Université de Montréal

Algorithmes heuristiques et exacts pour le problème de l'ensemble dominant connexe minimum

par
Sofiane Soualah

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Août, 2014

RÉSUMÉ

Dans ce mémoire, nous abordons le problème de l'ensemble dominant connexe de cardinalité minimale. Nous nous penchons, en particulier, sur le développement de méthodes pour sa résolution basées sur la programmation par contraintes et la programmation en nombres entiers. Nous présentons, en l'occurrence, une heuristique et quelques méthodes exactes pouvant être utilisées comme heuristiques si on limite leur temps d'exécution. Nous décrivons notamment un algorithme basé sur l'approche de décomposition de Benders, un autre combinant cette dernière avec une stratégie d'investigation itérative, une variante de celle-ci utilisant la programmation par contraintes, et enfin une méthode utilisant uniquement la programmation par contraintes. Des résultats expérimentaux montrent que ces méthodes sont efficaces puisqu'elles améliorent les méthodes connues dans la littérature. En particulier, la méthode de décomposition de Benders avec une stratégie d'investigation itérative fournit les résultats les plus performants.

Mots clés : Ensemble dominant connexe, décomposition de Benders, investigation itérative, programmation par contraintes, programmation en nombres entiers.

ABSTRACT

In this work, we address the minimum connected dominating set problem. We focus, in particular, on the development of solution methods based on constraint programming and integer programming, including one heuristic and some exact methods that can be used as heuristics if we limit their running time. These are based on Benders decomposition and exploit a stand-alone and an iterative probing strategy. In particular, we develop a method based on the stand-alone Benders decomposition, another combining this one with an iterative probing strategy, a variant of the latter using constraint programming, and finally, a method using only constraint programming. Experimental results show that these methods are effecient, since they perform better than those known in the literature. In particular, iterative probing Benders decomposition provides the best results overall.

Keywords: Connected dominating set, Benders decomposition, iterative probing, constraint programming, integer programming.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES SIGLES

CDS	Connected dominating set
MLST	Maximum leaf spanning tree
MLSTP	Maximum leaf spanning tree problem
MCDS	Minimum connected dominating set
MCDSP	Minimum connected dominating set problem
PPC	Programmation par contraintes
TDS	Total dominating set

À mes chers parents, pour votre amour et vos prières,
je vous suis très reconnaissant pour les valeurs que vous avez su m'inculquer.
À mes deux frères, ma sœur, et ma belle sœur, pour votre soutien moral.
À ma sœur et mon beau-frère, pour votre chaleureux accueil au Québec.
À ma femme, mon inépuisable source d'inspiration,
je n'aurais jamais pu aller jusqu'au bout sans toi.
À mes amis intimes, Hakim & Massinissa, Karim & Zazak.

REMERCIEMENTS

Je voudrais exprimer toute ma gratitude à mon directeur de recherche le professeur Bernard Gendron et mon codirecteur de recherche le professeur Gilles Pesant pour leur soutien accru, leurs vifs encouragements, leur confiance et surtout leurs judicieux conseils qui ont contribué à alimenter ma réflexion.

Je remercie également les professeurs Jean-Yves Potvin et Patrice Marcotte d'avoir consacré du temps à la lecture et l'évaluation de ce travail.

Je voudrais remercier aussi le département d'informatique et de recherche opérationnelle (DIRO) ainsi que la faculté des études supérieures et postdoctorales (FESP) de l'Université de Montréal pour les bourses qu'ils m'ont octroyées.

J'adresse mes sincères remerciements à tous les professeurs qui m'ont instruit, mes amis, collègues ainsi que toute l'équipe administrative et technique du CIRRELT, en particulier Serge Bisaillon pour sa grande disponibilité et pour nos conversations stimulantes.

Enfin dans le souci de n'oublier personne, je tiens à remercier toute personne ayant participé ou aidé de près ou de loin à la réalisation de ce modeste travail.

INTRODUCTION

Bien que relativement peu abordée à ce jour, l'étude du problème de l'ensemble dominant connexe remonte aux années 1970. L'intérêt pour ce problème s'est accru dès les années 1990 en raison de ses importantes applications dans les télécommunications et les réseaux informatiques, notamment son utilisation comme épine dorsale virtuelle (″virtual backbone″, VB) dans les réseaux sans fil.

En plus de ses applications dans les télécommunications, les transports et les réseaux sans fil, nous pouvons évoquer, entre autres, son utilité dans la diffusion des innovations technologiques, le traitement des défaillances en cascade dans les réseaux électroniques, le contrôle de la propagation des maladies transmissibles, etc. En effet, l'identification des nœuds influents dans les réseaux représentant ces situations est très importante pour des interventions efficaces en ciblant les nœuds qui couvrent la plus grande partie possible du réseau en question.

Le traitement efficace de ce problème peut donc avoir des retombées avantageuses aussi bien sur le plan scientifique que sur les plans industriel, social, technologique, etc. Pour ces raisons, il est souhaitable de pouvoir résoudre ce problème de manière efficace.

Il existe dans la littérature plusieurs algorithmes performants développés à cet effet, mais ceux-ci présentent certaines limites non négligeables relatives à la densité du graphe et à sa taille. Nous citons à titre d'exemple l'algorithme de décomposition de Benders [20], qui devient moins performant lorsque la densité du graphe est faible, ou encore la méthode de branch-and-cut [20], qui ne fonctionne pas bien sur les graphes de grande taille.

Notre objectif principal dans ce travail est de proposer des algorithmes performants qui dépassent ces limites pour fournir des résultats satisfaisants sur tous les types de graphes. Pour ce faire, nous nous sommes tournés vers les domaines de la programmation par contraintes et de la programmation en nombres entiers pour développer de nouveaux algorithmes de résolution de notre problème en nous inspirant de quelques travaux de la littérature, principalement ceux de Gendron

et al. [20], et en nous basant sur quelques méthodes de ces derniers, notamment la méthode de décomposition de Benders.

Avant de présenter le noyau de notre travail dans le chapitre 3, dans lequel les principales méthodes que nous avons développées sont présentées, nous ouvrons ce mémoire en présentant, dans le chapitre 1, quelques notions de base de la programmation par contraintes et de la programmation en nombres entiers que nous jugeons nécessaires pour faciliter la compréhension de ce qui suit. Nous enchaînons, dans le chapitre 2, avec la description du problème et sa formulation mathématique, ainsi qu'une brève revue de la littérature traitant de ce sujet. Le chapitre 4 est consacré à la présentation et à la comparaison des résultats numériques que l'exécution de nos méthodes nous a permis d'obtenir.

CHAPITRE 1

REVUE DES APPROCHES DE RÉSOLUTION

Parmi plusieurs approches de résolution pouvant être, ou ayant déjà été, utilisées pour la résolution du problème de l'ensemble dominant connexe minimum, les méthodes basées sur la programmation par contraintes et sur la programmation en nombres entiers font l'objet de ce chapitre. En effet, étant donné que nos travaux, dans ce mémoire, s'inscrivent dans le cadre de ces deux classes de méthodes, nous avons jugé intéressant de consacrer un chapitre à leur présentation. Cependant, le but principal du présent chapitre n'est pas de faire un état de l'art complet de ces domaines, mais plutôt d'en présenter les principales notions nécessaires à la compréhension des approches de résolution que nous avons développées et qui seront présentées plus loin. Nous fournissons quelques références intéressantes que nous recommandons de consulter pour de plus amples explications et une meilleure documentation sur ces domaines, notamment Apt [1], Marriott et Stuckey [30], Rossi et al. [37], Rousseau et Pesant [38] pour la programmation par contraintes et Chen et al. [12], Nemhauser et Wolsey [33], Wolsey [43] pour la programmation en nombres entiers.

1.1 Programmation par contraintes

La programmation par contraintes (PPC) est un paradigme pour la résolution des problèmes combinatoires qui a fait ses preuves dans de nombreux domaines, notamment la gestion du temps, l'affectation de ressources, la planification, l'ordonnancement, etc. Bien que ses origines remontent aux années 1960, son application effective au domaine de la recherche opérationnelle ne s'est faite qu'autour des années 1990.

Dans cette section, nous présentons de manière succincte les concepts fondamentaux de la PPC. Nous commençons par une brève description du principe général de

la PPC, puis nous définissons les problèmes de satisfaction de contraintes, et enfin nous survolons les principales approches de résolution de ces derniers, notamment la propagation de contraintes et la recherche de solutions.

1.1.1 Principe général

Pour résoudre un problème, la programmation par contraintes a recours à une modélisation par un ensemble de variables soumises à des contraintes exprimant les relations logiques qui les lient. Chacune de ces variables prend ses valeurs dans un ensemble fini de valeurs possibles dit «domaine de la variable ».

La résolution, à proprement dit, du problème repose sur deux principes indissociables appliqués sur le modèle ainsi construit, à savoir la propagation et la recherche. Le premier mécanisme, la propagation, permet de réduire le domaine de chaque variable en y éliminant les valeurs incohérentes, c'est-à-dire celles qui violeraient au moins une des contraintes du modèle si jamais elles étaient affectées à cette variable, et ceci en utilisant de nouvelles informations déduites de l'état courant du modèle. Ce processus, à lui seul, n'est souvent pas assez efficace, ni même suffisant. En effet, même si son utilisation permet de trouver une solution réalisable du problème lorsque tous les domaines des variables sont réduits en singletons ou même trancher sur sa non-réalisabilité lors de la réduction du domaine d'une variable à l'ensemble vide, éliminer toutes les valeurs incohérentes relève très souvent d'un problème **NP**-difficile et nécessite un temps d'exécution trop long en pratique. L'intervention du mécanisme de recherche devient alors nécessaire pour réduire davantage les domaines des variables. Ce dernier décompose récursivement le problème en sous-problèmes en fixant une variable, choisie selon une certaine politique préalablement définie, à une valeur de son domaine. Ceci permet d'énumérer implicitement toutes les solutions potentielles du problème en créant un arbre de recherche dans lequel une propagation des contraintes est effectuée au niveau de chaque sous-problème.

1.1.2 Problèmes de satisfaction de contraintes

Les problèmes combinatoires que la PPC s'efforce de résoudre sont généralement définis comme des problèmes de satisfaction de contraintes (″constraint satisfaction problem″, CSP) qui se caractérisent par des relations logiques entre des variables ayant chacune un domaine de valeurs possibles. Plus formellement :

Definition Un problème de satisfaction de contraintes (CSP) est défini par un triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ tel que :

- $\mathcal{X} =< x_1, x_2, \ldots, x_n >$ est un ensemble fini de n variables.
- $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ est un ensemble de n domaines finis. Chaque domaine $\mathcal{D}_i$ représente l'ensemble des valeurs possibles que peut prendre la variable x_i.
- $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ est un ensemble de m contraintes. Chaque contrainte est une relation entre certaines variables de $\mathcal{X}$ restreignant les valeurs que celles-ci peuvent prendre simultanément.

Résoudre un CSP consiste à affecter à chaque variable x_i une valeur $v_i \in D_i$ de telle sorte que chaque contrainte soit satisfaite.

1.1.3 Propagation de contraintes

Le mécanisme de propagation repose sur des opérations de réduction des domaines des variables par algorithmes de filtrage. Chaque type de contrainte possède son propre algorithme de filtrage qui examine chaque variable impliquée dans cette contrainte en vue de supprimer de son domaine toute valeur incohérente. Dès que le domaine d'une variable est ainsi réduit, les algorithmes de filtrage de toutes les contraintes impliquant cette variable sont activés pour détecter d'éventuelles modifications. Ce mécanisme est répété jusqu'à ce qu'aucune autre réduction ne soit possible. Il est nécessairement fini en autant que les domaines des variables sont finis et que l'activation de chaque algorithme de filtrage résulte en une réduction de ceux-ci.

1.1.4 Recherche de solutions

Pour résoudre les problèmes complexes, la propagation des contraintes n'est généralement pas suffisante. En effet, à l'issue de celle-ci, les domaines de certaines variables peuvent encore contenir plusieurs valeurs, et il n'est pas certain que l'on puisse aboutir à une solution réalisable en les assignant arbitrairement. Une recherche arborescente s'impose alors pour compléter le processus de résolution. Celle-ci consiste à choisir au niveau de chaque nœud et en utilisant une heuristique de choix de variables, une variable x non instanciée ainsi qu'une valeur v de son domaine, puis générer à partir de celui-ci deux autres nœuds fils qui héritent du même sous-problème associé au nœud courant enrichis des contraintes $x = v$ et $x \neq v$ respectivement. Ces deux nouveaux nœuds seront alors traités aux prochaines itérations selon l'ordre prescrit par la politique d'exploration des nœuds utilisée.

1.1.4.1 Heuristiques de choix de variable

La politique de sélection de variables et de valeurs (ou stratégie de branchement) a une incidence majeure sur le nombre de nœuds à explorer avant d'aboutir à une solution réalisable. Un choix judicieux de la stratégie est alors primordial pour une résolution efficace du problème. Il existe, dans la littérature, plusieurs stratégies de branchement. Une des plus répandues consiste à sélectionner la variable de plus petit domaine ou encore celle ordonnant les variables en fonction du nombre de contraintes dans lesquelles elles sont impliquées. Parmi les heuristiques qui ont montré d'excellents résultats, on retrouve Impact Based Search (Refalo [34]) qui utilise une notion de réduction de l'espace de recherche par une affectation donnée ou encore maxSD (Zanarini et Pesant [45]) basée sur une notion de dénombrement des solutions des contraintes.

1.1.5 Quelques contraintes globales

Malgré leur importance et leur utilisation étendue en PPC, il n'existe pas de définition formelle pour les contraintes globales. Cette appellation est utilisée pour désigner des contraintes dont l'utilisation offre plus de filtrage que n'importe quelle conjonction de contraintes équivalente. Celles-ci ont été introduites pour pallier au principal inconvénient que présente le mécanisme de propagation des contraintes. En effet, celui-ci traite chaque contrainte indépendamment des autres alors que la connaissance des autres contraintes lors du traitement de chacune d'entre elles peut sensiblement améliorer le filtrage. Parmi ces contraintes globales, nous pouvons citer :

AllDifferent$(x_1, x_2, \ldots, x_n)$ **:** Cette contrainte impose que les valeurs prises par chacune des variables $x_i, i \in \{1, \ldots, n\}$ soient deux à deux distinctes. Il existe un algorithme de filtrage efficace pour cette contrainte, qui est basé sur le calcul d'un couplage maximum dans un graphe bipartie.

Nvalue$(val, x_1, x_2, \ldots, x_n)$ **:** Cette contrainte est une généralisation de la contrainte *Alldifferent*. Elle impose que le nombre de valeurs distinctes prises par les variables $x_1, \ldots, x_n$ soit égal à val. Pour le cas particulier où $val = n$, *Nvalue* est équivalente à *Alldifferent*. Contrairement à cette dernière, seul un filtrage partiel peut être effectué pour *nvalue* au moyen d'un algorithme polynomial.

1.2 Programmation linéaire en nombres entiers

Cette section est consacrée à la présentation théorique des problèmes linéaires mixtes en nombres entiers, ainsi que de quelques méthodes connues pour leur résolution, notamment les méthodes de branch-and-bound et de branch-and-cut, de même que la méthode de décomposition de Benders.

1.2.1 Problème linéaire mixte en nombres entiers

Un problème de programmation linéaire (PL) est un problème d'optimisation consistant à minimiser (ou maximiser) une fonction objectif linéaire de n variables réelles $x_1, x_2, \ldots, x_n$ soumises à un ensemble de contraintes exprimées sous forme d'équations ou d'inéquations linéaires. Un problème de programmation linéaire en nombres entiers (PLNE) est un PL dans lequel les variables doivent prendre exclusivement des valeurs entières. La forme générale d'un PLNE est la suivante :

$$\begin{array}{ll} \underset{x}{\text{minimiser}} & \mathrm{c}x \\ \text{sous} & Ax \leq b \\ & x \geq 0 \\ & x \quad \textit{entier} \end{array}$$

où A est une matrice rationnelle de taille $m \times n$, b et c sont des vecteurs rationnels de dimensions m et n respectivement.

D'autre part, si le problème contient à la fois des variables entières et continues, on parle de problème de programmation linéaire mixte en nombres entiers (″Mixed Integer Programming″, MIP). Il est généralement formulé comme suit :

$$(P) \quad \begin{cases} \underset{x,\ y}{\text{minimiser}} & \mathrm{c}x + dy & \text{(MIP.1)} \\ \text{sous} & Ax + By \leq b & \text{(MIP.2)} \\ & x \geq 0, y \geq 0 & \text{(MIP.3)} \\ & x \quad \textit{entier} & \text{(MIP.4)} \end{cases}$$

où la notation additionnelle y désigne un vecteur de dimension p correspondant aux variables continues, d est le vecteur des coefficients de y dans la fonction objectif et B est une matrice rationnelle $m \times p$.

On appelle relaxation linéaire d'un problème de programmation linéaire mixte en nombres entiers, le problème de programmation linéaire obtenu en supprimant les contraintes d'intégralité (MIP.4) sur les variables entières x.

1.2.2 Méthode de branch-and-bound

La méthode de séparation et évaluation progressive, plus connue sous son appellation anglophone branch-and-bound, compte parmi les approches les plus connues pour la résolution des problèmes d'optimisation combinatoire. Elle suit le principe «diviser pour régner » en décomposant récursivement le problème en sous-problèmes plus simples à résoudre. L'algorithme génère et parcourt un arbre de recherche, dont chaque nœud correspond à un sous-ensemble du domaine réalisable du problème originel. Lors de ce parcours, les solutions explorées au niveau de chaque nœud sont évaluées pour ne garder que ceux contenant, potentiellement, de meilleures solutions que la solution courante.

