème ne possèdera pas de solutions admissibles.
Nous avons établi une méthode pour générer des instances du problème qui admettent des solutions réalisables. Nous commençons par expliciter cette méthode que nous avons utilisé pour générer nos instances.

4.1 Génération des instances du PVCM :

La difficulté qui découle de la génération des instances du PVCM réside dans le choix des limites a et b de chaque couleur. Un choix aléatoire de ces paramètres peut mener à des instances qui ne possèdent pas de solutions admissibles.
Nous utilisons les notations suivantes :

- n_i : le nombre de sommets d'une couleur *i*.
- n : le nombre total des sommets.
- a_i et b_i : sont respectivement a et b de la couleur *i*.
- H : c'est le nombre total des couleurs.

Nous pouvons générer nos problèmes de deux façons qui sont les suivantes:

- Nous fixons les limites a_i et b_i de chaque couleur i et le nombre total des sommets n et nous déterminons le nombre de sommets n_i de chaque couleur.
- Nous fixons le nombre de sommets n_i de chaque couleur i puis nous déterminons leurs limites a_i et b_i.

Nous faisons référence au cycle illustré par la figure 4.1 pour bien expliquer la démarche.

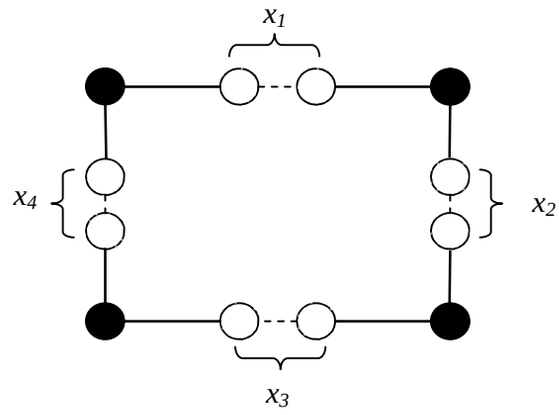


Figure 4.1 : nombre des arcs entre deux sommets de même couleur

4.1.1 Première méthode : Détermination des n_i pour a_i, b_i fixés

Le cycle ci-dessus comporte n_i sommets de couleur *i*. Entre chaque paire de sommets adjacents de couleur *i*, il existe x_k sommets de couleurs différentes, avec $k \in \{1,...,n_i\}$ (voir l'exemple de la couleur noir illustré par la figure 4.1). x_k est compris entre le minimum $(a_i - 1)$ et le maximum $(b_i - 1)$ fixés auparavant. Ainsi, la valeur de n_i est borné par :

$$\left\lceil \frac{n}{b_i} \right\rceil \leq n_i \leq \left\lfloor \frac{n}{a_i} \right\rfloor$$

Preuve :

$(a_i - 1) \leq x_k \leq (b_i - 1)$

$n_i(a_i - 1) \leq \sum_{k=1}^{n_i} x_k \leq n_i(b_i - 1)$

on a $\sum_{k=1}^{n_i} x_k = n - n_i$

donc $n_i(a_i - 1) \leq n - n_i \leq n_i(b_i - 1)$

$(a_i - 1) \leq \frac{n - n_i}{n_i} \leq (b_i - 1)$

$a_i \leq \frac{n}{n_i} \leq b_i$

$\frac{n}{b_i} \leq n_i \leq \frac{n}{a_i}$

$=> \left\lceil \frac{n}{b_i} \right\rceil \leq n_i \leq \left\lfloor \frac{n}{a_i} \right\rfloor$ □

La difficulté de cette méthode réside dans le choix du nombre total des sommets. En effet, le choix d'un n trop petit peut être insuffisant pour satisfaire les besoins des sommets de chaque couleur. Par exemple, si $n < \sum_{i=1}^{H} (b_i - 1)$ et on choisi, pour chaque couleur *i*, $n_i = (b_i - 1)$ on se trouve dans la contradiction $n = \sum_{i=1}^{H} (b_i - 1) < \sum_{i=1}^{H} (b_i - 1)$!!.

