Faculté
des Sciences
Département d'Informatique

UMONS
Université de Mons

Comparaison d'algorithmes pour le problème d'ensemble dominant minimum dans un graphe

Mémoire réalisé par Zakariae BOUALI
pour l'obtention du diplôme de Master en sciences informatiques

Année académique 2018–2019

Directeur: Hadrien Mélot
Service: Service d'Algorithmique

Faculté des Sciences • Université de Mons • Place du Parc 20 • B-7000 Mons

Remerciements

J'ai envie d'adresser mes sincères remerciements à ceux qui ont contribué à la réalisation de ce mémoire.

Je tiens tout particulièrement à remercier mon directeur de mémoire, M. Hadrien Mélot, professeur à l'université de Mons. pour sa disponibilité, son aide et surtout ces judicieux conseils tout au long de ce travail.

Je remercie également toute l'équipe pédagogique de l'université de Mons.

Je n'oublie pas de remercier ma chère mère qui m'a soutenu durant toute la période de mes études.

Table des matières

Table des figures

Introduction

Ce mémoire concerne l'étude d'un problème d'optimisation intitulé « l'ensemble dominant minimum dans un graphe » (*Minimum Dominating Set*) [1].

Ce problème consiste à trouver un ensemble minimum de sommets dans un graphe, tel que chaque sommet du graphe est soit dans cet ensemble, soit voisin d'un sommet de cet ensemble.

L'étude de ce problème et, plus généralement, les problèmes de domination remonte aux années 70 [2]. Il trouve son application dans plusieurs domaines. Dans les réseaux sans fil par exemple, l'ensemble dominant est utilisé pour trouver des itinéraires efficaces au sein des réseaux ad hoc. Il est également utilisé dans la synthèse de documents et aussi dans l'étude des problèmes liés aux réseaux sociaux. Ces applications seront détaillées dans le premier chapitre.

Mon objectif principal est de faire une revue de la littérature de quelques méthodes de résolution, précisément les méthodes exactes et les approches heuristiques, ainsi que de faire une comparaison de performances entre les différentes méthodes.

Le premier chapitre est dédié à la présentation des notions essentielles à la bonne compréhension du sujet. Je rappellerai certaines notions de la théorie des graphes, le problème de l'ensemble dominant et ses applications associées y seront également introduits. Je présenterai ensuite certaines notions concernant la théorie de complexité.

Dans le deuxième et le troisième chapitre, je présenterai les algorithmes exacts dédiés à résoudre ce problème. Le quatrième chapitre concerne les solutions heuristiques. Le dernier et cinquième chapitre, quant à lui, sera consacré à la présentation et la comparaison des résultats obtenus par les différents algorithmes.

Quatre articles scientifiques ont été la source principale de ma recherche tout au long du développement des algorithmes.

Dans le premier article [3], les auteurs détaillent des solutions exactes au problème de l'ensemble dominant. Je me suis focalisé sur l'implémentation de deux solutions ; la première solution consiste à la recherche de l'ensemble dominant minimum dans tous types de graphes, alors que la deuxième concerne les graphes de degré maximum $\leqslant 3$.

Les auteurs du deuxième article [4], proposent une solution au problème de domination en réduisant celui-ci à un problème de couverture par ensembles [5].

Dans le troisième article [6], l'auteur propose des approches heuristiques pour résoudre le problème en un temps polynomial.

Dans le quatrième article, les auteurs proposent deux méta-heuristiques de type population, l'approche utilisant un algorithme génétique a été implémentée afin de résoudre le problème de domination.

Chapitre I

Notions de base

Dans ce chapitre, nous présentons les notions de bases que l'on va retrouver tout au long de ce mémoire. Ces notions sont les bases de la théorie des graphes, la définition du problème de l'ensemble dominant et les applications associées. Nous terminerons par un rappel sur la théorie de la complexité.

La première section de ce chapitre présente les bases de la théorie des graphes. Pour en savoir plus, nous conseillons aux lecteurs l'ouvrage *Graph Theory* [7] de *Reinhard Diestel*.