L'algorithme commence par relaxer les contraintes d'intégralité (MIP.4) et résout à l'optimum le problème linéaire résultant. Si la solution optimale de ce dernier est entière, alors c'est une solution optimale du (MIP). Sinon, on effectue une opération de branchement en choisissant une variable x_i^* non entière dans la solution du (PL) que l'on a résolu. L'ensemble des solutions se divise alors en deux :

$$\mathsf{PL}^1 : \begin{cases} (\mathsf{P}) \\ x_i \leq \lfloor x_i^* \rfloor \end{cases} \quad \text{et} \quad \mathsf{PL}^2 : \begin{cases} (\mathsf{P}) \\ x_i \geq \lceil x_i^* \rceil \end{cases}$$

Pour les deux MIP obtenus à l'issue du processus de branchement, une première approximation est obtenue par la résolution de leurs relaxations linéaires. Si l'une de ces premières approximations n'est toujours pas entière, le PL ayant conduit à celle-ci devient candidat pour un futur branchement.

Ce processus de branchement est répété jusqu'à ce qu'une première approximation entière soit atteinte. La valeur de la fonction objectif de cette solution devient alors une borne supérieure du problème, et tous les sous-problèmes dont la première approximation, entière ou pas, excède cette borne sont écartés. Le processus de branchement est alors repris à partir des sous-problèmes restants. Si, par la suite, on obtient une nouvelle solution entière dont la valeur de la fonction objectif est inférieure à la borne supérieure, cette dernière est mise à jour avec la valeur de

la nouvelle solution ; le sous-problème ayant fourni la borne précédente, ainsi que tous ceux qui ont une première approximation supérieure à la nouvelle borne du problème, sont écartés. Le processus se poursuit ainsi jusqu'à ce qu'il ne reste plus de sous-problème dont la valeur de la relaxation linéaire est inférieure à la meilleure borne supérieure connue. Cette dernière est alors la solution optimale du problème.

La règle de branchement, qui régit le processus de choix de la variable sur laquelle l'algorithme branche, ainsi que la politique d'exploration de l'arbre de recherche, entres autres en profondeur ou en largeur, a une incidence majeure sur le nombre de nœuds à explorer avant d'aboutir à une solution optimale. Un choix judicieux est alors primordial pour une résolution efficace du problème.

1.2.3 Méthode des plans de coupes

La méthode de plans de coupes consiste à relaxer les contraintes d'intégralité (MIP.4) et résoudre à l'optimum le problème linéaire résultant. Si la solution optimale $\overline{x}$ de ce dernier est entière, alors c'est une solution optimale du (MIP). Autrement, on résout le problème dit de séparation qui consiste à déterminer si $\overline{x}$ satisfait toutes les contraintes du problème, sinon on trouve une contrainte violée, on l'ajoute au problème et on le résout de nouveau.

Grötschel et al. [22] ont démontré qu'un MIP peut être résolu en temps polynomial si et seulement si le problème de séparation des contraintes définissant son enveloppe convexe peut l'être aussi. Ce n'est malheureusement pas le cas des problèmes **NP**-difficiles, car on ne connaît pas de description linéaire complète de leurs enveloppes convexes ni d'algorithmes exacts de séparation pour toutes les familles de facettes qui les définissent. C'est pour cela qu'il arrive souvent à cette procédure de s'arrêter avant d'avoir trouvé une solution optimale.

1.2.4 Méthode de branch-and-cut

L'algorithme de branch-and-cut est une méthode hybride combinant les procédés de la méthode de branch-and-bound et ceux de la méthode des plans de

coupes. Ainsi, pour résoudre un problème linéaire en nombres entiers, la méthode de branch-and-cut commence par relaxer les contraintes d'intégralité (MIP.4) et résout à l'optimum la relaxation linéaire correspondante puis applique la méthode des plans de coupes sur la solution trouvée. Si cette méthode n'arrive pas à obtenir une solution entière alors une recherche arborescente est entreprise comme dans le cas de la méthode de branch-and-bound, mais cette fois en appliquant la méthode de plans de coupes au niveau de chaque nœud de l'arbre de branchement.

1.2.5 Méthode de décomposition de Benders

Étant donné que la contribution principale apportée dans ce travail est inspirée de la méthode de décomposition de Benders décrite dans [20], nous présentons, dans cette section, le principe de son fonctionnement.

1.2.5.1 Décomposition de Benders classique

La méthode de décomposition de Benders classique a été initialement développée par Benders [4] en 1962 pour résoudre les problèmes linéaires mixtes en nombres entiers. Elle est basée sur le partitionnement du problème original en un problème maître et un sous-problème. À chaque itération, le problème maître est résolu pour fixer la valeur d'un sous-ensemble de variables, puis le sous-problème complète cette affectation. Si cette dernière est non réalisable, le sous-problème produit une coupe dite de Benders qui est ajoutée au problème maître. Cette coupe, obtenue par la résolution du dual du sous-problème, élimine non seulement la solution particulière qui a permis sa déduction, mais aussi toute une classe de solutions non réalisables pour les mêmes raisons.

La méthode de décomposition de Benders classique peut s'appliquer sur les problèmes ayant la forme suivante :

$$\begin{aligned} \underset{x,\ y}{\text{minimiser}} \quad & z = f(x) + dy \\ \text{sous} \quad & g(x) + By \geq b \\ & x \in D, y \geq 0 \end{aligned}$$

où D est un domaine discret.

Si les valeurs des variables x sont fixées à $x = \overline{x} \in D$, on obtient une borne inférieure sur la valeur de la fonction objectif en résolvant le (PL) suivant, appelé sous-problème de Benders :

$$\begin{aligned} \underset{y}{\text{minimiser}} \quad & dy \\ \text{sous} \quad & By \geq b - g(\overline{x}) \\ & y \geq 0 \end{aligned} \tag{1.1}$$

En associant des variables duales u aux contraintes (1.1), le dual correspondant à ce sous-problème s'écrit comme suit :

$$\begin{aligned} \underset{u}{\text{Maximiser}} \quad & u(b - g(\overline{x})) \\ \text{sous} \quad & uB \leq d \\ & u \geq 0 \end{aligned}$$

D'après la théorie de la dualité, et sous l'hypothèse que le sous-problème de Benders admet un optimum fini (voir Benders [4] pour un exposé plus général), la valeur de la solution optimale u^* du problème dual est nécessairement finie et égale à la valeur de la solution optimale y^* du sous-problème de Benders, i.e., $u^*(b - g(\overline{x})) = dy^* \leq dy$.

Étant donné que le domaine réalisable du problème dual est indépendant de $\overline{x}$, l'inégalité $dy \geq u^*(b - g(x))$ est donc valide. Il en est de même pour l'inégalité suivante :

$$z = f(x) + dy \geq f(x) + u^*(b - g(x))$$

Ces coupes sont ajoutées au problème maître de Benders qui prend la forme suivante :

$$\begin{array}{lll} \underset{x}{\text{minimiser}} & z & \\ \text{sous} & z \geq f(x) + u_i^*(b - g(x)) & i = 1, \ldots, I \\ & x \in D & \end{array} \tag{1.2}$$

où $u_1^*, u_2^*, \ldots, u_I^*$ sont les solutions optimales des I premiers problèmes duaux associées aux I premiers sous-problèmes de Benders.

1.2.5.2 Décomposition de Benders logique

L'applicabilité de la méthode de décomposition de Benders classique est conditionnée par l'obtention explicite des variables duales au sens de la dualité en programmation linéaire. Néanmoins, Hooker et Ottosson [26] proposent de pallier à cette limite et d'élargir la notion de dual couramment utilisée en introduisant un dual logique (*inference dual*) pour tout type de sous-problème. Ce dernier consiste à déduire, à travers la solution courante du problème maître de Benders et les contraintes du problème, la meilleure borne inférieure possible sur la valeur de la fonction objectif. Si de telles coupes n'existent pas, le problème maître combiné avec la solution du sous-problème est une solution optimale globale du problème. Sinon, celles-ci sont ajoutées au problème maître pour être résolu de nouveau. Nous allons voir, dans le chapitre 2, un exemple d'application de cette méthode lorsque nous présenterons la méthode de décomposition de Benders développée dans Gendron et al. [20].

CHAPITRE 2

DESCRIPTION DU PROBLÈME ET REVUE DE LA LITTÉRATURE

Ce chapitre est consacré à la description du problème de l'ensemble dominant connexe, ses champs d'application ainsi que sa formulation mathématique. Nous y présentons aussi un bref état de l'art sur le problème que nous traitons. Nous y abordons notamment les principaux résultats ayant inspiré notre travail.

Notation

Étant donné un graphe non orienté $G = (V, E)$, où V est l'ensemble de ses sommets ($n = |V|$) et E celui de ses arêtes ($m = |E|$), pour tout ensemble $S \subset V$, on définit le voisinage de S comme suit :

$$\Gamma(S) = \{u \in V \setminus S : \exists v \in S, \{u, v\} \in E\}$$

Autrement dit, $\Gamma(S)$ correspond aux sommets hors S qui sont adjacents aux sommets de S. Dans le cas particulier où $S = \{s\}$, $\Gamma(\{s\})$ est l'ensemble des voisins du sommet s. Pour alléger le texte, nous adoptons la notation $\Gamma(s)$ pour désigner $\Gamma(\{s\})$.

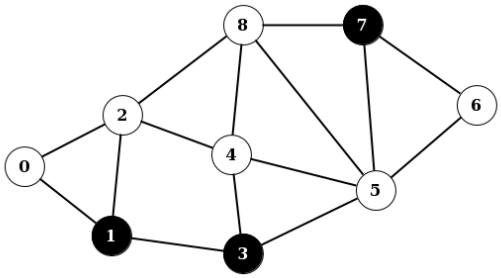
Nous aurons aussi recours à la notation $\overline{D}$ pour désigner le complémentaire d'un sous-ensemble D de V : $\overline{D} = V \setminus D$.

2.1 Description du problème

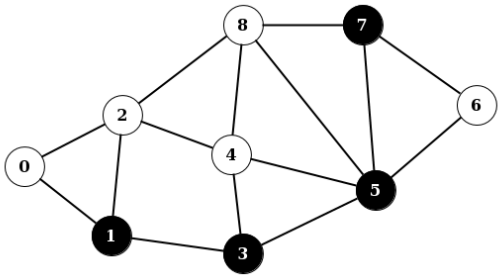
Un sous-ensemble de sommets, $\mathcal{D} \subseteq V$, est un ensemble dominant du graphe $G = (V, E)$ si chaque sommet de G est soit un membre de $\mathcal{D}$, soit adjacent à un sommet de $\mathcal{D}$. En outre, si $\mathcal{D}$ induit un sous-graphe connexe, il est appelé ensemble dominant connexe ($''$connected dominating set$''$, CDS). Le cardinal mi-

nimum d'un ensemble dominant connexe, noté $\gamma_c(G)$, est appelé le nombre de dominance connexe de G. Un CDS ayant une taille égale au nombre de domination connexe est dit ensemble dominant connexe minimum ou minimum connected dominating set (MCDS) en anglais.

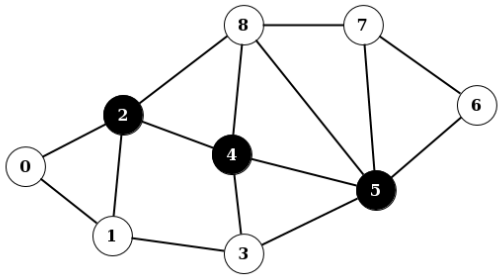
L'exemple illustré par la figure 2.1 montre trois types d'ensembles dominants formés par les sommets noirs.



(a) Ensemble dominant non connexe



(b) Ensemble dominant connexe



(c) Ensemble dominant connexe minimum

Figure 2.1 – Types d'ensembles dominants

Plus formellement, un ensemble dominant d'un graphe $G = (V, E)$ est un sous-ensemble $\mathcal{D} \subseteq V$ tel que $\Gamma(\mathcal{D}) \cup \mathcal{D} = V$. Autrement dit, $\forall v \in V,\ (\Gamma(v) \cup \{v\}) \cap \mathcal{D} \neq \emptyset$.

Un ensemble dominant $\mathcal{D}$ est dit connexe si le sous-graphe $(\mathcal{D}, E(\mathcal{D}))$, qu'il induit, est connexe, avec $E(\mathcal{D}) = \{\{i, j\} \in E |\ i \in \mathcal{D}, j \in \mathcal{D}\}$.

La construction d'un ensemble dominant connexe minimum d'un graphe est un problème NP-complet [23].

2.1.1 Problèmes apparentés

Nous recensons dans la littérature deux problèmes analogues au problème de l'ensemble dominant connexe minimum, le problème de l'arbre couvrant à nombre maximum de feuilles, qui y est étroitement lié, et celui du recouvrement d'ensemble connexe, dont notre problème est un cas particulier.

2.1.1.1 Le problème de l'arbre couvrant à nombre maximum de feuilles (MLSTP)

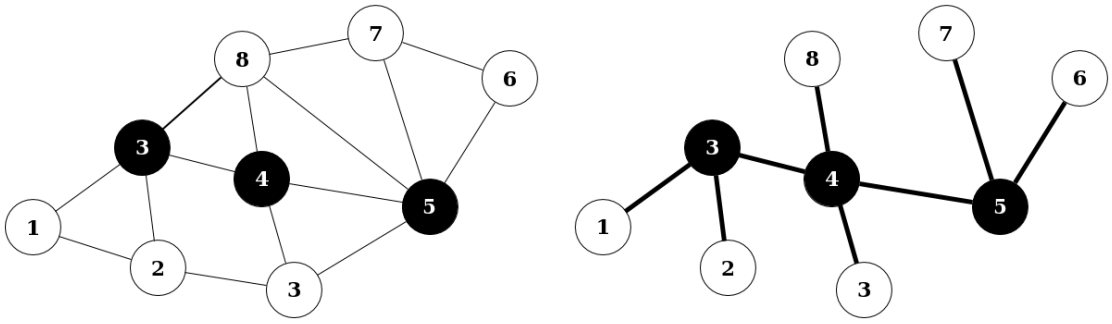
Un arbre couvrant d'un graphe G est un sous-graphe T connexe, sans cycle contenant tous les sommets de G. Une feuille dans un arbre T est un sommet dont le degré dans T est égal à un.

Le problème de l'arbre couvrant à nombre maximum de feuilles (″maximum leaf spanning tree problem″, MLSTP), consiste à trouver un arbre couvrant avec autant de feuilles que possible (voir Lucena et al. [29] pour une revue de la littérature sur ce problème).

MCDSP et MLSTP

Le MCDSP est étroitement lié au MLSTP. En effet, étant donné un ensemble dominant connexe $\mathcal{D}$, un arbre couvrant de $G(\mathcal{D})$ peut aisément être identifié. Ce dernier peut être généralisé pour former un arbre couvrant de G dans lequel tous les sommets de $V \setminus \mathcal{D}$ sont des feuilles. Ainsi, pour chaque ensemble dominant connexe $\mathcal{D}$ de G, un arbre couvrant de G avec au minimum $|V| - |\mathcal{D}|$ feuilles peut être trouvé. Par conséquent, si $\mathcal{D}$ est un ensemble dominant connexe de cardinalité minimale, l'arbre couvrant de G qui en sera déduit aura le plus grand nombre de feuilles possible.

Dans la figure 2.2a, les sommets de l'ensemble dominant connexe minimum sont représentés en noir. À partir de celui-ci, on obtient l'arbre couvrant à nombre maximum de feuilles représenté à la figure 2.2b



(a) Ensemble dominant connexe minimum (b) Arbre couvrant à nombre maximal de feuilles

Figure 2.2 – MCDSP et MLSTP

2.1.1.2 Le problème de recouvrement d'ensemble connexe (CSC)

Soit $\mathcal{U} = \{1, \ldots, m\}$ un ensemble fini de m éléments, appelé univers, et $\mathcal{X}$ une collection de sous-ensembles de $\mathcal{U}$; $\mathcal{X} = \{S_1, S_2, \ldots, S_n\}$, où $S_i \subseteq \mathcal{U},\ i = 1, \ldots, n$. Le problème de recouvrement d'ensemble à coûts unitaires consiste à déterminer un ensemble $C \subseteq \mathcal{X}$ aussi petit que possible tel que tous les éléments de $\mathcal{U}$ soient recouverts ; autrement dit, chaque élément de $\mathcal{U}$ se trouve dans au moins un des sous-ensembles de C.

Supposons qu'en plus de l'univers $\mathcal{U}$ et de la collection $\mathcal{X}$, on a un graphe auxiliaire $H = (\mathcal{X}, E(\mathcal{X}))$, tel que $E(\mathcal{X})$ est un ensemble d'arêtes reliant certains sous-ensembles de la collection $\mathcal{X}$. Un ensemble $C \subseteq \mathcal{X}$ est un recouvrement d'ensemble connexe si le sous-graphe induit $H(C) = (C, E(C))$ est connexe.

MCDSP et CSC

Il est facile de voir que le problème de l'ensemble dominant connexe est un cas particulier du problème de recouvrement d'ensemble connexe. En effet, étant donné un graphe non orienté $G = (V, E)$ on peut dériver une instance du problème CSC à partir d'une instance MCDS du graphe G comme suit :

- L'univers $\mathcal{U}$ est l'ensemble des sommets V du graphe G ; $\mathcal{U} = V$.

- Chaque ensemble S_i représente le sommet i et ses voisins $\Gamma(i)$; $S_i = \Gamma(i) \cup \{i\}$
- Le graphe auxiliaire H est le même que le graphe original G, sauf que chaque sommet v de G est remplacé par S_v dans H.