De même que le choix d'un n très grand pourrait entraîner la non-affectation de sommets à des couleurs. En effet, le choix d'un n trop grand peut être insuffisant pour satisfaire les besoins des sommets de chaque couleur. Par exemple, si $n > \sum_{i=1}^{H} (b_i - 1)$ et on choisi, pour chaque couleur i, $n_i < (b_i - 1)$ on se trouve dans la contradiction $n = \sum_{i=1}^{H} n_i < \sum_{i=1}^{H} (b_i - 1) < n$!!.

4.1.2 Deuxième méthode : Détermination des a_i, b_i pour n_i fixés

Soit la couleur *i*, ce cycle comporte un nombre connu n_i de sommets *i*. Entre chaque paire de sommets adjacents de cette couleur il existe x_k sommets de couleurs différentes, avec $k \in \{1,...,n_i\}$. x_k est compris entre le minimum $(a_i - 1)$ et le maximum $(b_i - 1)$ déterminés par cette méthode. Les valeur de a_i et b_i seront déterminées par les deux encadrements suivant :

$$2 \leq a_i \leq \left\lfloor \frac{n}{n_i} \right\rfloor$$

$$\left\lceil \frac{n}{n_i} \right\rceil \leq b_i \leq n - a_i n_i + a_i$$

Preuve :

On a : $(a_i - 1) \leq \frac{n - n_i}{n_i} \leq (b_i - 1)$ (*c.f preuve de la première méthode*)

$a_i \leq \frac{n}{n_i} \leq b_i$

$\Rightarrow a_i \leq \left\lfloor \frac{n}{n_i} \right\rfloor$ et $b_i \geq \left\lceil \frac{n}{n_i} \right\rceil$

Entre deux sommets de même couleurs il doit y exister au moins un sommet de couleur différente, donc $a_i \geq 2$.

Entre deux sommets de même couleur on peut trouver en maximum $(n-n_i)-(n_i-1)$ sommets de couleurs différentes puisque entre les (n_i-1) autres sommets, de cette couleur, il doit y avoir au moins (n_i-1) sommets de couleur différente, donc $b_i \le n-2n_i+2$.

Lors du choix des a_i et b_i, si on fixe a_i la première, b_i est dépendant de a_i. Donc on peut diminuer la borne supérieure de b_i. En effet, entre deux sommets de même couleur on peut trouver en maximum $(n-n_i)-(a_i-1)(n_i-1)$ sommets de couleurs différentes puisque entre les (n_i-1) autres sommets, de cette couleur, il doit y avoir au moins $(a_i-1)(n_i-1)$ sommets de couleur différente, donc $b_i \le n-a_i n_i+a_i$. Nous remarquons que pour $a_i=2$ nous obtenons la première borne supérieure de b_i. □

Remarque :

Une façon pour vérifier si les valeurs de a_i et b_i sont bien choisit peut être la suivante :

$$\sum_{i=1}^{H}\frac{1}{b_i} \le 1 \le \sum_{i=1}^{H}\frac{1}{a_i}$$

Preuve :

On a : $n_i(a_i-1) \le n-n_i \le n_i(b_i-1)$ (*c.f preuve de la première méthode*)

$n_i a_i \le n \le n_i b_i$

$\frac{1}{b_i} \le \frac{n_i}{n} \le \frac{1}{a_i}$

$\sum_{i=1}^{H}\frac{1}{b_i} \le \sum_{i=1}^{H}\frac{n_i}{n} \le \sum_{i=1}^{H}\frac{1}{a_i}$

$=> \sum_{i=1}^{H}\frac{1}{b_i} \le 1 \le \sum_{i=1}^{H}\frac{1}{a_i}$

□

4.2 Expériences numériques

Les expériences du model et des algorithmes établis dans les chapitres précédents ont été effectués pour différentes instances. Le modèle a été résolu à l'aide du logiciel d'optimisation CPLEX qui utilise une méthode de résolution exacte de type « *Branch and Bound* ». Les heuristiques ont été aussi codés en C++ et compilé avec g++. Tous les tests ont été effectués sur un *Compaq AlphaServer DS20 bi-processor EV6/500 à 2 Go de mémoire.*