I.1 Théorie des graphes

Graphe
Un graphe non orienté $G = (V, E)$ est une paire d'ensembles tel que E contient des paires non ordonnées de V.
Les éléments de V sont appelés sommets (nœuds ou points) du graphe G.
Les éléments de E sont les arêtes du graphe G.
Pour une arête $e \in E$, on note $e = \{x, y\}$ lorsque cette arête relie deux sommets $x, y \in V$.
L'ensemble des sommets de G est noté $V(G)$, alors que l'ensemble des arêtes est noté $E(G)$.

Exemple
Dans la figure I.1, on a :

— $V = \{a, b, c, d, e, f, g, h\}$

— $E = \{\{a, d\}, \{b, c\}, \{b, d\}, \{d, e\}, \{e, c\}, \{e, h\}, \{h, d\}, \{f, g\}, \{d, g\}, \{g, h\}\}$

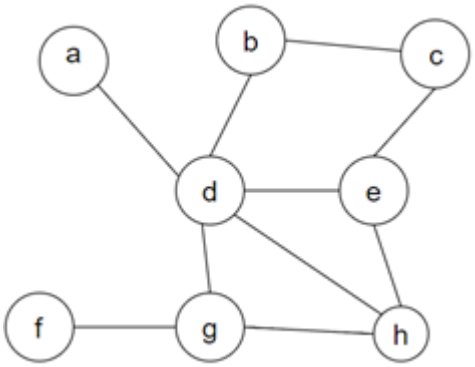

FIGURE I.1 – Graphe $G = (V, E)$.

Voisinage

Étant donné deux sommets u et v appartenant à l'ensemble V, u est adjacent à v si l'arête $\{u, v\} \in E$.
On dit que u et v sont adjacents, ou encore que u et v sont voisins. On dit aussi que le sommet v est incident à l'arête e.

— On définit le voisinage ouvert d'un sommet $v \in V$ par :
$N(v) = \{u \in V : \{u, v\} \in E\}$.

— On définit le voisinage fermé de v par : $N[v] = N(v) \cup \{v\}$.

Dans le graphe de la figure I.1, on a : $N(g) = \{f, d, h\}$, alors que $N[g] = \{g, f, d, h\}$.

Degré d'un graphe

Le degré d'un sommet v est le nombre d'arêtes incidentes à v. Il est noté $d(v)$.
Dans le graphe de la figure I.1 on a $d(e) = 3$, les arêtes incidentes à e sont : $\{c, e\}, \{d, e\}, \{h, e\}$.
On note respectivement $\delta(G)$ et $\Delta(G)$, les degrés minimum et maximum du graphe G, qui correspondent au plus petit et plus grand degré des sommets :

— $\delta(G) = min\{d(v)|v \in V(G)\}$
— $\Delta(G) = max\{d(v)|v \in V(G)\}$

Dans le graphe de la figure I.1 on a $\delta(G) = 1$, et $\Delta(G) = 5$.

Sous graphe

Soit $G = (V, E)$ un graphe. Le graphe $G' = (V', E')$ est un sous-graphe de G, si $E' \subseteq E$ et $V' \subseteq V$.
Pour un sous-ensemble de sommets $A \subseteq V$, le sous-graphe de G induit par A, est le graphe $G' = (A, E(A))$ noté $G[A]$, dont l'ensemble des sommets est A et l'ensemble des arêtes $E(A)$ est formé de toutes les arêtes de G ayant leurs deux extrémités dans A.
$G[A]$ est défini formellement par : $G[A] = (A, \{\{x, y\} \in E : x, y \in A\})$.

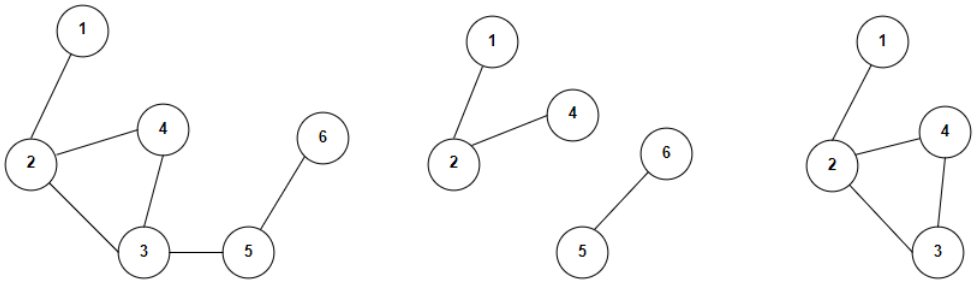

FIGURE I.2 – Représentation d'un graphe G, d'un sous-graphe G' et du sous-graphe induit par $A = \{1, 2, 3, 4\}$.