2.1.2 Champs d'application

Le concept de la dominance dans les graphes est à l'origine de plusieurs applications trouvées dans la littérature. Les premières pouvant être citées sont la localisation des stations de radars (Berge [5]), et certains réseaux de communication particuliers décrits dans Liu [28]. De nos jours, ce concept trouve des applications dans des domaines très variés tel que la diffusion des innovations technologiques (Rogers [36], Valente [40]), la commercialisation des nouveaux produits (Domingos et Richardson [14], Goldenberg et al. [21]), les systèmes d'alimentation (Asavathiratham et al. [2], Fan et Watson [17]), la propagation des maladies transmissibles (Eubank et al. [16]), et l'aide à l'atténuation des problèmes sociaux à travers les réseaux sociaux (Wang et al. [42]).

Parmi les nombreux champs d'applications que possède le problème de l'ensemble dominant connexe, nous avons choisi, dans ce qui suit, son utilisation comme épine dorsale virtuelle dans les réseaux sans fil.

Épine dorsale virtuelle (″virtual backbone″, VB)

En l'absence d'infrastructure physique dans un réseau sans fil, et sans utiliser l'épine dorsale (VB), chaque nœud de ce réseau doit stocker une table de routage pour pouvoir communiquer avec les autres nœuds. Ceci engendre d'importants coûts de communication relatifs au surplus de la diffusion, aux tailles et coûts de maintenance des tables de routage, etc. L'utilisation d'un VB peut considérablement réduire ces coûts en omettant les connexions redondantes dans le réseau par le choix d'un sous-ensemble de nœuds à travers lequel se font toutes les communications sur le réseau.

Considérons, à titre d'exemple, le réseau sans fil à sept nœuds représenté à la

figure 2.3a, et supposons que le réseau de communication est celui représenté à la figure 2.3b, sachant que les connexions possibles dépendent de la structure physique du réseau sans fil, ainsi que de la nature des périphériques qui le composent tel que leur portée et leur position. Le VB correspond, dans ce cas, à l'ensemble dominant connexe minimum du graphe associé à ce réseau. Il est constitué, ici, des nœuds 3 et 5. En effet, ces deux nœuds sont les seuls à devoir stocker une table de routage pour assurer la communication à travers tout le réseau. Il suffit à tous les autres de connaître un seul voisin appartenant à cet ensemble pour pouvoir communiquer avec tous les autres nœuds du réseau. Si, par exemple, le nœud 7 désire communiquer avec le nœud 4, il lui suffit de transmettre les données à envoyer au nœud 5 en lui indiquant le nœud destinataire, et celui-ci se chargera de les acheminer jusqu'au nœud 4, comme l'illustre la figure 2.3d, par un chemin de routage qu'il aura choisi à travers le VB en utilisant la table de routage dont il dispose.

2.1.3 Formulation mathématique du problème

En associant à chaque sommet $i \in V$ une variable binaire x_i, tel que

$$x_i = \begin{cases} 1, & \text{si le sommet } i \in V \text{ appartient à l'ensemble dominant connexe,} \\ 0, & \text{sinon,} \end{cases}$$

on peut modéliser le problème de l'ensemble dominant connexe minimum comme suit :

$$\min \sum_{i \in V} x_i \qquad (1)$$

$$\sum_{j \in \Gamma(i)} x_j + x_i \geq 1, \qquad i \in V \qquad (2)$$

$$Connexe(x) \qquad (3)$$

$$x_i \in \{0, 1\}, \qquad i \in V \qquad (4)$$

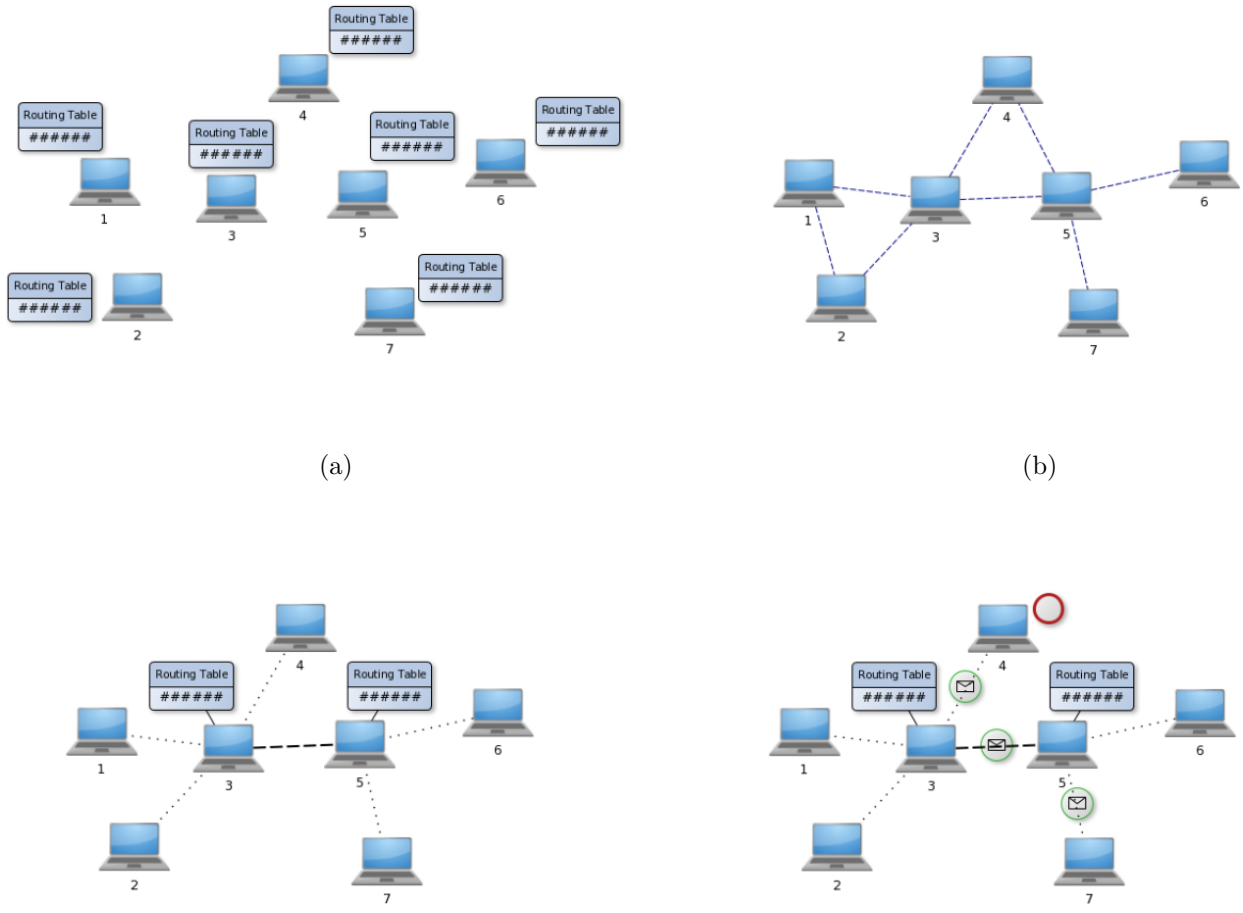


Figure 2.3 – Épine dorsale virtuelle (Virtual Backbone)

La fonction objectif (1) minimise le nombre de sommets faisant partie de l'ensemble dominant connexe. Les inégalités de couverture (2) définissent un ensemble dominant. Les contraintes (3), sous leur forme générale, imposent la connexité du sous-graphe engendré par x. Elles seront abordées à la section suivante.

L'existence d'un ensemble dominant de cardinalité 1 peut facilement être vérifiée par la condition $\Gamma(i) \cup \{i\} = V$, pour $i \in V$. Si un tel ensemble n'existe pas, et puisque nous cherchons un ensemble connexe, les contraintes (2), qui à la base

définissent un ensemble dominant, peuvent être renforcées en les remplaçant par les contraintes (5) qui définissent un ensemble dominant total (″total dominating set″, TDS). Un TDS est un ensemble dominant sans sommets isolés. Autrement dit, c'est un ensemble dominant dans lequel les sommets ne se couvrent pas eux-mêmes.

$$\sum_{j \in \Gamma(i)} x_j \geq 1, \qquad i \in V \qquad (5)$$

2.2 Revue de la littérature

Puisque notre problème est soumis à une contrainte de connexité, nous avons jugé intéressant d'ouvrir cette section avec ce sujet et de décrire les façons dont elle a été traitée dans certains travaux. Nous présentons ensuite quelques résultats de la littérature que nous avons utilisés dans notre travail ; nous nous attardons en particulier sur l'article de Gendron et al. [20] qui en est l'article de base et la première source d'inspiration.

2.2.1 Contraintes de connexité

Il existe dans la littérature plusieurs formulations pour modéliser la connexité dans les graphes. La plupart d'entre elles utilisent des variables binaires associées aux arêtes, entre autres Gendron et al. [20] et Simonetti et al. [39].

En 2005, Cerdeira et al. [10] proposent une nouvelle formulation du problème de recouvrement d'ensemble connexe. Ils sont parvenus, en exploitant la structure particulière de ce problème, à construire une formulation des contraintes de connexité qui lui est spécifique. Ils les ont ensuite renforcées en utilisant les résultats obtenus dans [11] sur les conditions nécessaires et suffisantes pour la caractérisation des facettes de l'enveloppe convexe de ce problème. Bien qu'une analogie entre ce problème et le nôtre soit établie dans plusieurs articles, entre autres Ren et Zhao [35], les résultats obtenus par Cerdeira et al. n'ont, à notre connaissance, jamais été transposés au problème de l'ensemble dominant connexe.

D'un autre coté, Fügenschuh et Fügenschuh [19] proposent, pour la première

fois dans leur article en 2008, une nouvelle formulation de la connexité en utilisant des variables associées aux nœuds du graphe ainsi que des contraintes basées sur la notion de coupes sur les sommets (″node-cuts″). Celle-ci n'est cependant pas assez développée, car le sujet principal de cet article était autre que la connexité. Ce n'est qu'en 2013 que cette piste s'est vue explorée plus en profondeur dans l'article de Carvajal et al. [9] qui a amélioré et développé cette formulation en proposant une meilleure caractérisation de la connexité à travers les coupes sur les sommets.

2.2.2 Approches heuristiques

Une grande partie des travaux réalisés dans la littérature sur le MCDSP adoptent des approches heuristiques ; la plupart sont réalisés dans le cadre spécifique de la construction des épines dorsales. Le travail de Guha et Khuller [23] peut être considéré comme l'article fondateur de cet axe : ils ont conçu le premier algorithme d'approximation en temps polynomial qui garantit un facteur d'approximation de $(1 + \ln n)$, où n est le nombre de sommets du graphe. Autrement dit, la taille de l'ensemble dominant connexe fourni par leur heuristique ne peut excéder $(1 + \ln n)$ fois la taille de l'ensemble dominant connexe minimum. Ils ont aussi démontré la non-approximabilité du MCDSP au-delà d'un facteur de $\rho \ln n$ pour $0 < \rho < 1$. Depuis, plusieurs heuristiques, tentant d'améliorer ce facteur d'approximation, ont vu le jour, et peuvent être classées en deux catégories.

La première catégorie s'exécute en deux phases. La première phase consiste à construire un ensemble dominant. Dans une seconde phase, les éléments de ce dernier sont connectés en ajoutant d'autres nœuds intermédiaires [23, 41].

La deuxième catégorie, quant à elle, se consacre à la construction d'un ensemble dominant connexe en une seule phase [23, 29, 44].

Dans cette dernière perspective, nous pouvons citer l'heuristique de Lucena et al. [29], qui n'est finalement autre qu'une implémentation basée sur la notion de voisinage de l'approche de Guha et Khuller [23]. Elle sera abordée plus en détail à la section 3.1.

2.2.3 Approche par la programmation par contraintes

À notre connaissance, la communauté scientifique ne s'est jamais penchée sur la résolution du problème de l'ensemble dominant connexe via la programmation par contraintes. Cependant, si on modélise le problème avec une collection de variables x_i qui désignent, parmi les voisins de chaque sommet i, celui qui le couvre, l'utilisation de la contrainte globale $Nvalue(val, x_1, ..., x_n)$ permet de trouver un ensemble dominant de taille val. Malheureusement, cette contrainte comporte beaucoup de symétrie et sa propagation relève d'un problème **NP**−difficile.

2.2.4 Approche par la programmation linéaire mixte en nombres entiers

Les approches MIP pour le MCDSP ont attiré moins d'attention que les approches heuristiques. Une formulation du problème par la programmation linéaire mixte en nombres entiers, utilisant un nombre exponentiel de contraintes basées sur l'arbre couvrant, ainsi qu'une version préliminaire d'un algorithme de branch-and-cut ont été décrits dans Simonetti et al. [39]. Cette dernière sera ensuite développée par Gendron et al. [20]. Des formulations plus compactes de ce problème, utilisant un nombre polynomial de contraintes, ont été proposées dans Fan et Watson [17]. Ces formulations ont été réalisées suite à l'imposition de la connexité via plusieurs moyens : les inégalités de Miller-Tucker-Zemlin initialement proposées pour résoudre le problème du TSP [32], les contraintes de multi-flots, et les contraintes d'élimination de sous-tours introduites dans Martin [31].

Par ailleurs, d'autres approches conçues pour traiter le MLSTP peuvent aussi être utilisées pour résoudre le MCDSP. Nous pouvons citer, entre autres, le travail de Lucena et al. [29] dans lequel deux formulations en programmation en nombres entiers ont été proposées. La première est basée sur une reformulation du problème par l'arbre de Steiner, et la deuxième considère le problème dans sa version orientée et tente de trouver une arborescente couvrante avec autant de feuilles que possible. Les résultats expérimentaux de cet article montrent que de meilleurs résultats sont

obtenus avec ce qui y est appelé la méthode DGR, qui exploite la deuxième formulation dans un algorithme de branch-and-cut (voir Lucena et al. [29] pour plus de détails).

Passons, à présent, à la présentation de l'article de Gendron et al. [20], qui est l'article de base et la principale source d'inspiration pour notre travail. Dans celui-ci, six méthodes exactes pour la résolution du MCDSP ont été proposées. Elles sont basées sur deux approches principales, à savoir la méthode de décomposition de Benders et la méthode de branch-and-cut. Une troisième approche hybride combinant ces deux dernières est aussi utilisée. Pour chacune de ces trois approches deux variantes sont considérées, une utilisant la stratégie pure (″stand-alone″), et l'autre utilisant la stratégie d'investigation itérative. Cette dernière est construite en se basant sur la propriété élémentaire qui stipule que la non-existence de CDS d'une certaine cardinalité `d` entraîne la non-existence de CDS de cardinalité inférieure à `d`.

Les deux approches principales ne diffèrent que dans la manière par laquelle les contraintes de connexité sont traitées. Considérons le fait qu'un sous-graphe est connexe si et seulement s'il contient un arbre couvrant : l'algorithme de branch-and-cut introduit des variables additionnelles associées aux arêtes ainsi que des contraintes d'élimination de sous-tours généralisées (GSEC) pour exprimer la connexité du sous-graphe induit par la solution, donnant ainsi lieu à une formulation en programmation linéaire mixte en nombres entiers du MCDSP. Le problème de séparation associé aux contraintes (GSEC) utilisées dans cette formulation peut être résolue efficacement en $O(n^4)$ via un algorithme de ″maximum flow-minimum cut″. Cependant, pour un temps d'exécution global plus avantageux de la méthode de branch-and-cut, les auteurs ont opté pour une séparation des contraintes par une méthode heuristique spécialement conçue à cet effet. Cette méthode est désignée par l'appellation (SABC) dans l'article de Gendron et al. [20].

Étant donné que notre travail est principalement basé sur la méthode de décomposition de Benders et la stratégie d'investigation itérative, nous nous attardons sur la présentation de celles-ci.

Méthode de décomposition de Benders

L'approche de Benders consiste à relaxer les contraintes de connexité (3) et à itérer entre :

i) Le problème maître de Benders, défini par les contraintes (1),(5),(4) et des coupes additionnelles dites *coupes de Benders*.

ii) Le sous-problème de Benders qui consiste à vérifier si la solution fournie par le problème maître engendre un ensemble connexe et à ajouter des coupes de réalisabilité dans le cas contraire.

Coupes de Benders

Supposons que le problème maître de Benders fournisse comme solution un ensemble dominant non connexe $\mathcal{D}$. Afin de satisfaire la contrainte de connexité, il est certain qu'au moins un élément de $\overline{\mathcal{D}} = V \setminus \mathcal{D}$ doit y être ajouté. La contrainte suivante est donc valide :

$$\sum_{i \in \overline{\mathcal{D}}} x_i \geq 1 \tag{2.1}$$

Du moment que ces contraintes sont en nombre fini, et que tous les ensembles dominants non connexes peuvent être écartés par leur ajout, cet algorithme converge vers une solution optimale.

Ces coupes sont généralement faibles vu que la plupart des ensembles non connexes nécessitent l'ajout de bien plus qu'un sommet pour les connecter. Pour les renforcer, les auteurs ont proposé de résoudre le problème de Steiner sur le graphe réduit de la solution courante $\mathcal{D}$, c'est-à-dire le graphe obtenu en compactant chaque composante connexe en un sommet terminal et laissant les autres sommets, ceux de $V \setminus \mathcal{D}$, comme sommets non terminaux.

Le problème de l'arbre de Steiner consiste, étant donné un ensemble T de sommets dits sommets terminaux, à trouver un ensemble $S \in (V \setminus T)$ de sommets additionnels, appelés points de Steiner, tel que le coût de l'arbre couvrant les som-

mets de T soit minimum. Le nombre de Steiner est le nombre minimum de sommets devant être ajoutés à T pour le connecter.