Nous avons testé des problèmes de n=6, 10, 16, 20, pour chacun nous avons généré des instances tout en variant le nombre des couleurs. La détermination des limites a et b a été effectuée par la deuxième méthode établie à la section précédente. Les valeurs numériques de nos tests sont reportées dans les tableaux suivants. Les entêtes des colonnes sont définies comme suit :

- ***n*** : nombre de sommets du graphe ;
- ***#coul*** : le nombre de couleurs du graphe ;
- ***#som/coul*** : le nombre de sommets de chaque couleur ;
- ***alpha*** : le nombre minimum d'arc qui peuvent joindre deux sommets successifs de la même couleur ;
- ***beta*** : le nombre maximum d'arc qui peuvent joindre deux sommets successifs de la même couleur ;
- ***Zex*** : le coût du cycle optimum trouvé par la méthode exacte ;
- ***RC*** : le coût du cycle optimum en relaxant la contrainte d'intégrité des variables en nombres entiers ;
- ***o.m*** : elle prend « oui » s'il y à une saturation de l'espace mémoire réservé par CPLEX à l'arbre de recherche de « *Branch and Bound* » ;
- ***T(s)*** : le temps en secondes mis par la méthode exacte ou par l'un des algorithmes pour résoudre une instance ;
- ***Gap*** : c'est le saut de dualité c'est-à-dire le pourcentage de le solution en nombre entier, trouvé par CPLEX, par rapport à l'optimum exact s'il y a une saturation de la mémoire ;
- ***Zh1*** : le coût du cycle trouvé par le premier algorithme ;
- ***Zh2*** : le coût du cycle trouvé par le deuxième algorithme ;
- ***Zh3*** : le coût du cycle trouvé par le troisième algorithme ;
- ***Zh4*** : le coût du cycle trouvé par le quatrième algorithme ;
- ***%Zex*** : c'est le pourcentage de la valeur de la solution, trouvée par un algorithme, par rapport à celle trouvée par la méthode exacte ;
- ***%RC*** : le pourcentage de la valeur de la solution, trouvé par un algorithme, par rapport à RC ;
- ***Moy(%Ze)*** : la moyenne arithmétique des pourcentages par rapport à l'optimum des instances d'une seul taille de graphe ;
- ***Moy(%RC)*** : la moyenne arithmétique des pourcentages par rapport à la valeur de la relaxation en continu des instances d'une seule taille de graphe ;
- ***Moy tot(%Ze)*** : la moyenne arithmétique des pourcentages par rapport à l'optimum de toutes les instances testées ;

- ***Moy tot(%RC)*** : la moyenne arithmétique des pourcentages par rapport à la valeur de la relaxation en continu de toutes les instances testées ;
- ***Moy tot**** : le pourcentage de nombre de fois par rapport au nombre des instances qu'une heuristique possède la meilleure solution.

4.2.1 Interprétations des résultats numériques de la méthode exacte

Nous interprétons dans cette partie les résultats numériques donnés par CPLEX lors des tests du model exacte sur les différentes instances.

Pour n=6 sommets (Tableau 4.1), la méthode exacte arrive à résoudre à l'optimum toutes les instances au bout d'un temps rapide. En ce qui concerne les instances de 10 sommets de deux et de trois couleurs nous remarquons que le temps a augmenté par rapport à celui de 6 sommets. Cela s'explique par le fait que nous avons augmenté le nombre total des sommets (Tableau 4.2). Pour quatre et cinq couleurs l'optimum est obtenu après un temps relativement raisonnable.

n	*#coul*	*#som/coul*	*alpha*	*beta*	*Zex*	*RC*	*o.m*	*T (s)*	*Gap*
6	2	3	2	2	1469	1461	***	0,033	***
		3	2	2					
	3	2	3	3	1592	1000,95	***	0,05	***
		2	3	3					
		2	3	3					

Tableau 4.1 : Résolution du modèle pour *n*=6

n	*#coul*	*#som/coul*	*alpha*	*beta*	*Zex*	*RC*	*o.m*	*T (s)*	*Gap*
10	2	5	2	2	2325	2251	***	1,35	***
		5	2	2					
	3	3	3	4	2069	1660,21	***	6,766	***
		4	2	3					
		3	3	4					
	4	2	5	5	2124	1460,95	***	13,76	***
		3	3	4					
		2	5	5					
		3	3	4					
	5	2	4	5	2186	1462,78	***	4,766	***
		2	4	5					
		2	4	5					
		2	4	5					
		2	4	5					