Chemin

Un chemin dans un graphe $G = (V, E)$ est une séquence de sommets $< v_1, v_2, ..., v_k >$ telle que $\{v_i, v_{i+1}\} \in E$ pour tout $i \in [1, k-1]$. La longueur du chemin est égale à $k-1$ et définie par le nombre d'arêtes utilisées.
Dans le graphe de la figure I.1, l'ensemble des sommets $\{a, d, h, g, f\}$ forme un chemin de longueur 4.

Cycle

Un cycle est un chemin dont les sommets de départ et de fin sont les mêmes. Un chemin $< v_1, v_2, ..., v_k >$ forme un cycle si $v_1 = x_k$. Un cycle de longueur k est noté C^k.
Dans le graphe de la figure I.1, l'ensemble des sommets $\{b, c, e, d\}$ forme un cycle de longueur 4.

Connexité

Un graphe non orienté $G = (V, E)$ est dit connexe ; si chaque sommet est accessible à partir de n'importe quel sommet. Autrement dit, si pour toute paire de sommets, il existe un chemin dans G reliant ces deux sommets. Une composante connexe de G est un sous-graphe induit $G' = (V', E')$ de G qui

est connexe.
Par exemple, le graphe de la figure I.3 n'est pas connexe, car il n'existe pas de chemin entre a et d. Par contre, il contient deux composantes connexes, le sous-graphe induit par les sommets $\{d, e, f\}$ et le sous-graphe induit par les sommets $\{a, b, c\}$.

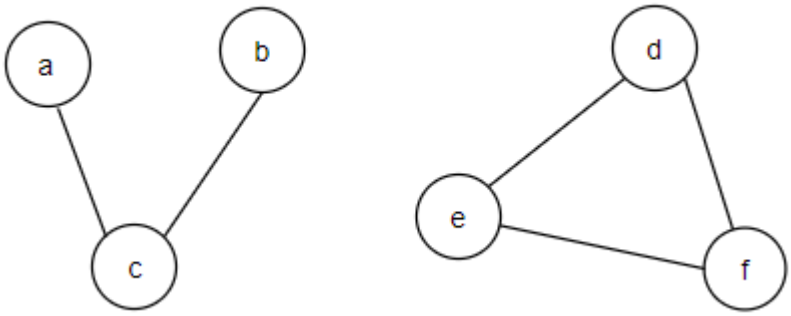

FIGURE I.3 – Un graphe contenant deux composantes connexes.

Liste d'adjacence

On peut représenter un graphe simple, en donnant pour chacun de ses sommets la liste des sommets auxquels il est adjacent.

Exemple

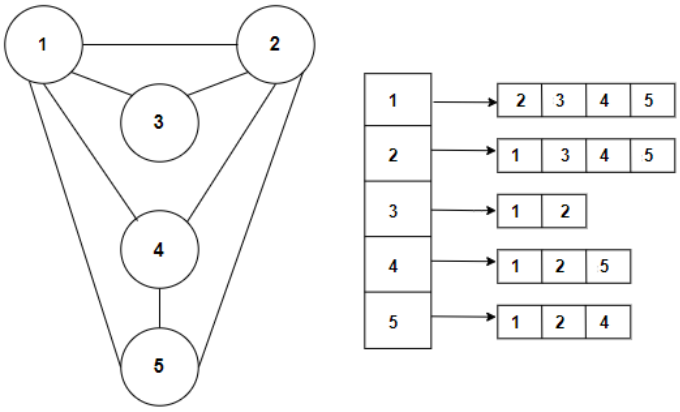

FIGURE I.4 – Liste d'adjacence.

I.2 Ensemble dominant

Ayant défini la notion de graphe et quelques termes fondamentaux associés, nous sommes maintenant prêts à discuter le concept d'un ensemble dominant dans un graphe.