Supposons que le nombre de Steiner ainsi obtenu soit égal à $m_{\mathcal{D}}$, alors la coupe 2.1 peut être renforcée comme suit :

$$\sum_{i \in \overline{\mathcal{D}}} x_i \geq m_{\mathcal{D}} \tag{2.2}$$

Cependant, le problème de Steiner est NP-complet [27], et sa résolution peut s'avérer très coûteuse en temps. Pour cela, les auteurs ont proposé de calculer, au lieu de sa valeur exacte, une borne inférieure du nombre de Steiner comme suit. La longueur, en termes de sommets non-terminaux intermédiaires, du plus court chemin entre chaque paire de sommets terminaux équivaut au nombre minimum de sommets devant être ajoutés à $\mathcal{D}$ pour les connecter. En choisissant la plus grande des plus courtes distances séparant chaque paire de ces sommets, on obtient une borne inférieure valide du nombre de Steiner, notée $\underline{m}_{\mathcal{D}}$. La contrainte 2.2 est donc remplacée par la contrainte suivante :

$$\sum_{i \in \overline{\mathcal{D}}} x_i \geq \underline{m}_{\mathcal{D}} \tag{2.3}$$

Déroulement de l'algorithme

Soit $\mathcal{D}^0$ un ensemble dominant connexe initial. L'algorithme commence par résoudre le problème maître de Benders, défini initialement par les contraintes (1),(4) et (5), auquel on ajoute la coupe d'optimalité suivante :

$$\sum_{i \in V} x_i \leq |\mathcal{D}^0| - 1$$

Si le problème est irréalisable, alors $\mathcal{D}^0$ est une solution optimale. Sinon, soit $\mathcal{D}$ l'ensemble dominant obtenu. S'il est connexe, alors c'est une solution optimale. Sinon, on génère la coupe de réalisabilité 2.3, comme indiqué précédemment, et on l'ajoute au problème maître de Benders pour le résoudre de nouveau.

Stratégie d'investigation itérative

Cette stratégie est construite en se basant sur la propriété élémentaire qui stipule que la non-existence d'un CDS d'une certaine cardinalité d entraîne la non-existence d'un CDS de cardinalité inférieure à d. Elle est basée sur la proposition suivante :

Proposition 2.2.1. *S'il existe un CDS de cardinalité $d < |V|$, alors il en existe un de cardinalité $d + 1$*

La démonstration de cette propriété est triviale. En effet, s'il existe un CDS $\mathcal{D}$ de cardinalité $d < |V|$, en ajoutant n'importe quel sommet de $V \setminus \mathcal{D}$ à $\mathcal{D}$ on obtient un CDS de cardinalité $d + 1$.

On peut alors énoncer le corollaire suivant, qui découle directement de cette proposition

Corollaire 2.2.2. *S'il n'existe aucun CDS de cardinalité $d > 0$, alors il n'en existe aucun de cardinalité $d - 1$.*

Comme son nom l'indique, la stratégie d'investigation itérative permet la recherche de la solution optimale du problème par une approche itérative. En partant d'une solution réalisable $\mathcal{D}$ de taille $d + 1 > 1$, cette stratégie cherche un CDS de cardinalité d. Si un tel ensemble n'existe pas, l'algorithme s'arrête en fournissant $\mathcal{D}$ comme solution optimale. Sinon, une nouvelle itération est réalisée en partant du nouvel ensemble dominant de taille d pour tenter d'en retrouver un autre de taille $d - 1$. Cette stratégie est décrite dans le pseudo-code 1.

Notons qu'en vertu du corollaire 2.2.2 énoncé plus haut, si le problème de satisfaction de contraintes, «**2.Investigation :** Trouver un CDS de cardinalité `d` », est résolu de manière exacte, cette stratégie fournit une solution optimale du problème de l'ensemble dominant connexe minimum.

Décomposition de Benders avec stratégie d'investigation itérative

À chaque étape de la stratégie d'investigation itérative, on peut exécuter l'algorithme de décomposition de Benders exposé plus haut pour résoudre le problème

début
  1.Trouver un ensemble dominant connexe initial $\mathcal{D}^0$.
  si $|\mathcal{D}^0| = 1$ **alors**
    retourner $\mathcal{D}^0$ est une solution optimale de cardinalité 1.
  sinon
    poser `d` $\leftarrow |\mathcal{D}^0| - 1$
  fin
  2.Investigation : Trouver un CDS de cardinalité `d`
  3. si un tel ensemble n'existe pas **alors**
    retourner $\mathcal{D}^0$ est une solution optimale de cardinalité `d+1`
  sinon
    Soit $\mathcal{D}$ l'ensemble dominant trouvé.
    `d` $\leftarrow |\mathcal{D}| - 1$
    Aller à l'étape 2.
  fin
fin

Algorithm 1: Pseudo-code de la stratégie d'investigation itérative

de satisfaction de contraintes «**2.Investigation :** Trouver un CDS de cardinalité `d` », et ceci en ajoutant au problème maître de Benders la contrainte de cardinalité $\sum_{i \in V} x_i = d$.

Cette variante de la méthode de décomposition de Benders a pu être améliorée en introduisant une procédure de renforcement qui consiste à vérifier, à chaque détection d'un CDS de cardinalité `d`, l'existence d'un autre CDS de cardinalité `d-1` en retirant chaque élément à tour de rôle. Si un tel ensemble existe, la procédure est réexécutée en partant de celui-ci, sinon des contraintes valides sont générées (voir [20] pour plus de détails).

Si, au lieu de la méthode de décomposition de Benders, on utilise, à chaque étape de la stratégie d'investigation itérative, la méthode de branch-and-cut présentée dans l'article de Gendron et al. [20], on obtient la méthode (IPBC) développée par ces mêmes auteurs.

CHAPITRE 3

RÉSOLUTION DU PROBLÈME

Le présent chapitre constitue le cœur de ce mémoire. Nous y présentons l'essence de notre travail, en l'occurrence les approches de résolution que nous avons proposées pour le traitement du problème présenté au chapitre 2, dont une heuristique et plusieurs méthodes exactes qui peuvent aussi être utilisées comme heuristiques si on limite leur temps d'exécution. Nous commençons par présenter la méthode heuristique qui est, en fait, une amélioration d'une heuristique déjà existante. Nous passons ensuite à la présentation des méthodes exactes, en l'occurrence une utilisant l'approche de décomposition de Benders, une combinant cette approche avec la stratégie d'investigation itérative, une variante de celle-ci en utilisant la programmation par contraintes, puis enfin, une dernière utilisant uniquement la programmation par contraintes.

3.1 Heuristique gloutonne dynamique renforcée (HDR)

Cette heuristique vient renforcer l'heuristique (HDM) développée dans Gendron et al. [20] qui est elle-même une variante à exécutions multiples de l'heuristique gloutonne dynamique (HD) développée dans Lucena et al. [29]

3.1.1 Principe de fonctionnement de HD

L'algorithme est implémenté à l'aide d'un ensemble $\mathcal{D}$ de sommets dominants et un ensemble $\mathcal{L}$ des voisins de $\mathcal{D}$, $\mathcal{L} = \Gamma(\mathcal{D})$. Son principe consiste à déplacer, à chaque itération, le sommet de $\mathcal{L}$ ayant le plus grand nombre de voisins dans $\mathcal{V} \setminus \{\mathcal{L} \cup \mathcal{D}\}$ vers $\mathcal{D}$ et cela jusqu'à obtention d'un ensemble dominant connexe. Le pseudo-code de la méthode est le suivant :

begin

Initialisation : $\mathcal{D} \leftarrow \{v\} \quad \mathcal{L} \leftarrow \Gamma(v)$

while $(\mathcal{D} \cup \mathcal{L} \neq \mathcal{V})$ **do**

`Choisir` $s \in \mathcal{L}$ `ayant le plus grand nombre de voisins dans`

$\mathcal{V} \setminus \{\mathcal{L} \cup \mathcal{D}\}$

Mise à jour :

$\mathcal{D} \leftarrow \mathcal{D} \cup \{s\}$

$\mathcal{L} \leftarrow (\mathcal{L} \setminus \{s\}) \cup \Gamma(s)$

end

`BorneSup` $\leftarrow |\mathcal{D}|$

return `BorneSup`

end

Algorithm 2: Pseudo-code de l'heuristique (HD) de Lucena et al. [29]

Dans le cas de (HDM), cette heuristique est exécutée n fois, et à chaque exécution, l'ensemble $\mathcal{D}$ est initialisé avec un nouveau sommet $v \in V$.

3.1.2 Inconvénients

Cette méthode permet d'aboutir à coup sûr à un ensemble dominant connexe. Elle est très efficace dans le sens où le fait de choisir les sommets ayant le plus grand nombre de voisins dans $\mathcal{V} \setminus \{\mathcal{L} \cup \mathcal{D}\}$, permet de garantir, à chaque itération, le choix des sommets les plus prometteurs et converger ainsi plus rapidement vers cet ensemble dominant. Cependant, il s'agit d'un algorithme glouton qui utilise des choix localement optimaux, mais qui ne le sont pas nécessairement à long terme. Ceci est dû au fait que le choix d'un sommet à une certaine étape, même si celui-ci en couvre plusieurs autres, peut s'avérer inutile à long terme. En effet, après plusieurs itérations, il se peut que d'autres sommets, devant impérativement appartenir à $\mathcal{D}$, car ils sont les seuls à pouvoir couvrir certains autres sommets du graphe, parviennent aussi à couvrir, collectivement, les voisins du sommet précédemment choisi. Dans ce cas, ce dernier devient inutile et sa suppression permet de garder

un ensemble dominant connexe de cardinalité moindre. De plus, cet algorithme ne traite pas les ex æquo, c'est-à-dire que dans le cas où il existe plus d'un sommet dans $\mathcal{L}$ avec le plus grand nombre de voisins, l'algorithme ne dispose pas de procédure lui permettant de les départager.

3.1.3 Amélioration

Notre contribution vient améliorer cette heuristique sur ces points. Dans un premier temps, nous avons introduit des règles de priorité qui nous permettent de gérer les ex æquo. Lorsqu'une telle situation se présente, le choix de l'un ou l'autre de ces sommets est plus ou moins avantageux selon la nature du graphe. En effet, lorsque celui-ci est de faible densité, il est préférable d'explorer des sommets éloignés car ils offrent de meilleures chances de couvrir tout le graphe. Dans ce cas, prioriser les sommets les plus récemment admis dans $\mathcal{L}$ serait le plus raisonnable. Par contre, lorsque le graphe est plutôt dense, la dominance est généralement assez facilement atteignable et il est donc préférable de choisir des sommets proches pour garantir la connectivité avec un minimum de sommets. Dans ce cas, il est plus intéressant de prioriser les premiers sommets introduits dans $\mathcal{L}$. Pour être sûr que le meilleur choix soit fait, on exécute la suite de la procédure à deux reprises, d'abord en priorisant les premiers sommets introduits dans l'ensemble $\mathcal{L}$, puis une seconde fois, en priorisant les sommets les plus récemment admis dans cet ensemble. On garde ensuite la meilleure alternative, i.e., celle qui fournit un ensemble dominant de plus petite cardinalité.

Dans un second temps, nous avons recours à une procédure d'élagage à la fin de l'exécution de l'algorithme pour éliminer les sommets superflus de l'ensemble dominant. Celle-ci consiste en la vérification de la conformité de l'ensemble $\mathcal{D}$, retourné par l'algorithme précédent, en le privant de ses éléments un à un, pour enfin éliminer ceux qui n'ont aucune influence sur ses propriétés de connectivité et de dominance.

3.2 Approche par la programmation linéaire en nombres entiers

Dans cette section, nous proposons une méthode de résolution du problème de l'ensemble dominant connexe en s'inspirant de divers travaux réalisés par plusieurs auteurs. Cette méthode est principalement inspirée de la méthode de décomposition de Benders appliquée à ce problème par Gendron et al. [20]. Elle en diffère par la manière avec laquelle les contraintes de connexité sont traitées.

Pour ce faire, nous avons utilisé quelques résultats pertinents tirés notamment des travaux de Cerdeira et al. [10]. Nous nous sommes aussi inspirés de l'amélioration apportée par Carvajal et al. [9] à la formulation des contraintes de connexité proposée par Fügenschuh et Fügenschuh [19], ainsi que quelques caractéristiques et résultats que nous avons nous-même établis pour renforcer cette formulation.

Principe de la méthode de décomposition de Benders (rappel)

L'approche de Benders consiste à relaxer les contraintes (3) et à itérer entre :

i) Le problème maître de Benders, défini par les contraintes (1),(5),(4) et des coupes additionnelles dites *coupes de Benders* :

$$\min \sum_{i \in V} x_i \qquad (1)$$

$$\sum_{j \in \Gamma(i)} x_j \geq 1, \qquad i \in V \qquad (5)$$

$$\textit{Coupes de Benders} \qquad (6)$$

$$x_i \in \{0,1\}, \qquad i \in V \qquad (4)$$

ii) Le sous-problème de Benders qui consiste à vérifier si la solution fournie par le problème maître engendre un ensemble connexe et ajouter des coupes de réalisabilité dans le cas contraire.

3.2.1 Coupes de Benders

Supposons que le problème maître de Benders fournisse comme solution un ensemble dominant non connexe $\mathcal{D}$. Afin de satisfaire la contrainte de connexité, il est certain qu'au moins un élément de $\overline{\mathcal{D}} = V \setminus \mathcal{D}$ doit y être ajouté. La contrainte suivante est donc valide :

$$\sum_{i \in \overline{\mathcal{D}}} x_i \geq 1 \tag{3.1}$$

Du moment que ces contraintes sont en nombre fini, et que tous les ensembles dominants non connexes peuvent être écartés par leurs ajouts, cet algorithme converge vers une solution optimale. Cependant, le nombre de coupes devant être ajoutées au problème maître avant qu'un ensemble dominant connexe soit atteint peut limiter l'utilisation de cette procédure, car sa convergence est lente. Pour accélérer cette méthode, nous avons entrepris d'ajouter, au lieu de cette contrainte, ou un renforcement de celle-ci (Gendron et al. [20], Cerdeira et al. [10]), une collection d'autres contraintes que nous avons formulées en exploitant les caractéristiques de notre problème. Pour plus d'efficacité nous les avons ensuite renforcées via la théorie polyédrale.

Proposition 3.2.1. *Soit $\mathcal{D}$ un ensemble dominant non connexe et soient $\mathcal{C}_1, \ldots, \mathcal{C}_p$ les p composantes connexes engendrées par le sous-graphe $G = (\mathcal{D}, E(\mathcal{D}))$. Pour tout $i = \{1, \ldots, p\}$ on a :*

1. *$\mathcal{C}_i$ ne constitue pas un ensemble dominant.*

2. *Tout ensemble dominant connexe contient au moins un sommet dans $\Gamma(\mathcal{C}_i)$.*

Démonstration. 1. Si une certaine composante connexe $\mathcal{C}_i$ constitue un ensemble dominant, alors les sommets de celle-ci couvrent tous les sommets du graphe, y compris les sommets d'une autre composante connexe $\mathcal{C}_j$. Il existe donc une arête reliant $\mathcal{C}_i$ et $\mathcal{C}_j$. Contradiction avec le fait que $\mathcal{C}_i$ et $\mathcal{C}_j$ soient deux composantes connexes distinctes.

2. Supposons qu'aucun sommet dans $\Gamma(\mathcal{C}_i)$ n'appartienne à un ensemble dominant connexe. Dans ce cas, la composante connexe $\mathcal{C}_i$ est isolée et ses éléments ne sont couverts que par des sommets de cette même composante.
   D'après 1, nous savons que $\mathcal{C}_i$ ne peut pas constituer, à elle seule, un ensemble dominant. Par conséquent celle-ci doit être complétée avec au moins un élément de $V \setminus (\mathcal{C}_i \cup \Gamma(\mathcal{C}_i))$. Puisque $\mathcal{C}_i$ est isolée, il ne peut exister de chemin la reliant aux autres sommets de l'ensemble dominant qui, par conséquent, ne peut être connexe.

□

De la proposition 3.2.1, il découle que les contraintes suivantes sont valides pour le problème de l'ensemble dominant connexe

$$\sum_{j \in \Gamma(\mathcal{C}_i)} x_j \geq 1, \quad \forall i \in \{1, \ldots, p\} \tag{3.2}$$

En transposant à notre problème le résultat obtenu par Cerdeira et Pinto [11], concernant les facettes du problème de recouvrement d'ensemble connexe, on déduit que pour que la contrainte 3.2 définisse une facette, les deux conditions suivantes doivent être satisfaites (Les conditions (a) et (b) correspondent respectivement aux conditions (e) et (f) dans l'article de Cerdeira et Pinto [11]).

a) $\forall v \in \Gamma(\mathcal{C}_i)$, l'ensemble $\overline{\Gamma(\mathcal{C}_i)} \cup \{v\}$ est un dominant connexe.

b) $\forall u \in \overline{\Gamma(\mathcal{C}_i)},\ \exists v \in \Gamma(\mathcal{C}_i)$ tel que $\overline{\Gamma(\mathcal{C}_i)} \setminus \{u\} \cup \{v\}$ est un dominant connexe.

La condition (a) revient à vérifier que l'ensemble $\Gamma(\mathcal{C}_i)$ est un séparateur minimal entre les deux composantes connexes $\mathcal{C}_i$ et $V \setminus (\mathcal{C}_i \cup \Gamma(\mathcal{C}_i))$. Selon Hong et Weifa [25], $\Gamma(\mathcal{C}_i)$ est un séparateur minimal si et seulement si chacun de ses éléments possède un voisin dans chacune des deux composantes connexes. Ainsi, la coupe précédente peut être renforcée en écartant de $\Gamma(\mathcal{C}_i)$, tous les sommets qui ne sont pas conformes à ce dernier critère.

La condition (b), quant à elle, consiste à vérifier si un élément $v \in \Gamma(\mathcal{C}_i)$, à lui seul, suffit pour que l'ensemble $\overline{\Gamma(\mathcal{C}_i)} \setminus \{u\} \cup \{v\}$ soit dominant et connexe indépendamment du sommet u.