Tableau 4.2 : Résolution du modèle pour *n*=10

Nous constatons que pour les instances où les cardinalités des couleurs sont différentes, l'optimum est obtenu pendant un temps plus important que celles qui possèdent des cardinalité égales. Par exemple pour l'instance de n=10 avec 4 couleurs a mis plus de temps que l'instance n=10 avec 5 couleurs malgré que la dernière possède plus de couleurs (Tableau 4.2).

Pour n=16 (Tableau 4.3), nous avons pu résoudre à l'optimum seulement les deux instances de deux et trois couleurs durant un temps acceptable. Pour celles de quatre et cinq couleurs l'arbre de recherche de « Branch and Bound » a subi une saturation de la mémoire. Nous avons récupéré les meilleures solutions en nombres entiers trouvées par CPLEX qui sont présentées en *italique*. Les sauts de dualité des deux instances sont un peu grand. Les instances de six et sept couleurs n'ont été pas résolues suite à la saturation mémoire après un temps relativement important.

n	***#coul***	***#som/coul***	***alpha***	***beta***	***Zex***	***RC***	***o.m***	***T (s)***	***Gap***
16	2	8	2	2	3727	3691,61	***	36,15	***
		8	2	2					
	3	2	8	8	4641	4641	***	21,1	***
		8	2	2					
		6	2	4					
	4	5	3	4	*5901*	3298,81	oui	2013	38,8
		3	5	6					
		5	3	4					
		3	5	6					
	5	2	8	8	*5781*	3225,87	oui	1787	39,9
		5	2	4					
		3	5	6					
		4	3	6					
		2	8	8					
	6	3	5	6	***	3213,23	oui	1420	***
		2	8	8					
		3	5	6					
		3	5	6					
		2	8	8					
		3	5	6					
	7	3	4	8	***	3200,06	oui	1393	***
		2	8	8					
		2	8	8					
		2	8	8					
		2	8	8					
		2	8	8					
		3	4	8					

Tableau 4.3 : Résolution du modèle pour n=16

Nous remarquons que lorsque nous augmentons le nombre des couleurs, la résolution de cette instance devient impossible à l'aide de CPLEX. Nous apercevons encore que le temps de

saturation de la mémoire réservé à l'arbre de « Branch and Bound » diminue tout en augmentant le nombre des couleurs.

La majorité des instances de n=20 n'ont été pas résolue par CPLEX (Tableau 4.4). Seulement l'instance de trois couleurs a été résolue au bout d'un temps important. Toutes les autres instances engendrent une saturation de mémoire. Comme précédemment la saturation de la mémoire s'effectue en un temps moins important lorsqu'on augmente le nombre des couleurs.
Nous pouvons conclure que l'augmentation de la taille du problème et du nombre des couleurs, rendrent de plus en plus difficile la résolution du PVCM à l'aide de CPLEX.
Nous avons arrêté les tests au niveau des instances de taille 20 qui sont des instances relativement petites comparent aux instances résolues d'une manière exacte pour le problème du voyageur de commerce à deux couleurs. Cela s'explique du fait que la complexité du PVCM augmente le nombre des couleurs croît.

n	*#coul*	*#som/coul*	*alpha*	*beta*	*Zex*	*RC*	*o.m*	*T (s)*	*Gap*
20	2	10	2	2	***	7335	oui	2523	***
		10	2	2					
	3	4	3	6	7135	7097,05	***	446,2	***
		7	2	4					
		9	2	3					
	4	3	6	7	***	4874	oui	2001,6	***
		7	2	4					
		6	3	5					
		4	4	6					
	5	3	6	7	***	3861,21	oui	1827	***
		4	4	6					
		6	2	5					
		3	6	8					
		4	4	6					
	6	2	10	10	***	4055,1	oui	1745,5	***
		3	5	10					
		5	3	5					
		2	10	10					
		3	4	10					
		5	4	4					
	7	4	5	5	***	3734,74	oui	1429,3	***
		3	4	8					
		2	10	10					
		2	10	10					
		2	10	10					
		3	4	8					
		4	5	5					

Tableau 4.4 : Résolution du modèle pour n=20

4.2.2 Interprétations des résultats numériques des algorithmes

Dans cette partie nous commentons les résultats numériques fournis par les tests des algorithmes déjà exposés sur les instances testées avec la méthode exacte.