Définition
Dans la théorie des graphes, **un ensemble dominant** D dans un graphe G, est un sous-ensemble de sommets de G, tel que chaque sommet du graphe est soit dans D soit fait partie des voisins d'un sommet de D.

Problème : ensemble dominant (*Dominating Set, DS*)
Entrée : un graphe $G = (V, E)$; un entier k.
Question : existe-t-il un ensemble dominant de taille au plus égal à k pour G ?

Un **ensemble dominant minimum** (*Minimum Dominating Set, MDS*), est un ensemble dominant, contenant le plus petit nombre possible de sommets. Un graphe peut posséder plusieurs ensembles dominants minimums.

Le nombre de sommet dans un ensemble dominant minimum, noté $\gamma(G)$, est appelé **nombre de domination**.

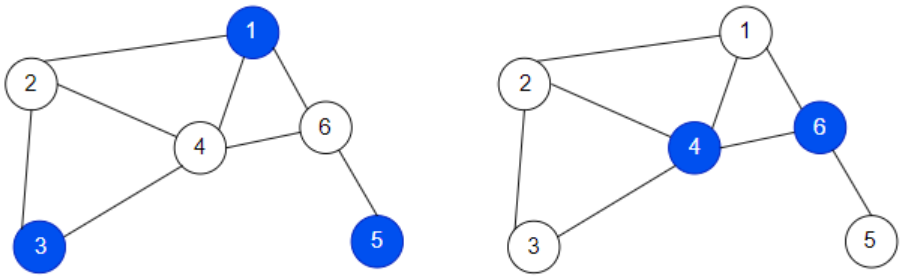

FIGURE I.5 – Ensemble dominant, ensemble dominant minimum.

L'ensemble $D_1 = \{1, 3, 5\}$ du graphe représenté dans la figure I.5 est un ensemble dominant, car tous les sommets du graphe soit appartiennent au voisinage des sommets $\{1, 3, 5\}$ soit, appartiennent à D_1. On dit que l'ensemble $\{1, 2, 3, 4, 5, 6\}$ est dominé par l'ensemble D_1. Par ailleurs, l'ensemble $D_2 = \{4, 6\}$ du graphe est un ensemble dominant minimum, puisque c'est un ensemble dominant et il n'existe pas un autre ensemble dominant qui soit de taille inférieure à 2. En effet, aucun sommet n'est adjacent à tous les autres sommets du graphe. Pour le graphe de la figure I.5 on a $\gamma(G) = 2$.

I.3 Champs d'application

Le concept de l'ensemble dominant a existé pendant longtemps. Plus de détails se trouvent dans l'ouvrage *Fundamentals of domination in graphs* [2]. Ce concept, trouve son application dans de nombreux domaines, dont voici quelques exemples :

I.3.1 Les reines dominantes

Le problème de domination des reines (*Queens dominating*) peut être considéré comme l'origine de l'étude des ensembles dominants dans les graphes [2].

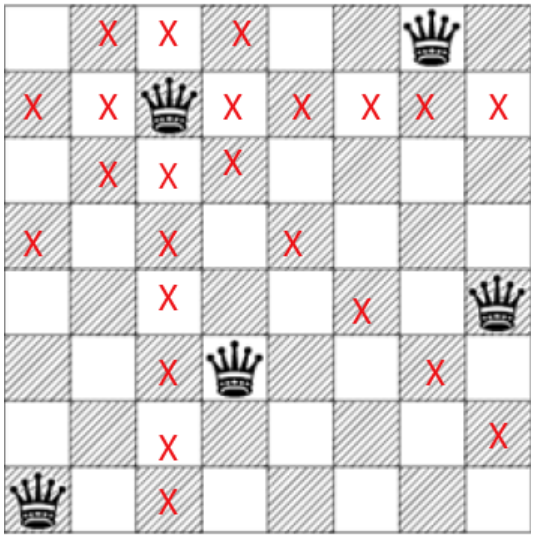

FIGURE I.6 – Échiquier.