Si pour un certain $u \in \overline{\Gamma(\mathcal{C}_i)}$ la condition b) n'est pas vérifiée la contrainte peut être renforcée comme suit

$$\sum_{j \in \Gamma(\mathcal{C}_i)} x_j + x_u \geq 2 \tag{3.3}$$

Ceci se traduit par le fait que, si on veut écarter le sommet u, on doit choisir au moins deux éléments dans $\Gamma(\mathcal{C}_i)$.

Exemple illustratif

Considérons le graphe de la figure 3.1 et supposons que le problème maître de Benders fournisse comme solution $\mathcal{D} = \{2, 4, 6, 9, 10, 11\}$. Puisque le sous-graphe induit par $\mathcal{D}$ n'est pas connexe, on est sûr de devoir y ajouter au moins un élément de $\overline{\mathcal{D}}$ =$\{$1,3,5,7,8,12,13$\}$. La contrainte suivante est donc valide

$$x_1 + x_3 + x_5 + x_7 + x_8 + x_{12} + x_{13} \geq 1 \tag{3.4}$$

Pour tirer de nouvelles coupes plus fortes que cette dernière, nous examinons chacune des composantes connexes de $\mathcal{D}$, en l'occurrence $C_1 = \{2, 4\}, C_2 = \{6, 9\}$ et $C_3 = \{10, 11\}$.

Composante connexe $\mathcal{C}_1 = \{2, 4\}$

Étant donné que l'ensemble des voisins de $\mathcal{C}_1$ est $\Gamma(\mathcal{C}_1) = \{1, 3, 5, 12\}$, la contrainte suivante est valide

$$x_1 + x_3 + x_5 + x_{12} \geq 1 \tag{3.5}$$

Les sommets 1 et 12 possèdent un voisin dans $\mathcal{C}_1 \cup \Gamma(\mathcal{C}_1) = \{1, 2, 3, 4, 5, 12\}$ mais pas dans $V \setminus (\mathcal{C}_1 \cup \Gamma(\mathcal{C}_1)) = \{6, 7, 8, 9, 10, 11, 13\}$. la contrainte 3.5 peut donc être renforcée comme suit

$$x_3 + x_5 \geq 1 \tag{3.6}$$

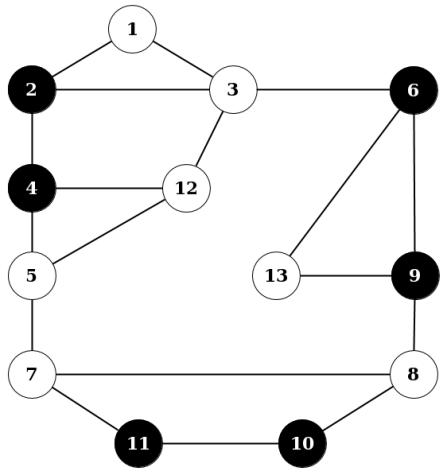


Figure 3.1 – Exemple illustratif. $\mathcal{D} = \{2, 4, 6, 9, 10, 11\}$

La condition (b) est satisfaite aussi.

Composante connexe $\mathcal{C}_2 = \{6, 9\}$

La contrainte $x_3 + x_8 + x_{13} \geq 1$ est valide mais peut etre renforcée par

$$x_3 + x_8 \geq 1 \tag{3.7}$$

La condition (b) est satisfaite aussi.

Composante connexe $\mathcal{C}_3 = \{10, 11\}$

La contrainte suivante est valide

$$x_7 + x_8 \geq 1 \tag{3.8}$$

Cette dernière satisfait a) ; par contre, elle ne satisfait pas b). En effet, pour $u = \{3\}$ aucun ensemble dominant connexe ne peut être trouvé dans $\overline{\Gamma(\mathcal{C}_3)} \setminus \{u\} \cup \{v\} = \{1, 2, 4, 5, 6, 9, 10, 11, 12, 13\} \cup \{v\}$ pour tout $v \in \{7, 8\}$. Ceci signifie que si on veut écarter le sommet 3, on doit choisir au moins deux éléments dans $\{7, 8\}$.

La contrainte suivante est donc plus forte :

$$x_7 + x_8 + x_3 \geq 2 \tag{3.9}$$

Il en est de même pour le sommet 6

$$x_7 + x_8 + x_6 \geq 2 \tag{3.10}$$

Contribution

Les coupes de Benders développées dans cette section nous ont permis de construire trois approches de résolution. L'une est basée sur la PPC et sera présentée à la section suivante. Les deux autres, la méthode de Benders pure (BEP) et la méthode de Benders avec investigation iterative (BEI) utilisent la programmation linéaire en nombres entiers. Elles résultent de l'enrichissement, par les nouvelles coupes développées plus haut, des méthodes « stand-alone » et « iterative-probing », respectivement, présentées dans Gendron et al. [20].

3.2.2 La méthode de Benders pure (BEP)

En partant de l'ensemble dominant connexe initial $\mathcal{D}^0$, obtenu suite à l'exécution de l'heuristique (HDR), cette méthode commence par résoudre le problème maître de Benders défini par les contraintes (1), (2), (4), (5), auquel on ajoute la coupe d'optimalité suivante :

$$\sum_{i \in V} x_i \leq |\mathcal{D}^0| - 1 \tag{3.11}$$

Si celui-ci est non réalisable, alors l'ensemble $\mathcal{D}^0$ fourni par l'heuristique HDR est une solution optimale. Si, par contre, une solution $\mathcal{D}$ est trouvée, on passe au traitement du sous-problème de Benders qui consiste à vérifier la connexité de cet ensemble $\mathcal{D}$ et agir en conséquence. Si ce dernier est bien connexe, alors l'algorithme s'arrête en retournant $\mathcal{D}$ comme solution optimale. Sinon, on identifie toutes ses

composantes connexes, on construit une coupe de réalisabilité associée à chacune d'entre elles, on leur applique d'éventuels renforcements en utilisant les résultats de la sous-section 3.2.1 et enfin, on les injecte dans le problème maître de Benders. On recommence la même procédure en résolvant le problème maître de Benders avec ces nouvelles coupes renforcées.

3.2.3 La méthode de Benders avec investigation iterative (BEI)

En partant de l'ensemble dominant connexe initial $\mathcal{D}^0$, obtenu suite à l'exécution de l'heuristique (HDR), cette méthode tente de trouver un CDS de cardinalité $|\mathcal{D}^0| - 1$. Si un tel ensemble n'existe pas, l'algorithme s'arrête en fournissant $\mathcal{D}^0$ comme solution optimale. Sinon, une nouvelle itération est réalisée en partant du nouvel ensemble dominant de taille $\texttt{d} = |\mathcal{D}^0| - 1$ pour tenter d'en retrouver un autre de taille `d-1`.

Pour trouver un CDS de taille `d-1`, on résout le problème maître et le sous-problème de Benders d'une manière analogue à celle de la méthode de Benders pure (BEP), mais en ajoutant au problème maître de Benders la coupe de cardinalité suivante :

$$\sum_{i \in V} x_i = \texttt{d-1} \tag{3.12}$$

3.3 Approche par la programmation par contraintes

Dans le pseudo-code de la stratégie d'investigation itérative définie dans la section 2.2.4, le problème «**2.Investigation :** Trouver un CDS de cardinalité `d` » est un problème de satisfaction de contraintes et l'utilisation de la PPC pour le résoudre est donc envisageable. Pour explorer cette piste, nous avons développé deux contraintes globales, une pour la connexité et une autre pour la dominance, ainsi qu'une stratégie de branchement basée sur les degrés résiduels des sommets. Nous les avons ensuite exploités pour développer, pour la première fois, deux approches de résolution, en l'occurrence la méthode PPC pure (PPCP) utilisant uniquement

l'approche d'investigation itérative, et la méthode PPC Benders (PPCB) combinant cette dernière avec la méthode de décomposition de Benders. Ces deux méthodes seront détaillées après la présentation des deux contraintes globales et de la stratégie de branchement.

3.3.1 Contrainte globale de connexité

Nous introduisons une nouvelle contrainte globale, `Connectivity(X,d)`, qui permet d'obtenir un sous-graphe connexe de cardinalité `d`.

Principe

Lors de la recherche d'un sous-ensemble $\mathcal{D} \subset V$ satisfaisant une certaine propriété, notamment la dominance, de cardinalité égale à `d`, nous savons qu'à chaque étape où la configuration actuelle de l'ensemble $\mathcal{D}$ est composée de `tailleActuelle` sommets, nous devons y ajouter exactement `d-tailleActuelle` sommets pour le compléter. D'un autre côté, nous savons que le nombre de Steiner, noté `NbSteiner`, représente le nombre minimum de sommets devant être ajoutés à $\mathcal{D}$ pour qu'il soit connexe. Nous pouvons donc, avec cette valeur, tirer des conclusions sur la réalisabilité du problème, en partant d'une certaine configuration de l'ensemble $\mathcal{D}$, relativement à la contrainte de connectivité. En effet, en calculant `NbSteiner`, les trois cas suivants peuvent se présenter :

1. `NbSteiner` = ∞ : Cela signifie que $\mathcal{D}$ contient au moins un sommet qui nécessite l'ajout d'un sommet écarté pour qu'il soit connexe. Dans ce cas, la configuration actuelle de $\mathcal{D}$ ne peut en aucun cas converger vers un ensemble connexe. Le problème est alors non réalisable en partant de la configuration de $\mathcal{D}$ considérée.

2. `d-tailleActuelle` < `NbSteiner` < ∞ : Cela signifie qu'il faut ajouter plus que `d-tailleActuelle` sommets pour connecter les sommets de $\mathcal{D}$. Dans ce cas, la configuration actuelle de $\mathcal{D}$ ne peut pas conduire à un ensemble connexe de cardinalité `d`. Donc le problème est non réalisable dans ce cas aussi.

3. `NbSteiner` $\leq$ `d-tailleActuelle` : Dans ce cas, la configuration actuelle de l'ensemble $\mathcal{D}$ est prometteuse et peut aboutir à un ensemble connexe de cardinalité `d`. Notons que le cas particulier `NbSteiner=0`, signifie tout simplement que la configuration actuelle de $\mathcal{D}$ est composée de sommets formant une seule composante connexe.

Estimation de la borne inférieure de Steiner

Pour des raisons pratiques, nous ne calculerons pas la valeur exacte du nombre de Steiner mais une borne inférieure, car le problème de Steiner est **NP**-complet [27]. Pour ce faire, nous orientons le graphe en remplaçant chaque arête par deux arcs de directions opposées. De plus, chacun de ces arcs se verra attribuer un coût nul si son extrémité finale fait partie de l'ensemble $\mathcal{D}$, et un coût unitaire sinon :

$$c_{ij} = \begin{cases} 0, & \text{si le sommet } j \text{ est dans l'ensemble } \mathcal{D}, \\ 1, & \text{sinon,} \end{cases}$$

comme l'illustre la figure suivante :

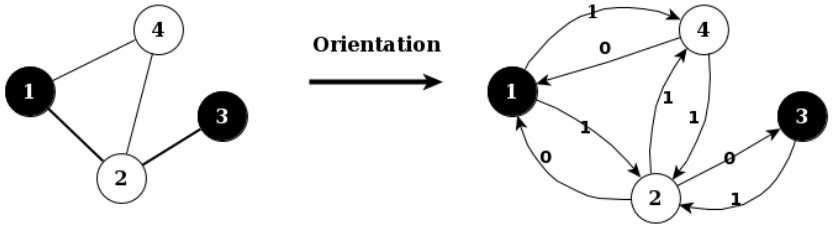


Figure 3.2 – $\mathcal{D} = \{0, 3\}$

La distance, en termes de sommets non-terminaux intermédiaires, du plus court chemin entre chaque paire de sommets terminaux équivaut au nombre minimum de sommets devant être ajoutés à $\mathcal{D}$ pour les connecter. En choisissant la plus grande des plus courtes distances séparant chaque paire de ces sommets, on obtient une borne inférieure valide du nombre de Steiner, notée InfSteiner.

Recherche des plus courts chemins

Les plus courts chemins, nécessaires pour le calcul de InfSteiner, peuvent être calculés en utilisant n'importe quel algorithme de recherche de plus courts chemins tel que l'algorithme de Djikstra, mais cela peut s'avérer coûteux en temps de calcul. Nous avons observé qu'une petite modification de l'algorithme de recherche en largeur (BFS) lui permet de retrouver ces plus courts chemins et ceci en un temps de calcul moindre. En effet, nous savons que cet algorithme permet de retrouver les plus courts chemins dans un graphe non valué qui, notons-le, équivaut à un graphe valué avec des coûts unitaires sur tous ses arcs. Donc, étant donné que notre graphe est porteur de coûts 0 ou 1, si l'on parvient à trouver un moyen de traiter les arcs de coût 0 de manière à normaliser notre graphe, le BFS redeviendra applicable et efficace pour y trouver les plus courts chemins. Pour ce faire, nous avons exploité le fait que le coût 0 sur un arc (i, j) indique que la distance minimale séparant le sommet j d'un autre sommet k est égale à celle séparant le sommet i de ce dernier. Par conséquent, lors de la recherche, nous pouvons passer aux sommets appartenant à $\mathcal{D}$ sans modification du coût de parcours. Cela permet d'éviter le traitement des arcs de coût 0 qui posaient problème sans pour autant perdre d'informations pertinentes vu que, comme expliqué précédemment, les distances séparant des sommets accessibles à coût nul des autres sommets reste la même. Ainsi, en se basant sur le BFS et en adoptant cette petite modification, nous avons pu construire une procédure permettant de trouver les longueurs des plus courts chemins reliant un sommet donné à tous les autre sommets du graphe.

Pour illustrer le fonctionnement de cette procédure, considérons le graphe représenté à la figure 3.3 à une étape où les sommets représentés en noir ont déjà été choisis pour faire partie de l'ensemble $\mathcal{D}$ recherché.

Supposons que l'on veuille retrouver les plus courtes distances séparant le sommet 1 de tous les autres sommets. Après avoir orienté et valué le graphe, comme expliqué plus haut, on commence par marquer le sommet 1 en lui affectant la dis-

tance *0*, puisque la distance le séparant de lui-même est nulle, puis on commence son traitement.

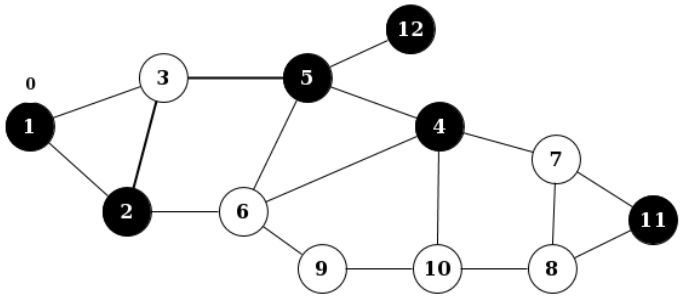


Figure 3.3 – $T = \{1\}$

On marque tous les voisins non encore marqués du sommet 1, et on affecte la distance du sommet en cours de traitement à ceux qui en sont accessibles à coût nul (le sommet 2), et la distance du sommet en cours de traitement *+1* à ceux qui en sont accessibles par un arc de valeur unitaire (le sommet 3), comme indiqué dans la figure 3.4. Ainsi, les sommets candidats au traitement à la prochaine étape, c'est-à-dire ceux marqués non encore traités, sont les sommets 2 et 3 (les sommets candidats au traitement à chaque étape seront listés dans les légendes des figures qui accompagneront les commentaires des étapes suivantes).

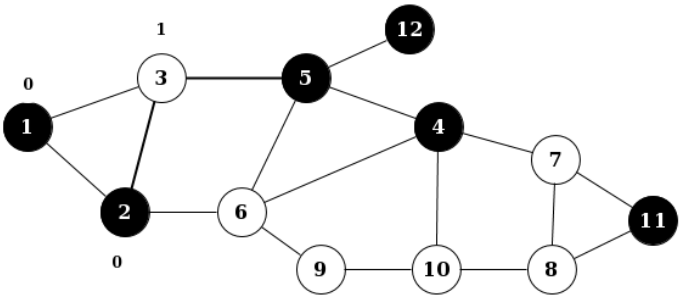


Figure 3.4 – $T = \{2, 3\}$

Sachant que les sommets appartenant à $\mathcal{D}$ sont prioritaires, le prochain sommet à traiter à chaque étape est le premier sommet marqué non encore traité du graphe. Donc, on passe au traitement du sommet 2, on marque tous ses voisins

non encore marqués et on leur affecte leurs distances respectives comme expliqué précédemment, en l'occurrence la distance *1* au sommet 6, comme indiqué à la FIGURE 3.5.

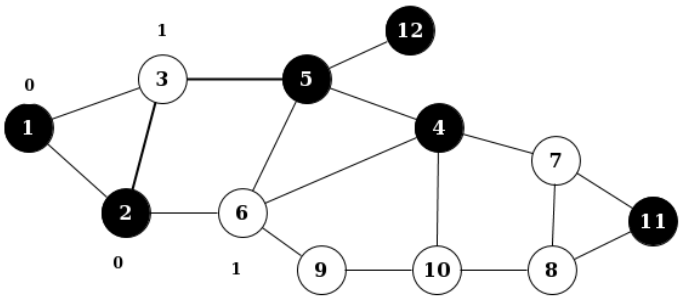


Figure 3.5 – $T = \{3, 6\}$

Le prochain sommet à traiter est le sommet 3. Le seul voisin non encore marqué de celui-ci étant le sommet 5, on le marque et on lui affecte la distance *1*, qui correspond à la distance du sommet en cours de traitement car il appartient à $\mathcal{D}$, comme illustré à la FIGURE 3.6.