Nous commençons par les tests, du premier algorithme, présentés dans Tableau 4.5. Nous l'avons testé seulement sur les instances de six et de dix sommets et qui possèdent la même cardinalité pour toutes les couleurs. Nous remarquons que les résultats fournis par cet algorithme sont en moyenne de 23.84% à l'optimum donné par la méthode exacte. La grandeur de ce pourcentage par rapport à l'optimum nous a menés de se limité à ces instances, aussi bien que l'impuissance de cet algorithme de résoudre des instances de couleurs de cardinalités différentes.

n	*#coul*	*#som/coul*	*alpha*	*beta*	*Zh1*	*T (s)*	*% Zex*	*% RC*
6	2	3	2	2	1906	0	22,93	23,35
		3	2	2				
	3	2	3	3	2328	0	31,62	57,00
		2	3	3				
		2	3	3				
10	2	5	2	2	2434	0	4,48	7,52
		5	2	2				
	3	3	3	4	***	***	***	***
		4	2	3				
		3	3	4				
	4	2	5	5	***	***	***	***
		3	3	4				
		2	5	5				
		3	3	4				
	5	2	4	5	3434	7,05	36,34	57,40
		2	4	5				
		2	4	5				
		2	4	5				
		2	4	5				

Tableau 4.5 : Résultats numériques de l'algorithme 1

Nous passons maintenant aux interprétations des résultats obtenus après les tests des algorithmes 2, 3 et 4. Nous les comparons entre eux et par rapports aux résultats obtenus par la méthode exacte.

Pour les instances de dix sommets (Tableau 4.7), nous remarquons qu'il y à toujours l'une des heuristiques qui fournit une solution à 0% de l'optimum en un temps négligeable.

Lors des tests des instances de 10 sommets (Tableau 4.8), la première heuristique commence à perdre la capacité de s'approcher à l'optimum. Ceci est expliqué par son aspect glouton lors de la construction du premier cycle global en relaxant les contraintes d'espacement des couleurs. En calculant la moyenne arithmétique des pourcentages de chaque heuristique par rapport à l'optimum exacte, nous trouvons que la quatrième heuristique est la meilleure avec 7.66%, suivie par la première avec 9,27% et enfin la deuxième avec 10,08%. Nous remarquons aussi que pour les instances de quatre et de cinq couleurs, le troisième algorithme donne des solutions meilleures que le deuxième malgré que ce dernier possède une phase d'amélioration, c'est grâce à l'algorithme GENIUS qui donne un bon cycle au début de l'algorithme et lors de la phase de correction la qualité de ce cycle ne se dégrade pas beaucoup.

Passant aux instances de 16 sommets (Tableau 4.9), où nous constatons que le deuxième algorithme est en moyen de 13,6 % à l'optimum exacte, elle est à 34,14 % par rapport à la valeur de la relaxation en continu. Par contre, le troisième algorithme est à 3,10% à l'optimum et 27,8% par rapport à ***RC***, tandis que les moyennes du quatrième sont à 3,10% et 24,39%. Nous pouvons expliquer ces différences par le fait que le premier algorithme utilise l'heuristique du plus proche voisin pour construire son cycle initial. Ce cycle est considérer comme une borne inférieure que l'algorithme va l'utiliser pour résoudre le PVCM. Comme il est connu que l'heuristique du plus proche voisin perd son efficacité lorsqu'il s'agit d'un cycle de taille plus grande, la borne inférieur prise par le deuxième algorithme est moins efficace que celles prises par le troisième et le quatrième générée par GENIUS. D'où lors de la phase de correction des couleurs, le coût du cycle sera plus dégradé dans le deuxième algorithme.