La figure I.6 illustre un échiquier 8x8 standard sur lequel sont placées des reines. Selon les règles des échecs, une reine peut, en un mouvement, avancer de n'importe quel nombre de carrés horizontalement, verticalement ou en diagonale.
Par exemple, la reine placée à la ligne 2 et la colonne 3, peut se déplacer dans toutes les cases marquées par X. On dit que, cette reine domine toutes les cases marquées par X. Le problème de domination des reines est de déterminer le nombre minimum de reines pouvant dominer toutes les cases de l'échiquier.

On peut formuler ce problème de la façon suivante :
Problème : ensemble dominant des reines.
Entrée : un échiquier NxN, un nombre positif k.
Question : est-il possible de placer k reines sur l'échiquier tel que toutes les cases sont dominées ?

La figure I.6 montre cinq reines pouvant dominer toutes les cases de l'échiquier. Il était connu en 1850, que cinq est le nombre minimum de reines pouvant dominer tous les carrés d'un échiquier de 8x8.

Voici d'autres solutions pour la même instance :

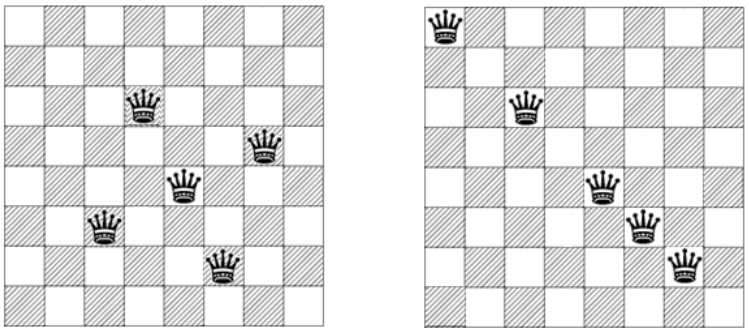

FIGURE I.7 – Solutions pour un échiquier 8x8 avec 5 reines.

Le tableau suivant, présente les résultats connus pour les instances du problème jusqu'à une taille 17x17. La première ligne du tableau représente la taille de l'échiquier (N représente le nombre de lignes et de colonnes de l'échiquier), alors que la deuxième ligne représente le nombre minimum de reines pouvant dominer un échiquier NxN.

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Solution	1	1	1	2	3	3	4	5	5	5	5	6	7	8	9	9	9

Le problème de domination des reines, peut être formulé sous une forme plus générale qui est le problème de l'ensemble dominant minimum dans un graphe.

Un graphe G peut représenter l'échiquier NxN tel que :
— chaque carré de l'échiquier est représenté par un sommet ;
— deux sommets u, v du graphe G sont adjacents, si leurs carrés correspondants sont sur la même ligne, la même colonne ou la même diagonale.

On peut observer que chercher le nombre minimum de reine dominante dans un échiquier, est équivalent à chercher l'ensemble dominant minimum dans le graphe G construit.

I.3.2 La synthèse de documents

Ce problème est étudié dans le domaine de la recherche d'informations. Un algorithme résolvant ce problème vise à extraire les informations les plus importantes depuis un ensemble de documents afin de générer un résumé compressé.
Ce problème peut être formulé sous forme d'un problème de domination dans un graphe de la façon suivante :
Les phrases sont représentées par des sommets, et une arête entre deux sommets indique la similarité entre deux phrases.

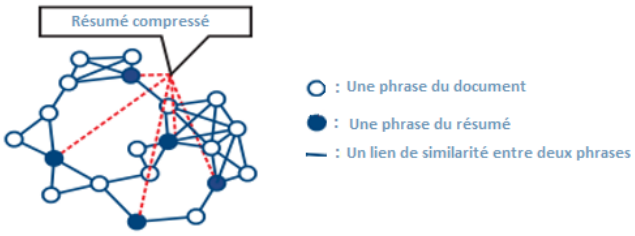

FIGURE I.8 – Illustration graphique de la synthèse de plusieurs documents via l'ensemble dominant minimum [8].