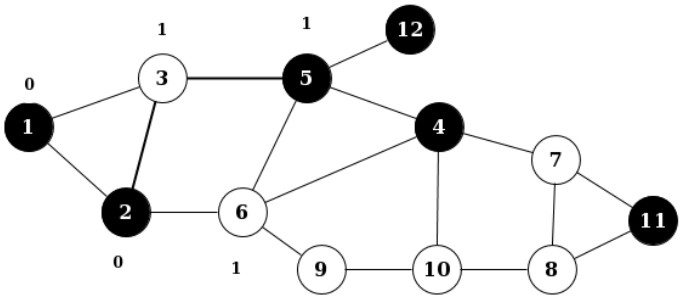


Figure 3.6 – $T = \{5, 6\}$

À cette étape les sommets candidats au traitement sont les sommets 5 et 6. Malgré que le sommet 6 ait été marqué avant le 5 on passe au traitement de ce dernier, car celui-ci est prioritaire puisqu'il appartient à $\mathcal{D}$. On marque alors tous ses voisins non encore marqués (4,12) et on leur affecte la distance *1* du sommet en cours de traitement car ils appartiennent tous les deux à $\mathcal{D}$, comme l'illustre la FIGURE 3.7.

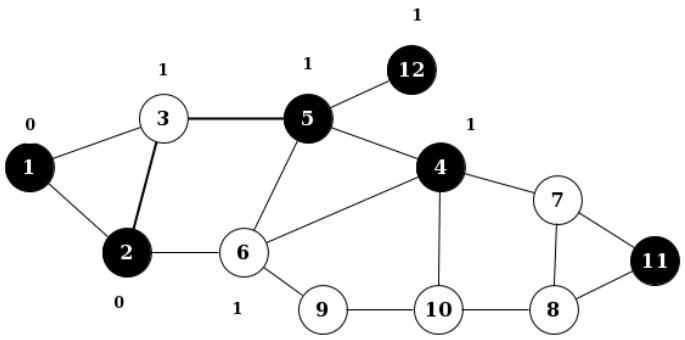


Figure 3.7 – $T = \{4, 12, 6\}$

Le prochain sommet à traiter est le sommet 4. Celui-ci possède deux voisins non encore traités (les sommets 7 et 10), on les marque et on leur affecte la distance *2* (distance du sommet en cours de traitement + *1*), comme illustré dans la figure 3.8.

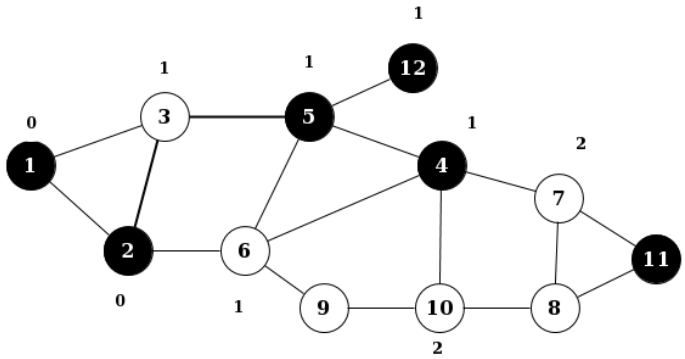


Figure 3.8 – $T = \{12, 6, 7, 10\}$

Le prochain sommet à traiter étant le sommet 12 qui ne possède aucun voisin non encore marqué, on passe au traitement du prochain qui est le sommet 6. Celui-ci a le sommet 9 comme seul voisin non encore marqué, alors on le marque et on lui affecte la distance *2*, comme l'illustre la figure 3.9.

On passe au traitement du sommet 7 qui possède deux voisins non encore marqués (les sommets 8 et 11). On les marque et on affecte la distance *2* au sommet 11, car il appartient à $\mathcal{D}$, et la distance *3* au sommet 8, car il n'appartient pas à

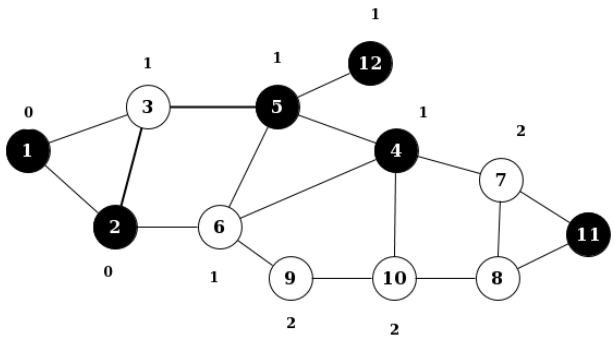


Figure 3.9 – $T = \{7, 10, 9\}$

$\mathcal{D}$, comme l'illustre la figure 3.10.

À ce stade du déroulement de la procédure, tous les sommets ont été marqués. Par conséquent, le traitement des sommets non encore traités n'engendrera aucun changement. On peut alors arrêter la procédure et passer à la récupération des résultats.

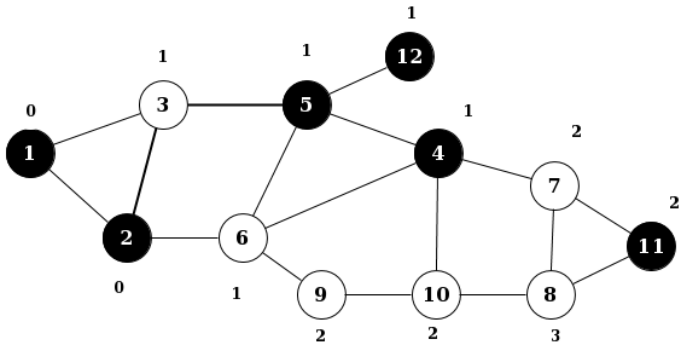


Figure 3.10 – $T = \{11, 10, 9, 8\}$

À l'issue de cette procédure, tous les sommets du graphe se retrouvent porteurs de valeurs représentant la longueur du plus court chemin reliant chacun d'entre eux au sommet 1.

Ajout du filtrage

Si on cherche un CDS de taille `d` en supposant que $\mathcal{D}$ contient présentement `tailleActuelle` sommets, alors pour qu'un sommet `s` puisse faire partie de $\mathcal{D}$, il faut que sa plus courte distance (en terme de nombre de sommets non dominants intermédiaires) à $\mathcal{D}$ soit au plus égale à `d-tailleActuelle`. En effet, si cette distance est égale à un certain `x` > `d-tailleActuelle`, il faudra ajouter à $\mathcal{D}$, en plus de ce sommet en question, au moins `x` autres sommets pour garantir sa connexité. Ceci conduit à un ensemble $\mathcal{D}$ de cardinalité `tailleActuelle+x` supérieure à `d`.

Reprenons, à titre d'exemple, le graphe de la figure 3.10, et la configuration de $\mathcal{D}$ qui y est présentée. On remarque que la plus courte distance séparant le sommet 7 de la composante connexe constituée des sommets 0 et 1 est égale à 3. Étant donné qu'on cherche un CDS de cardinalité `8` et que D contient déjà 6 sommets à cette étape, la `tailleResiduelle=d-tailleActuelle` vaut 2 et elle inférieure à la distance entre la composante connexe et le sommet 7. Ce dernier doit, par conséquent, être écarté.

Propagation de la contrainte

À chaque instanciation d'une variable x_i à 1, le cardinal actuel, `tailleActuelle`, de l'ensemble dominant $\mathcal{D}$ est incrémenté de 1, et le sommet associé à cette variable est choisi comme sommet `source`. Les plus courtes distances séparant ce sommet source de tous les autres sommets du graphe sont, ensuite, calculées en utilisant la fonction `ShortestPaths#(source)` qui, par la même occasion, détecte toutes les composantes connexes de $\mathcal{D}$ et en sauvegarde un élément de chacune dans une liste `CC`. La fonction ShortestPaths(v) est ensuite utilisée à partir de chaque élément `v` de la liste `CC` pour calculer les plus courtes distances séparant les sommets du graphe de la composante connexe contenant le sommet v et permettre d'écarter ceux qui en sont trop éloignés. Si la distance entre une paire de composantes connexes excède `d-tailleActuelle`, alors le problème est non réalisable en partant de la configuration de $\mathcal{D}$ considérée.

3.3.2 Contrainte globale de dominance

Pour construire notre contrainte de dominance, nous nous basons sur une propriété inspirée d'un théorème sur la borne inférieure de la cardinalité minimale d'un ensemble dominant, extrait de Haynes et al. [24], qui s'énonce comme suit :

Théorème 3.3.1. *Si le graphe G possède une séquence de degrés $(d_1, d_2, ..., d_n)$ avec $d_i \geq d_{i+1}$, alors $\gamma(G) \geq min\{k : k + (d_1 + d_2 + ... + d_k) \geq n\}$.*

Ce théorème stipule que, sachant que les sommets du graphe G sont rangés par ordre de degrés décroissants, l'indice du sommet à partir duquel la somme des degrés des sommets le précédant (le sien inclus) dépasse le nombre de sommets du graphe, constitue une borne inférieure pour la cardinalité $\gamma(G)$ de l'ensemble dominant minimum de ce graphe.

Adaptation de ce théorème au cas de TDS

La différence entre la situation que couvre le théorème présenté précédemment et la nôtre, réside dans le fait que ce théorème est énoncé pour le cas où un sommet peut se couvrir lui-même, alors que dans notre cas cette situation ne peut se produire que lorsqu'un sommet, à lui seul, suffit pour couvrir tous les sommets du graphe. Sinon, chaque sommet doit impérativement être couvert par un de ses voisins.

Pour adapter ce théorème au cas de l'ensemble dominant total, il suffit de prendre en considération ce détail en n'incluant dans le calcul que les degrés de nos sommets sans compter les sommets eux-mêmes. On obtient ainsi le théorème suivant :

Théorème 3.3.2. *Si le graphe G possède une séquence de degrés $(d_1, d_2, ..., d_n)$ avec $d_i \geq d_{i+1}$, alors $\gamma_t(G) \geq min\{k : d_1 + d_2 + ... + d_k \geq n\}$.*

tel que $\gamma_t(G)$ est la taille de l'ensemble dominant total minimum.
On peut conclure que ce théorème peut être aussi utile pour déterminer le nombre minimum de sommets à ajouter à une certaine configuration préalablement connue

de l'ensemble $\mathcal{D}$ pour qu'il soit dominant. En effet, il suffit de considérer uniquement les sommets non encore fixés et d'utiliser, au lieu de leurs degrés initiaux, leurs degrés résiduels (le nombre de voisins non encore couverts), et comparer la somme au nombre total de sommets non encore couverts au lieu du nombre de sommets du graphe.

Exemple

Considérons le graphe de la figure 3.11. Les degrés initiaux de ses sommets sont donnés dans la seconde ligne du tableau 3.I

Supposons que l'on cherche un ensemble dominant de taille 2 et que le sommet 1 en fasse partie. Ce dernier couvre ses deux voisins. Le nombre de sommets couverts est alors `nbCouvert=2`, et il reste 5 autres sommets à couvrir.

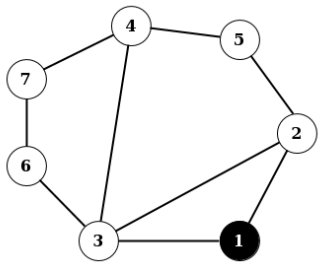


Figure 3.11 – Degrés résiduels des sommets

Sommets	1	2	3	4	5	6	7
Degrés initiaux (d)	2	3	4	3	2	2	2
Degrés résiduels (r)	0	2	3	2	1	1	2

Tableau 3.I – Degrés résiduels des sommets

Sommets	3	2	4	7	5	6	1
Degrés résiduels ordonnés	3	2	2	2	1	1	0

Tableau 3.II – Degrés des sommets par ordre décroissant

Une borne inférieure du nombre minimum de sommets à ajouter pour couvrir tous les sommets du graphe est donnée par $min\{k : r_1 + r_2 + ... + r_k \geq 5\} = 2$.

En effet, les degrés résiduels des sommets après le choix du sommet 1 sont donnés dans la troisième ligne du tableau 3.I. On remarque qu'en ordonnant les sommets par ordre décroissant des degrés résiduels, tel que présenté dans le tableau 3.II, leur somme ne dépassera 5 qu'après l'ajout des deux sommets 3 et 2.

On peut donc conclure que cette configuration ne peut jamais aboutir à un ensemble dominant de cardinalité 2, car en plus du sommet 1 déjà introduit dans $\mathcal{D}$, au moins deux autres sommets doivent y être ajoutés, ramenant ainsi la taille de ce dernier à 3 (alors qu'elle ne devait pas dépasser 2).

Intérêt de ce théorème pour notre problème

Ce théorème peut servir lors de la propagation à tester la réalisabilité du problème en partant d'une certaine configuration de l'ensemble $\mathcal{D}$, relativement à la contrainte de dominance. En effet, à chaque instanciation d'une variable x_i à 1, celle-ci couvrira tous ses voisins qui ne sont pas encore couverts, ce qui laisse à chacun des sommets non encore fixés, un certain nombre de sommets qu'ils sont susceptibles de couvrir, si jamais ils sont choisis, qu'on appellera degré résiduel et qu'on notera r_j pour le sommet `j`. Si l'on désigne par `nbCouverts` le nombre de sommets couverts par la configuration actuelle de l'ensemble $\mathcal{D}$ et `tailleActuelle` le nombre de sommets qu'elle contient, on sait que les `tailleResiduelle=d-tailleActuelle` sommets devant compléter l'ensemble $\mathcal{D}$ doivent couvrir à eux seuls tous les sommets non encore couverts du graphe. Donc, si l'on ordonne les degrés résiduels des sommets non encore fixés du graphe, il faudra que

$$\min\{k : r_1 + r_2 + ... + r_k \geq \texttt{n-nbCouverts}\} \leq \texttt{d-tailleActuelle}$$

Ceci est équivalent à dire que

$$r_1 + r_2 + ... + r_{d-tailleActuelle} \geq \texttt{n-nbCouverts}$$

Donc le fait que la somme des degrés résiduels des `tailleResiduelle` premiers sommets ordonnés n'atteint pas `nbNonCouverts=n-nbCouverts`, suffit pour conclure à la non réalisabilité du problème et ainsi définitivement écarter la configuration de $\mathcal{D}$ considérée.

Propagation de la contrainte

À chaque instanciation d'une variable x_i, les degrés résiduels des sommets sont mis à jour de façon incrémentielle. Si la variable x_i est fixée à 0, le sommet i est écarté, ceci signifie que ce dernier ne pourra plus couvrir aucun autre sommet et son degré résiduel est alors réduit à 0. Par contre, lorsqu'elle est fixée à 1, le sommet i est ajouté à $\mathcal{D}$ et couvre ainsi ses voisins non encore couverts ; `nbCouverts` est alors augmenté du degré résiduel de i et les degrés résiduels associés aux sommets susceptibles de couvrir ces sommets nouvellement couverts sont alors tous décrémentés de 1.

Une fois tous les degrés résiduels ainsi mis à jour, on ordonne les sommets par ordre décroissant de leurs nouveaux degrés résiduels et on calcule la somme des `d-tailleActuelle` premiers sommets ainsi ordonnés. Si cette somme n'atteint pas `n-nbCouverts` alors la configuration actuelle de $\mathcal{D}$ est rejetée, car elle ne permet pas d'aboutir à un ensemble dominant connexe de taille `d`. Si, par contre, cette valeur est atteinte, on ajoute du filtrage à la procédure pour identifier les sommets devant impérativement appartenir à $\mathcal{D}$ et ceux ne devant absolument pas être choisis. Pour ce faire, on teste, dans un premier temps, la nécessité des `d-tailleActuelle` premiers sommets ordonnés en réalisant le même test précédent, mais en excluant chaque élément de ces derniers à tour de rôle. Les sommets du graphe étant préalablement rangés par ordre décroissant de leurs degrés résiduels, si on suppose que l'un des `d-tailleActuelle` premiers sommets est écarté, le test se fera en le remplaçant par le `d-tailleActuelle+1`$^{\text{ieme}}$ sommet, et la valeur de la borne sera alors obtenue en retranchant le degré résiduel du sommet supposé écarté de la somme des `d-tailleActuelle+1` premiers sommets. Il suffit alors de vérifier si cette borne est inférieure à `n-nbCouvert` pour affirmer que le sommet en

question est indispensable pour la formation d'un ensemble $\mathcal{D}$ conforme aux conditions du problème. Dans un second temps, on teste la présence d'éventuels sommets inutiles parmi ceux restants en réalisant toujours le même test, mais cette fois-ci en supposant leur appartenance à $\mathcal{D}$. Dans ce cas, si l'on suppose qu'un certain sommet de rang supérieur ou égal à `d-tailleActuelle` appartienne à $\mathcal{D}$, le test précédent doit être réalisé avec une borne égale à la somme des degrés résiduels des `d-tailleActuelle-1` premiers sommets et celui du sommet testé. Si cette borne dépasse `n-nbCouvert`, ce sommet, ainsi que tous ceux ayant un rang supérieur au sien, peuvent être définitivement écartés.

Exemple

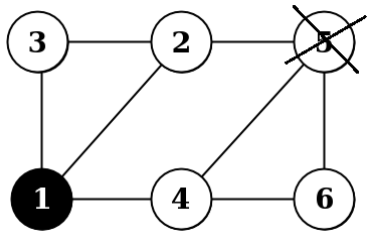


Figure 3.12 – Propagation de la contrainte de dominance

Supposons que l'on cherche, dans le graphe de la figure 3.12, un CDS de cardinalité 2. Initialement, les degrés résiduels de tous les sommets coïncident avec leurs degrés respectifs dans le graphe, comme indiqué au tableau 3.III.

Sommets	1	2	3	4	5	6	`nbCouvert`
Degrés initiaux (d)	3	3	2	3	3	2	0

Tableau 3.III – Degrés initiaux des sommets du graphe

Supposons qu'à une certaine étape le sommet 5 soit exclu et que le sommet 1 soit choisi pour appartenir à $\mathcal{D}$. Dans ce cas, le degré résiduel du sommet 5 est réduit à 0, `tailleActuelle` est mis à jour à 1, les degrés résiduels de tous les sommets

susceptibles de couvrir les sommets nouvellement couverts par 1 (les voisins des voisins) sont décrémentés de 1 et `nbCouvert` est augmenté de 3 (degré résiduel du sommet `1`), comme indiqué au tableau 3.IV.