Les mêmes remarques et arguments s'appliquent sur les instances de 20 sommets (Tableau 4.10). Nous pouvons ajouter que le temps mis par les heuristiques pour résoudre le problème est beaucoup moins important que la méthode exacte.

Le temps mis par les heuristiques augmente tous en agrandissant le nombre des couleurs pour un nombre total de sommets fixe. Cette augmentation est provoquée par le nombre des séquences de couleurs tentées lors de la phase d'admissibilité des couleurs qui est la même pour les trois algorithmes.

Si nous calculons la moyenne des pourcentages de chaque heuristique par rapport à l'optimum de tous les instances testées, nous trouvons que la quatrième heuristique possède la meilleur moyenne de 4 ,71% suivit par la troisième avec 6,24% et nous trouvons en dernier lieu la deuxième avec 9,42%. En plus des pourcentages par rapport à l'optimum, le quatrième algorithme occupe la première place avec 26,26% par rapport à la valeur de la relaxation en continu (voir Tableau 4.6).

Pour la majorité des instances, la quatrième heuristique possède la meilleure solution, *valeurs barrées en noir*, elle est suivie par le troisième algorithme et le deuxième reste en dernier lieu. (Voir pourcentage dans Tableau 4.6).

	Algorithme 2	***Algorithme 3***	***Algorithme 4***
Moy tot(%Ze)	9,42	6,24	4,71
Moy tot(%RC)	32,78	29,62	26,26
Moy tot*	22,2	28	77,8

Tableau 4.6 : Moyennes des pourcentages

En conclusion de ces tests numériques, nous pouvons classer la quatrième heuristique en premier lieu grâce aux bonnes solutions qu'elle a généré. Elle reste sous la disposition d'autres améliorations pour s'approché autant plus de l'optimum vue l'incapacité de la méthode exacte de résoudre des problème de grande taille.

n	*#coul*	*#som/coul*	*alpha*	*beta*	*Zh2*	*T (s)*	*% Ze*	*% RC*	*Zh3*	*T (s)*	*% Ze*	*% RC*	*Zh4*	*T (s)*	*% Ze*	*% RC*
6	2	3	2	2	1469	0	0,00	0,54	1469	0	0,00	0,54	1469	0	0,00	0,54
		3	2	2												
	3	2	3	3	1592	0	0,00	37,13	1614	0	1,36	37,98	1601	0	0,56	37,48
		2	3	3												
		2	3	3												
					Moy(%Ze)	**0**	***Moy(%RC)***	**18,83**	***Moy(%Ze)***	**0,68**	***Moy(%RC)***	**19,26**	***Moy(%Ze)***	**0;28**	***Moy(%RC)***	**19,01**

Tableau 4.7 : Résultats numériques des algorithmes 2, 3 et 4 pour *n*=6

n	*#coul*	*#som/coul*	*alpha*	*beta*	*Zh2*	*T (s)*	*% Ze*	*% RC*	*Zh3*	*T (s)*	*% Ze*	*% RC*	*Zh4*	*T (s)*	*% Ze*	*% RC*
10	2	5	2	2	2374	0	2,06	5,18	2403	0	3,25	6,33	2403	0	3,25	6,33
		5	2	2												
	3	3	3	4	2329	0	11,16	28,72	2557	0,183	19,08	35,07	2284	0,183	9,41	27,31
		4	2	3												
		3	3	4												
	4	2	5	5	2262	0	6,10	35,41	2257	0	5,89	35,27	2257	0	5,89	35,27
		3	3	4												
		2	5	5												
		3	3	4												
	5	2	4	5	2659	9,566	17,79	44,99	2487	7,05	12,10	41,18	2487	7,016	12,10	41,18
		2	4	5												
		2	4	5												
		2	4	5												
		2	4	5												
					Moy(%Ze)	**9,27**	***Moy(%RC)***	**28,57**	***Moy(%Ze)***	**10,08**	***Moy(%RC)***	**29,46**	***Moy(%Ze)***	**7,66**	***Moy(%RC)***	**27,52**