Sommets	1	2	3	4	5	6	`nbCouvert`
Degrés résiduels (r)	0	2	1	3	0	1	3

Tableau 3.IV – Degrés résiduels des sommets du graphe

On classe ensuite les sommets non encore fixés par ordre décroissant de degrés résiduels, tel qu'indiqué dans le tableau 3.V, puis on somme les degrés résiduels des `d-tailleActuelle` premiers sommets ainsi ordonnés.

Sommets	4	2	3	6	`n-nbCouvert`
Degrés résiduels (r)	3	2	1	1	3

Tableau 3.V – Degrés résiduels des sommets non encore fixés

Puisque `d-tailleActuelle` est égale à 1 dans notre cas, il suffit de vérifier le degré résiduel du premier sommet, celui ayant le degré résiduel le plus élevé. Celui-ci est égal à 3 dans notre exemple et il coïncide avec `n-nbCouverts=3`. On peut donc conclure que la configuration actuelle de $\mathcal{D}$ est prometteuse. On peut alors passer à la recherche d'éventuels filtrages additionnels.

Commençons par vérifier la nécessité de l'appartenance du sommet 4 à $\mathcal{D}$. Pour ce faire, on suppose que ce dernier en est écarté, et on calcule la nouvelle borne dont la valeur équivaut à celle de l'ancienne de laquelle on retranche le degré résiduel du sommet supposé écarté (sommet 4) et on y ajoute celui du sommet qui le succède dans le tableau trié 3.V, en l'occurrence le sommet 2. Dans ce cas, la nouvelle borne vaut `3-3+2=2` et est strictement inférieure à `n-nbCouverts` qui vaut 3, ce qui indique que le problème devient non réalisable lorsque le sommet 4 est écarté ; on peut alors déduire que ce dernier doit impérativement faire partie de l'ensemble dominant $\mathcal{D}$.

On passe ensuite à la vérification de la possibilité d'appartenance du sommet 2 à $\mathcal{D}$. Pour cela, on calcule la valeur de la borne dans le cas où ce sommet est admis

dans $\mathcal{D}$, en ajoutant le degré résiduel du sommet testé à l'ancienne borne après avoir retranché celui du sommet le précédant dans le tableau trié 3.V, c'est-à-dire celui du sommet 4. On constate alors que la nouvelle borne ainsi calculée vaut 2 et est inférieure à `n-nbCouverts`. On déduit la non-réalisabilité du problème lorsque le sommet 2 est choisi pour appartenir à $\mathcal{D}$.

De plus, étant donné que le tableau 3.V est préalablement trié, on peut écarter, sans vérification, tous les sommets succédant le premier sommet écarté. Dans le cas de notre exemple, étant donné qu'on a vérifié que le sommet 2 doit être écarté, pour que la configuration actuelle de $\mathcal{D}$ aboutisse à une solution réalisable, on peut déduire la nécessité d'écarter les sommets 3 et 6 aussi.

3.3.3 Heuristique de branchement

Notre heuristique de branchement consiste à choisir de brancher sur la variable associée au sommet ayant le plus grand degré résiduel. Ce choix est justifié par le fait que le degré résiduel d'un sommet, à une certaine étape, représente le nombre de sommets étant nouvellement couverts si ce sommet est choisi à cette étape. En choisissant ceux ayant le plus grand degré résiduel, on garantit, à chaque étape, de couvrir la plus grande partie possible du graphe, progressant ainsi plus rapidement vers la solution escomptée.

3.3.4 La méthode PPC Benders (PPCB)

Cette méthode est analogue à celle de Benders avec investigation itérative (BEI) présentée plus haut, mais celle-ci utilise, pour la résolution du problème maître de Benders, la programmation par contraintes au lieu de la programmation en nombres entiers. La démarche générale qui y est utilisée reste la même que celle de la méthode de (BEI). Cependant, l'usage de la PPC nous permet d'introduire plus de filtrage en ajoutant au problème maître la contrainte globale de dominance précédemment définie. Dans cette méthode, la recherche est guidée selon l'heuristique de recherche présentée dans la section 3.3.3.

3.3.5 La méthode PPC pure (PPCP)

Cette méthode utilise la stratégie d'investigation itérative et exploite les deux contraintes globales définies plus haut, en l'occurrence la contrainte globale de connexité et celle de dominance.

En partant d'une solution de taille `d`, on essaye de trouver un CDS de taille `d-1`. Pour ce faire, on résout le problème défini par les contraintes (2), (4), (5) et (3.12), en y ajoutant les contraintes globales de dominance et de connexité et en guidant la recherche suivant l'heuristique de recherche de la section 3.3.3.

CHAPITRE 4

RÉSULTATS

4.1 Exemplaires

Nous avons réalisé des exécutions de chacun des algorithmes que nous avons proposés et présentés dans le chapitre précédent sur les 41 instances utilisées dans Lucena et al. [29]. Elles sont associées à des graphes de densité allant de 5% à 70% avec un nombre de sommets appartenant à {30,50,70,100,120,150,200}. Dans ce qui suit, la notation de la forme vN_dM sera utilisée pour désigner les instances associées à des graphes de N sommets avec une densité de M %.

4.2 Environnement de développement

Nous avons implémenté tous nos algorithmes en langage C++ en utilisant les librairies CPLEX et CP OPTIMIZER 12.6, et toutes les exécutions de ceux-ci sur toutes les instances ont été réalisées dans un environnement Linux sur la même machine ayant un processeur Intel Xeon X5675 @3.07 Ghz avec 96 Go de mémoire vive.

4.3 Heuristique HDR

Nous avons réalisé une étude comparative entre les trois heuristiques citées précédemment, en l'occurrence l'heuristique gloutonne dynamique renforcée (HDR), celle utilisée dans l'article de Gendron et al. [20] (HDM), et celle utilisée dans l'article de Lucena et al. [29] (HD). Pour ce faire, nous avons réalisé des exécutions de chacune de ces heuristiques sur 41 instances du problème. Nous avons ensuite recueilli et calculé pour chacune des méthodes testées, quelques données statistiques pertinentes pour la comparaison de ces méthodes, en l'occurrence la déviation moyenne par rapport à l'optimum en pourcentage et en unités (Gap moy(%)

et Gap moy(U)), la déviation maximale par rapport à l'optimum en pourcentage et en unités (Gap max(%) et Gap max(U)), le nombre d'instances sur lesquelles chaque méthode a retourné la solution optimale (#Opt), le nombre d'instances sur lesquelles chaque méthode a retourné la meilleure approximation (#Best), et enfin le temps d'exécution moyen sur machine en secondes (#CPU). Ces données sont résumées dans le tableau 4.I.

	GAP				#Opt	#Best	CPU (sec)
	Moy		Max				
	(%)	(U)	(%)	(U)			
HD	9.56	1.10	50	7	19	19	0.01
HDM	2.20	0.32	25	2	31	34	0.04
HDR	**0.73**	**0.15**	**7.69**	**1**	**35**	**41**	0.11

Tableau 4.I – Résultats d'exécutions des heuristiques HD, HDM et HDR

Les résultats présentés au tableau 4.I montrent que l'heuristique HDR est nettement plus performante que les deux autres. En effet, cette heuristique a retourné la meilleure approximation de la solution optimale pour toutes les instances utilisées lors des tests. De plus, sur les 41 exécutions réalisées, la solution optimale a été atteinte 35 fois par celle-ci contre 31 fois pour HDM et 19 fois pour HD. On remarque aussi que les déviations moyennes et maximales par rapport aux solutions optimales sont sensiblement plus faibles avec cette heuristique. Elles ont, en effet, été réduites de presque 2 fois par rapport à l'heuristique de Gendron et al. [20], tout ceci en un temps d'exécution moyen de 0.11s.

4.4 Algorithmes exacts

Au tableau 4.II sont résumés les temps d'exécution des approches de résolutions que nous avons développées, ainsi que quelques-unes des meilleures méthodes proposées dans la littérature. Dans les deux premières colonnes de ce tableau sont données les références des instances testées et la valeur optimale de chacune. Les quatre colonnes qui suivent concernent les méthodes que nous avons développées,

en l'occurrence BEP, BEI, PPCB et PPCP dans cet ordre. Elles sont suivies de quatre autres colonnes qui concernent les méthodes stand-alone Benders decomposition (SABE), iterative probing Benders decomposition (IPBE), stand-alone branch-and-cut (SABC) et iterative probing branch-and-cut (IPBC) tirées de Gendron et al. [20]. Enfin, la dernière colonne correspond à la méthode DGR tirée de Lucena et al. [29]. Celle-ci a été initialement conçue pour le problème MLSTP, mais peut être utilisée pour la résolution de notre problème, tel qu'expliqué à la sous-section 2.1.1. Notons que nous avons nous-mêmes implémenté les méthodes SABE et IPBE de Gendron et al. [20], et les avons exécutées dans notre environnement. Nous avons aussi exploité, pour la recherche de la borne inférieure de Steiner, la procédure décrite à la sous-section 3.3.1. Quant aux méthodes SABC et IPBC des mêmes auteurs, elles ont été codées en langage C et exécutées sur une machine ayant un processeur XEON de 2Ghz avec 8 Gbytes de mémoire vive. La méthode DGR de Lucena et al. [29] a été exécutée sur une machine Intel XEON X5472 de 3.00 GHz avec 16Gbytes de mémoire vive. Le caractère ″-″ est utilisé lorsqu'une instance n'a pas pu être résolue par un algorithme en moins de 3600 secondes.

L'observation du dernier bloc de données dans le tableau 4.II (les trois dernières colonnes) montre que les méthodes SABC, IPBC et DGR, basées sur le branch-and-cut, sont efficaces pour les instances de petite taille, mais deviennent de moins en moins performantes à mesure que les tailles des instances augmentent. On remarque, en effet, que les temps d'exécution deviennent longs à partir des instances de taille 70 et que certaines instances de taille dépassant 150 n'ont pas pu être résolues.

Les deux colonnes qui précèdent celles-ci montrent que les méthodes SABE et IPBE, basées sur la méthode de décomposition de Benders, sont plus efficaces que les précédentes pour le traitement des graphes de grande taille, mais rencontrent des difficultés lorsque les graphes sont de densité réduite. On remarque notamment que ces méthodes échouent à résoudre des instances de densité 5 ou alors le temps d'exécution est très long dans le cas où la solution optimale est atteinte.

Ces résultats montrent que les méthodes existantes dans la littérature pour

la résolution du MCDSP rencontrent certaines difficultés relatives à la taille des graphes et/ou à leur densité. C'est pour surmonter ces difficultés que nous avons entrepris d'en améliorer certaines ou de développer de nouvelles méthodes de résolution en nous basant sur la programmation par contraintes et la programmation en nombres entiers. Les résultats exposés dans les blocs ″nos méthodes″ du tableau 4.II montrent que nous avons assez bien atteint notre objectif, puisque les résultats obtenus par les méthodes connues dans la littérature ont été améliorés par les nôtres. En effet, les méthodes BEP et BEI, que nous avons développées en nous basant sur la méthode de décomposition de Benders en programmation en nombres entiers et qui peuvent être considérées comme les plus performantes, ont pu résoudre 40 des 41 instances testées avec des temps d'exécution globalement satisfaisants. La seule instance non résolue par ces deux méthodes n'a été résolue par aucune autre méthode.

Les méthodes PPCP et PPCB que nous avons développées en nous basant sur la PPC, malgré qu'elles ne soient pas toujours efficaces pour les graphes de faible densité, se sont avérées être efficaces pour la résolution des problèmes impliquant des graphes de très grande taille. Elles fournissent, en effet, les meilleurs résultats pour les instances de taille 200 avec une densité de 30 et plus, avec des temps d'exécution sensiblement moindres que ceux des autres méthodes, comme on peut le constater aux trois dernières lignes du tableau 4.II.

Pour analyser plus en détails nos méthodes, nous exploitons les données du tableau 4.III dans lequel sont résumés, pour les méthodes basées sur la décomposition de Benders, le nombre de problèmes maîtres résolus (*iter*) et le nombre de coupes ajoutées au problème pour la résolution de chaque instance testée. Ces données fournissent une explication pour la réduction du temps d'exécution observé lors de l'utilisation de nos méthodes. On remarque, en effet, que les méthodes BEP et BEI, qui présentent les temps d'exécutions les plus courts, résolvent la plupart des instances de densité 20 ou plus, quelque soit leur taille, au bout d'une seule itération et sans l'ajout d'aucune coupe additionnelle. Seules trois instances parmi

celles-ci ont nécessité pour leur résolution au plus trois itérations et quatre coupes additionnelles. On remarque cependant que ces données sont assez proches de celles des autres méthodes pour ces instances, mais que les temps d'exécution sont différents. Ceci peut être expliqué par la qualité et l'efficacité des coupes utilisées dans chacune de ces méthodes. Les instances de faible densité nécessitent, quant à elles, un nombre d'itérations et de coupes additionnelles nettement moindres avec les méthodes BEI et BEP comparativement aux méthodes SABE et IPBE. Ceci explique la différence marquante entre les temps d'exécution pour ces instances. Les nombres d'itérations et de coupes nécessaires pour la méthode PPCB, bien que légèrement plus importants que ceux des méthodes BEI et BEP, restent loin de ceux des méthodes SABE et IPBE, ce qui confirme les conclusions basées sur les temps d'exécution.

Finalement, nous avons étudié l'impact de l'heuristique d'initialisation utilisée dans ces méthodes sur les performances de celles-ci. Pour ce faire, nous avons réalisé quelques exécutions de la méthode BEI sur les instances utilisées précédemment, mais sans lui fournir de borne supérieure (la solution de départ contient tous les sommets du graphe). Les résultats de ces tests sont donnés dans le tableau 4.IV. Contrairement à nos attentes, ces résultats montrent que l'étape d'initialisation de l'algorithme n'est pas aussi cruciale qu'on l'aurait pensé. On remarque effectivement qu'il n'y a pas de grandes différences entre les temps d'exécution de la méthode en utilisant ou non l'heuristique d'initialisation. On remarque même la présence de certains cas où la méthode fournit de meilleurs résultats lorsqu'elle est utilisée sans cette heuristique ; c'est le cas des instances $v150\_d20, v200\_d5$ et $v200\_d20$. Bien que l'initialisation de la méthode puisse sembler important, puisque le fait de partir d'une solution proche de l'optimum semble intuitivement favoriser l'atteinte de ce dernier plus rapidement, la constatation du contraire dans certains cas peut être expliquée par le fait que, partir d'une solution plus éloignée permet d'ajouter, au cours du processus de résolution, plus de coupes valides. Ceci permet d'accumuler des coupes de meilleure qualité qui conduisent à une résolution plus efficace et rapide du problème. Il semble donc qu'il serait intéressant d'identifier les meilleures

coupes et les ajouter au problème avant même de faire appel au solveur. Ceci ouvre une perspective d'amélioration de ce travail qui consiste à rechercher de mécanismes et méthodes permettant d'identifier rapidement de telles coupes.

Instances	Opt	Nos méthodes				Littérature				
		MIP		PPC		Même machine		Autres machines		
		BEP	BEI	PPCP	PPCB	SABE	IPBE	SABC	IPBC	DGR
v30_d10	15	0.04	0.03	0.02	0.06	373.65	384.01	0.03	0.02	**0.01**
v30_d20	7	0.01	0.01	**0**	**0**	**0**	0.01	0.02	0.02	0.10
v30_d30	4	**0**	**0**	**0**	0.01	**0**	**0**	0.05	0.06	0.03
v30_d50	3	**0**	**0**	**0**	**0**	**0**	0.01	0.01	0.03	0.08
v30_d70	2	**0**	**0**	0.01	0.01	**0**	**0**	0.02	**0**	0.01
v50_d5	31	1.51	0.58	9.55	41.66	-	-	**0.01**	0.02	**0.01**
v50_d10	12	0.06	**0.04**	0.07	0.05	0.05	0.07	0.82	0.20	0.36
v50_d20	7	0.04	**0.03**	**0.03**	0.05	0.04	**0.03**	0.77	0.97	1.32
v50_d30	5	0.05	0.04	**0.02**	**0.02**	0.05	0.04	0.32	0.25	1.21
v50_d50	3	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**	0.23	0.06	0.51
v50_d70	2	0.02	0.02	0.02	0.02	0.02	0.02	0.24	0.01	0.04
v70_d5	27	2.57	1.38	2100.54	–	-	-	2.06	0.39	**0.26**
v70_d10	13	0.07	**0.06**	1.28	0.35	0.07	0.07	18.68	5.25	4.73
v70_d20	7	0.11	0.08	0.12	0.08	0.11	**0.07**	2.68	1.88	16.30
v70_d30	5	0.11	0.1	0.05	**0.04**	0.1	0.1	1.20	0.99	2.90
v70_d50	3	**0.05**	**0.05**	**0.05**	0.06	**0.05**	**0.05**	0.64	0.40	1.33
v70_d70	2	0.06	0.06	0.06	0.06	0.069	0.06	0.99	**0.04**	1.92
v100_d5	24	1.66	**1.24**	913.01	428.91	116.44	117.77	58.77	64.13	12.50
v100_d10	13	**0.17**	0.22	10.95	3.15	**0.17**	0.21	28.25	39.71	9.36
v100_d20	8	**0.58**	0.75	4.14	1.86	**0.58**	0.75	283.23	414.49	86.16
v100_d30	6	1.11	1.37	1.58	**0.89**	1.14	1.37	329.05	638.89	258.15
v100_d50	4	0.5	0.57	0.39	**0.31**	0.54	0.58	48.00	41.51	132.55
v100_d70	3	0.52	0.48	0.23	**0.22**	0.52	0.45	13.20	12.02	154.10
v120_d5	25	**0.95**	1.62	-	-	6.42	1.84	1465.05	199.01	2.65
v120_d10	13	**0.26**	2.68	622.2	154.14	0.27	1.65	-	-	65.49
v120_d20	8	**1.39**	2.1	10.31	4.72	1.36	2.13	1316.70	-	393.47
v120_d30	6	1.75	2.12	3.02	**1.74**	**1.74**	2.13	790.91	1913.36	653.70
v120_d50	4	1.42	1.43	0.75	**0.59**	1.44	1.4	246.93	202.30	815.64
v120_d70	3	0.83	0.81	**0.38**	**0.38**	0.83	0.81	36.84	28.90	356.31
v150_d5	26	**69.11**	79.8	-	–	-	550.61	-	-	2954.00
v150_d10	14	37.97	30.56	-	-	**28.63**	58.96	-	-	3247.89
v150_d20	9	151.34	189	565.92	645.52	111.87	**148.11**	-	-	-
v150_d30	6	4.35	6.66	6.68	**3.86**	4.1	6.63	2972.83	-	2317.35
v150_d50	4	3.15	3.2	1.53	**1.17**	3.16	3.19	724.92	477.10	2756.36
v150_d70	3	1.58	1.5	**0.72**	0.73	1.61	1.48	62.56	49.69	1828.86
v200_d5	27	715.33	843.77	-	-	596.16	**489.47**	-	-	-
v200_d10	16	-	-	-	-	-	–	-	-	-
v200_d20	9	712.43	1028.33	2738.33	1120.78	**709.24**	1060.56	-	-	-
v200_d30	7	1414.36	1056.05	669.48	**527.12**	1393.39	1034.08	-	-	-
v200_d50	4	18.29	12.27	4.38	**3.24**	18.78	12.03	3363.33	1887.43	2015.50
v200_d70	3	3.5	3.2	1.68	**1.67**	3.44	2.94	340.20	275.84	815.41