Tableau 4.8 : Résultats numériques des algorithmes 2, 3 et 4 pour *n*=10

n	*#coul*	*#som/coul*	*alpha*	*beta*	*Zh2*	*T (s)*	*% Ze*	*% RC*	*Zh3*	*T (s)*	*% Ze*	*% RC*	*Zh4*	*T (s)*	*% Ze*	*% RC*
16	2	8	2	2	4152	0	10,24	11,09	3938	0,016	5,36	6,26	3938	0,016	5,36	6,26
		8	2	2												
	3	2	8	8	5589	0	16,96	16,96	4681	0,016	0,85	0,85	4681	0,016	0,85	0,85
		8	2	2												
		6	2	4												
	4	5	3	4	6322	3,8	***	47,82	6020	3,166	***	45,20	5324	3,183	***	38,04
		3	5	6												
		5	3	4												
		3	5	6												
	5	2	8	8	4780	5,183	***	32,51	4746	5,833	***	32,03	4603	5,8	***	29,92
		5	2	4												
		3	5	6												
		4	3	6												
		2	8	8												
	6	3	5	6	5881	4,783	***	45,36	5938	50,799	***	45,89	5035	49,099	***	36,18
		2	8	8												
		3	5	6												
		3	5	6												
		2	8	8												
		3	5	6												
	7	3	4	8	6540	154,116	***	51,07	5044	79,8	***	36,56	4930	78,816	***	35,09
		2	8	8												
		2	8	8												
		2	8	8												
		2	8	8												
		2	8	8												
		3	4	8												
					Moy(%Ze)	**13,6**	***Moy(%RC)***	**34,13**	***Moy(%Ze)***	**3,10**	***Moy(%RC)***	**27,28**	***Moy(%Ze)***	**3,10**	***Moy(%RC)***	**24,39**

Tableau 4.9 : Résultats numériques des algorithmes 2, 3 et 4 pour *n*=16

n	*#coul*	*#som/coul*	*alpha*	*beta*	*Zh2*	*T (s)*	*% Ze*	*% RC*	*Zh3*	*T (s)*	*% Ze*	*% RC*	*Zh4*	*T (s)*	*% Ze*	*% RC*
20	2	10	2	2	8502	0	***	13,73	7616	0,033	***	3,69	7449	0,033	***	1,53
		10	2	2												
	3	4	3	6	8971	1,516	20,47	20,89	7784	0,033	8,34	8,83	7510	0,033	4,99	5,50
		7	2	4												
		9	2	3												
	4	3	6	7	8568	1,283	***	43,11	7539	3,516	***	35,35	6826	3,466	***	28,60
		7	2	4												
		6	3	5												
		4	4	6												
	5	3	6	7	9002	4,216	***	57,11	8740	15,633	***	55,82	6857	15,483	***	43,69
		4	4	6												
		6	2	5												
		3	6	8												
		4	4	6												
	6	2	10	10	8184	174,216	***	50,45	8579	144,366	***	52,73	7977	143,933	***	49,17
		3	5	10												
		5	3	5												
		2	10	10												
		3	4	10												
		5	4	4												
	7	4	5	5	7182	460,833	***	48,00	8031	441,166	***	53,50	7433	445,75	***	49,75
		3	4	8												
		2	10	10												
		2	10	10												
		2	10	10												
		3	4	8												
		4	5	5												
					Moy(%Ze)	**20,47**	***Moy(%RC)***	**29,36**	***Moy(%Ze)***	**8,34**	***Moy(%RC)***	**25,68**	***Moy(%Ze)***	**4,99**	***Moy(%RC)***	**22,42**

Tableau 4.10 : Résultats numériques des algorithmes 2, 3 et 4 pour *n*=20

Conclusion

Dans ce projet de recherche, nous avons présenté le problème du PVCM, puis nous avons établit un model pour résoudre ce problème d'une façon exacte. Nous avons constaté que la méthode exacte n'arrive pas résoudre le problème pour des instances plus au moins grandes et surtout lorsque le nombre des couleurs est important. Nous avons recours alors, à développer des heuristiques qui arrivent à résoudre le problème en un temps moins important que celui de la méthode exacte, mais qu'elle n'arrive pas toujours à trouver l'optimum. Nous avons classé la troisième heuristique comme la meilleur pour résoudre le PVCM. Cette heuristique est basée sur des techniques et des heuristiques déjà établit avant pour résoudre le PVC classique.

La plus grande instance de ce problème, résolue d'une façon exacte, est de vingt sommets et de trois couleurs. Au point de vue nombre de couleurs, nous avons résolu en maximum une instance de dix sommets et de cinq couleurs. La meilleure heuristique développée donne des solutions de moyenne de 4,71 % de la solution optimale.

Puisque, le PVCM est une extension du PVC classique nous pouvons le classer dans la catégorie des problèmes NP-dur. C'est à dire qu'on ne peut pas trouver un algorithme polynomial qui résout ce problème.

Ce travail peut être un début d'un long trajet de recherche pour améliorer les techniques de résolution du PVCM, aussi bien de trouver des nouvelles méthodes pour progresser la résolution au point de vue temps et qualité de solution.

Une idée qui peut être innovante est de développer une heuristique qui utilise les techniques de GENIUS en tenant compte directement des contraintes des couleurs. C'est à dire qu'on modifie son algorithme en prenons compte des contraintes spécifiques au PVCM. Pour la méthode exacte, l'ajouts d'autre contraintes valides peut rétrécire le champs de recherche de « Branch and Bound » d'ou l'élimination du risque de la saturation mémoire…

Bibliographie

[1] M. Desrochers et G. Laporte. « Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints » *Operations Reasearch Letters*, 10 (1991) 27-36.

[2] R. Wolfler Calvo et R. Cordone. « A heuristic approach to the overnight security service problem » *Computer and Operations Research*, 30 (2003) 1269-1287.

[3] Mélanie Bourgeois, Gilbert Laporte et Frédéric Semet. « A heuristic for the black and white traveling salesman problem » *Computer and Operations Research*, 30 (2003) 75-85.

[4] G. B. Dantzig, D. R. Fulkerson et S. M Johnson « Solution of a large scale traveling salesman problem » *Operations Research*, 2 (1954) 393-410.

[5] C. E. Miller, A.W. Tucker et R. A. Zemlin « Integer programming formulations and traveling salesman problems » *Journal of ACM*, (1960) 326-329.

[6] A. Langevin, F. Soumis and J. Desrosiers « Classification of traveling salesman problem formulation » *Operation Research Letters*, (1990) 127-132.

[7] F. Semet « Cours de gestion des systèmes de transport » *DEA-AISIH-UVHC* 2002-2003.

[8] K.T. Mark et A.J. Morton « A modified Lin-Kernighan travelling salesman heuristic » *Operation Research Letters*, 13(1993) 127-132.

[9] Gendreau M., Hertz A., Laporte G. « New Insertion and Postoptimization procedures for the Travelling Salesman Problem » *Operations Research*, 40 (1992) 1086-1094.

[10] http://www.societe-de-securite.com/presentation.htm

[11] M. Grotschel, M.W. Padberg « Partial linear characterizations of the asymmetric travelling salesman polytope » *Mathematical Programming*, 8 (1975) 378-381.

[12] M.W.Padberg, « On the facial structure of set packing polyhedra » *Math.Programming* 5 (1973) 199-216.

[13] M.W.Padberg et T.Y. Sung, « An analytical comparison of different formulations of the travelling salesman problem » *Working paper, New York University*, 1988.

[14] Rosenkrantz D.J., Stearns R.E. et Lewis II.P.M « An analysis of several heuristic for the travelling salesman problem » *SIAM Journal on Computing*, 6 (1977) 563-581.

[15] Stewart Jr., W.R. « A computationally efficient heuristic for the travelling salesman problem » *Proceeding of the 13th Annual Meetings of S.E. Times*, 75-85.

[16] Norback J., Love R. « Geometric approaches to solving the travelling salesman problem » *Management Science*, 23 (1977) 1208-1223.

[17] Norback J., Love R. « Heuristic for the hamiltonien path problem in Euclidian two space » *Journal of the Operational Research Society*, 30 (1979) 363-368.

[18] Lin S. « Computer solutions of the travelling salesman problem » *Bell System Computer Journal*, 44 (1965) 2245-2269.

[19] Ghiani G., Laporte G. et Semet F. « The black and white travelling salesman problem » *rapport technique*, Mai 2003.