Tableau 4.II – Temps d'exécution en seconde des méthodes exactes

Instances	Nos méthodes						Littérature			
	BEP		BEI		PPCB		SABE		IPBE	
	Iter	Coupes	Iter	Coupes	Iter	Coupes	Iter	Coupes	Iter	Coupes
v30_d10	15	44	14	36	18	60	1124	1123	1237	1236
v30_d20	2	3	2	3	2	3	2	1	2	1
v30_d30	1	0	1	0	1	0	1	0	1	0
v30_d50	1	0	1	0	1	0	1	0	1	0
v30_d70	1	0	1	0	1	0	1	0	1	0
v50_d5	41	183	41	188	50	223	1123	1122	7368	7367
v50_d10	3	7	3	7	3	7	4	3	4	3
v50_d20	1	0	1	0	1	0	1	0	1	0
v50_d30	1	0	1	0	1	0	1	0	1	0
v50_d50	1	0	1	0	1	0	1	0	1	0
v50_d70	1	0	1	0	1	0	1	0	1	0
v70_d5	30	149	45	214	43	199	1453	1452	3610	3609
v70_d10	1	0	1	0	1	0	1	0	1	0
v70_d20	1	0	1	0	1	0	1	0	1	0
v70_d30	1	0	1	0	1	0	1	0	1	0
v70_d50	1	0	1	0	1	0	1	0	1	0
v70_d70	1	0	1	0	1	0	1	0	1	0
v100_d5	9	49	8	42	8	48	263	262	262	261
v100_d10	1	0	1	0	1	0	1	0	1	0
v100_d20	1	0	1	0	1	0	1	0	1	0
v100_d30	1	0	1	0	1	0	1	0	1	0
v100_d50	1	0	1	0	1	0	1	0	1	0
v100_d70	1	0	1	0	1	0	1	0	1	0
v120_d5	6	12	15	55	3	16	32	30	43	41
v120_d10	2	0	4	4	2	0	2	0	4	2
v120_d20	1	0	1	0	1	0	1	0	1	0
v120_d30	1	0	1	0	1	0	1	0	1	0
v120_d50	1	0	1	0	1	0	1	0	1	0
v120_d70	1	0	1	0	1	0	1	0	1	0
v150_d5	14	45	20	74	1	4	212	211	246	244
v150_d10	4	4	4	4	1	0	4	2	4	2
v150_d20	3	4	3	4	3	0	3	2	3	2
v150_d30	1	0	1	0	1	0	1	0	1	0
v150_d50	1	0	1	0	1	0	1	0	1	0
v150_d70	1	0	1	0	1	0	1	0	1	0
v200_d5	4	4	6	17	1	2	6	4	10	8
v200_d10	1	0	1	0	1	0	1	0	1	0
v200_d20	1	0	1	0	1	0	1	0	1	0
v200_d30	2	2	2	2	2	0	2	1	2	1
v200_d50	1	0	1	0	1	0	1	0	1	0
v200_d70	1	0	1	0	1	0	1	0	1	0

Tableau 4.III – Nombre d'itérations et de coupes des méthodes de Benders

Instances	BEI					
	Avec Borne Sup			Sans Borne Sup		
	Temps	Iter	Coupes	Temps	Iter	Coupes
v30_d10	0.04	14	36	0.07	32	44
v30_d20	0.01	2	3	0.04	29	13
v30_d30	0	1	0	0.04	28	2
v30_d50	0	1	0	0.03	28	0
v30_d70	0	1	0	0.03	29	0
v50_d5	0.58	41	188	0.77	94	243
v50_d10	0.05	3	7	0.21	53	44
v50_d20	0.03	1	0	0.14	49	11
v50_d30	0.04	1	0	0.13	47	3
v50_d50	0.02	1	0	0.11	48	0
v50_d70	0.02	1	0	0.1	49	0
v70_d5	1.38	45	214	1.19	105	205
v70_d10	0.07	1	0	0.4	73	36
v70_d20	0.09	1	0	0.3	65	2
v70_d30	0.09	1	0	0.31	66	0
v70_d50	0.05	1	0	0.26	68	0
v70_d70	0.06	1	0	0.26	69	0
v100_d5	1.24	8	42	3.43	139	209
v100_d10	0.21	1	0	0.82	94	13
v100_d20	0.75	1	0	1.33	94	2
v100_d30	1.29	1	0	1.92	95	0
v100_d50	0.58	1	0	1.94	97	0
v100_d70	0.49	1	0	1.91	98	0
v120_d5	1.62	15	55	3.09	132	136
v120_d10	2.68	4	4	5.35	116	20
v120_d20	2.21	1	0	2.86	113	0
v120_d30	2.14	1	0	3.89	119	8
v120_d50	1.42	1	0	1.99	117	0
v120_d70	0.81	1	0	1.51	118	0
v150_d5	79.8	20	74	139.12	222	368
v150_d10	3.08	4	4	36.95	145	18
v150_d20	188.81	3	4	124.41	146	8
v150_d30	6.59	1	0	11.08	145	0
v150_d50	3.26	1	0	5.27	147	0
v150_d70	1.5	1	0	2.8	148	0
v200_d5	843.77	6	17	648.17	216	153
v200_d10	-	1	0	-	188	6
v200_d20	1028.33	1	0	924.32	194	4
v200_d30	1056.05	2	2	1760.99	195	2
v200_d50	12.27	1	0	18.56	197	0
v200_d70	3.22	1	0	6.31	198	0

Tableau 4.IV – Exécution de la méthode BEI sans heuristique d'initialisation

CONCLUSION

Dans ce mémoire, nous avons abordé le problème de l'ensemble dominant connexe de cardinalité minimale. Nous nous sommes penchés, en particulier, sur le développement de méthodes pour sa résolution faisant appel à la programmation par contraintes et la programmation en nombres entiers. Nous y avons présenté, en l'occurrence, une heuristique et quelques méthodes exactes pouvant être utilisées comme heuristiques si on limite leur temps d'exécution.

Après avoir exposé quelques notions théoriques sur les domaines de la programmation par contraintes et en nombres entiers, nous avons présenté le problème et ses principales caractéristiques et champs d'application, ainsi qu'une brève revue de la littérature traitant ce sujet. Nous nous sommes attardés, en particulier, sur le travail de Gendron et al. [20] qui constitue la principale source d'inspiration et l'article de base de ce mémoire. Nous sommes ensuite passés à la présentation de l'essentiel de nos contributions, notamment certains résultats théoriques que nous avons établis et que nous avons exploités lors du développement de nos méthodes. Ces méthodes comprennent une heuristique qui améliore celles déjà existantes et des méthodes exactes basées sur la méthode de décomposition de Benders et la stratégie d'investigation itérative, la résolution des problèmes associés se faisant soit par la programmation linéaire en nombres entiers soit par la programmation par contraintes.

Nous avons finalement implémenté chacune des méthodes que nous avons développées en langage C++, et nous avons réalisé, pour chacune, des exécutions sur une série de 41 instances du problème impliquant des graphes de tailles et densités variées. L'analyse comparative des temps d'exécution, nombre d'itérations et de coupes additionnelles ont révélé que nos méthodes sont efficaces, puisqu'elles améliorent les résultats obtenus par les méthodes connues dans la littérature. Nous constatons, entre autre, que les méthodes basées sur la programmation linéaire en nombres entiers, dont fait partie celle de décomposition de Benders avec stratégie d'investigation itérative, qui peut être considérée comme la plus performante, four-

nissent les résultats les plus satisfaisants. Celles basées sur la programmation par contraintes sont plus efficaces pour la résolution des instances de très grandes taille et densité.

Finalement, la comparaison de l'une de nos méthodes avec et sans heuristique d'initialisation nous a permis de constater que l'utilisation de cette dernière peut s'avérer, dans certains cas, moins avantageuse. Ceci nous a permis d'ouvrir quelques perspectives de recherche.

Perspectives

Pour améliorer le présent travail, nous proposons quelques perspectives et pistes de recherches qui nous semblent prometteuses.

Dans un premier temps, nous proposons d'exploiter les conclusions que nous avons tirées sur l'impact de l'heuristique d'initialisation sur les performances des méthodes de résolution, en développant des méthodes permettant l'identification des coupes les plus fortes et efficaces pour les ajouter au problème avant de faire appel au solveur, afin d'accélérer le processus de résolution.

Dans un second temps, nous proposons d'intégrer les coupes 3.2 dans la méthode de branch-and-cut en utilisant une formulation de la connexité qui utilise des variables associées aux sommets. En effet, la méthode de branch-and-cut est efficace, mais rencontre des difficultés lorsqu'elle est utilisée sur les instances de grande taille à cause du grand nombre de contraintes et de variables que l'ancienne formulation implique. Puisque le problème que nous traitons est défini sur les sommets, nous pensons qu'une formulation de la connexité avec des variables associées, elles aussi, aux sommets, serait mieux adaptée à notre cas. Ceci évitera, en effet, d'encombrer et de compliquer le modèle en ayant recours à deux types de variables.

Finalement, dans la contrainte globale de connexité, chaque modification au niveau des variables entraîne le calcul de la borne inférieure de Steiner. Nous jugeons intéressant de faire en sorte que le calcul de cette borne se fasse de manière incrémentielle.

BIBLIOGRAPHIE

[1] K. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

[2] C. Asavathiratham, S. Roy, B. Lesieutre et G. Verghese. The influence model. *IEEE Control Systems Magazine*, 21:52–64, 2001.

[3] J.C. Beck, K.N. Brown, I. Miguel et P. Prosser. Enforcing connectivity in a fixed degree graph : A constraint programming case study. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.5663`.

[4] J.F. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[5] C. Berge. *Graphs and hypergraphs*. Elsevier Science Ltd., 1973.

[6] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan et T. Walsh. Filtering algorithms for the NValue constraint. Dans *Constraints*, volume 11, pages 271–293, 2006.

[7] M.E. Bouzgarrou. *Parallélisation de la méthode du branch-and-cut pour résoudre le problème du voyageur de commerce*. Thèse de doctorat, Institut d'Informatique et de Mathématiques Appliquées de Grenoble, 1998.

[8] K. Brown, P. Prosser, J.C. Beck et C. Wu. Exploring the use of constraint programming for enforcing connectivity during graph generation. Dans *Proceedings of The 5th workshop on modelling and solving problems with constraints (held at IJCAI05)*, pages 26–31, 2005.

[9] R. Carvajal, M. Constantino, M. Goycoolea, J.P Vielma et A. Weintraub. Imposing connectivity constraints in forest planning models. *Operations Research*, 51:824–836, 2013.

[10] J.O. Cerdeira, K.J. Gaston et L.S. Pinto. Connectivity in priority area selection for conservation. *Environmental Modeling and Assessment*, 10:183–192, 2005.

[11] J.O Cerdeira et L.S. Pinto. Requiring connectivity in the set covering problem. *Journal of Combinatorial Optimization*, 9:35–47, 2005.

[12] D.S. Chen, R.G. Batson et Y. Dang. *Applied integer orogramming : Modeling and solution*. Wiley, 2011.

[13] F. Dai et J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 15:908–920, 2004.

[14] P. Domingos et M. Richardson. Mining the network value of customers. Dans *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66, 2001.

[15] D.Z. Du et P.J. Wan. *Connected dominating set : Theory and applications*, volume 77. Springer Publishing Company, Incorporated, 2012.

[16] S. Eubank, H. Guclu, VS. Kumar, MV. Marathe, A. Srinivasan, Z. Toroczkai et N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429:180–184, 2004.

[17] N. Fan et J.P. Watson. Solving the connected dominating set problem and power dominating set problem by integer programming. Dans G. Lin, éditeur, *Combinatorial Optimization and Applications*, volume 7402 de *Lecture Notes in Computer Science*, pages 371–383. Springer Berlin Heidelberg, 2012.

[18] F.V. Fomin, F. Grandoni et D. Kratsch. Solving connected dominating set faster than 2^n. *Algorithmica*, 52:153–166, 2008.

[19] A. Fügenschuh et M. Fügenschuh. Integer linear programming models for topology optimization in sheet metal design. *Mathematical Methods of Operations Research*, 68:313–331, 2008.

[20] B. Gendron, A. Lucena, A. Cunha et L. Simonetti. Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating

set problem. *INFORMS Journal on Computing*, 2013. URL `http://dx.doi.org/10.1287/ijoc.2013.0589`.

[21] J. Goldenberg, B. Libai et E. Muller. Using complex systems analysis to advance marketing theory development. *Academy of Marketing Science Review*, 2001:1–18, 2001.

[22] M. Grötschel, L. Lovász et A. Schrijver. *Geometric algorithms and combinatorial optimization*. Algorithms and combinatorics. Springer-Verlag, 1993.

[23] S. Guha et S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20:374–387, 1998.

[24] T.W. Haynes, S. Hedetniemi et P. Slater. *Fundamentals of domination in graphs*. Chapman & Hall/CRC Pure and Applied Mathematics. Taylor & Francis, 1998.

[25] S. Hong et L. Weifa. Efficient enumeration of all minimal separators in a graph. *Theoretical Computer Science*, 180:169–180, 1997.

[26] J.N. Hooker et G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming, Series B*, 96:33–60, 2003.

[27] R.M. Karp. Reducibility among combinatorial problems. Dans R.E. Miller et J.W. Thatcher, éditeurs, *Complexity of computer computations*, pages 85–103. Plenum Press, 1972.

[28] C.L. Liu. *Introduction to combinatorial mathematics*. McGraw-Hill, 1968.

[29] A. Lucena, N. Maculan et L. Simonetti. Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Computational Management Science*, 7:289–311, 2010.

[30] K. Marriott et P.J. Stuckey. *Programming with constraints : An introduction*. Adaptive Computation and Machine. MIT Press, 1998.

[31] R.K. Martin. Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters*, 10:119–128, 1991.

[32] C.E. Miller, A.W. Tucker et R.A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, 1960.

[33] G.L. Nemhauser et L.A Wolsey. *Integer and combinatorial optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1999.

[34] P. Refalo. Impact-based search strategies for constraint programming. Dans Mark Wallace, éditeur, *Principles and practice of constraint programming – CP 2004*, volume 3258 de *Lecture Notes in Computer Science*, pages 557–571. Springer Berlin Heidelberg, 2004.

[35] W. Ren et Q. Zhao. A note on ′algorithms for connected set cover problem and fault-tolerant connected set cover problem′. *Theoretical Computer Science*, 412:6451–6454, 2011.

[36] E.M. Rogers. *Diffusion of innovations, 5th Edition*. Free Press, 2003.

[37] F. Rossi, P. van Beek et T. Walsh. *Handbook of constraint programming*. Foundations of Artificial Intelligence. Elsevier Science, 2006.

[38] L.M. Rousseau et G. Pesant. Programmation par contraintes. Dans P. Baptiste, éditeur, *Gestion de production et ressources humaines : méthodes de planification dans les systèmes productifs*. Presses Internationales Polytechnique, 2005.

[39] L. Simonetti, A. Cunha et A. Lucena. The minimum connected dominating set problem : Formulation, valid inequalities and a branch-and-cut algorithm. Dans J. Pahl, T. Reiners et S. Vob, éditeurs, *Network Optimization*, volume 6701 de *Lecture Notes in Computer Science*, pages 162–169. Springer Berlin Heidelberg, 2011.

[40] Thomas W. Valente. *Network models of the diffusion of innovations*. Quantitative methods in communication. Hampton Press, 1995.

[41] P.J. Wan, K.M. Alzoubi et O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9:141–149, 2004.

[42] F. Wang, H. Du, E. Camacho, K. Xu, W. Lee, Y. Shi et S. Shan. On positive influence dominating sets in social networks. *Theoretical Computer Science*, 412:265–269, 2011.

[43] L.A. Wolsey. *Integer programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.

[44] J. Wu et H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. Dans *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, DIALM '99, pages 7–14. ACM, 1999.

[45] A. Zanarini et G. Pesant. Solution counting algorithms for constraint-centered search heuristics. *Constraints*, 14:392–413, 2009.