Un algorithme de synthèse de documents (*Multi-document summarization*) reçoit un ensemble de documents en entrée et produit comme résultat un résumé.
Dans leurs travaux [8], les auteurs représentent les phrases comme des vecteurs [9], puis ils obtiennent la similarité cosinus pour chaque paire de phrases. Si la similarité entre une paire de phrases s_i et s_j est au-dessus d'un seuil donné λ, il y a donc une arête entre s_i et s_j. Voici les étapes clés pour cet algorithme :

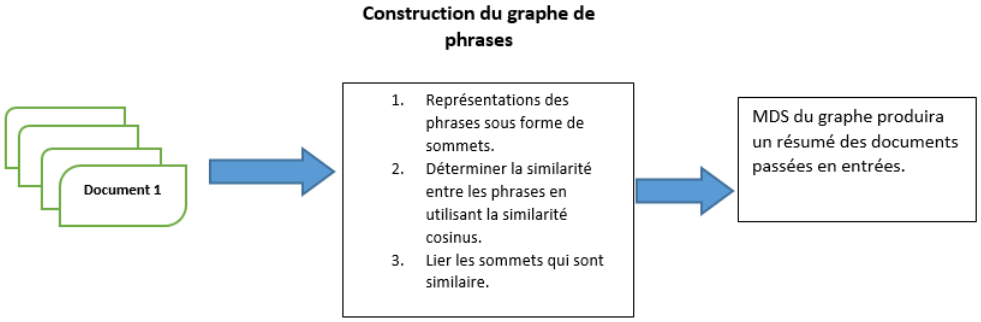

FIGURE I.9 – Étapes clés d'un algorithme de synthèse de documents.

I.3.3 Les réseaux des capteurs sans fil (*WSN*)

Les capteurs sans fil sont des petits périphériques à consommation faible, et sont fortement limités en capacité de calcul et de stockage. Leur principal but, reste de détecter des événements ou des changements dans leurs environnements, et d'envoyer ces informations à d'autres appareils électroniques, qui sont souvent des unités de calcul plus puissantes pour analyser ces données. Les réseaux de capteurs sans fil sont de plus en plus utilisés dans différents domaines : la santé, l'armée, l'agriculture, l'industrie, la sécurité [2, 10, 11]. La figure suivante illustre un système de surveillance, basé sur les réseaux de capteurs sans fil.

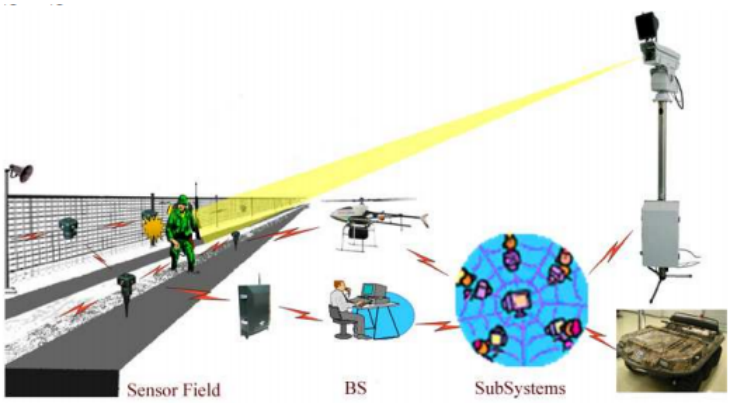

FIGURE I.10 – Wireless network [11].

Ce type de système est divisé en trois parties :
— une partie où sont mis les capteurs sans fil (*Sensor field*) ;
— une station de contrôle (*Base station*) ;
— des sous-systèmes : comprenant un drone, une caméra.

Un scénario qui peut se produire dans ce système de surveillance :

- un objet (humain, véhicule) entre dans l'intervalle de détection des capteurs sans fil. Les capteurs transmettent l'information à la station de contrôle à travers le réseau WSN ;
- la station de contrôle commande à distance la caméra afin de montrer l'emplacement où s'est produit l'événement ;
- la station de contrôle intervient si c'est nécessaire en commandant le drone à distance.

La transmission des informations à travers le réseau *WSN*, doit se faire de manière optimale. Dans ces réseaux, il n'existe pas d'infrastructure centrale commune pour l'organisation du réseau, la coordination doit se faire par les nœuds eux-mêmes. Afin de transmettre les informations collectées par les nœuds et dues à leur portée limitée, une épine dorsale (*Virtual backbone, VB*) est donc considérée comme très utile pour la retransmission des données.

La topologie du réseau *WSN* peut être représentée sous forme d'un graphe $G = (V, E)$ tel que l'ensemble de sommets V représente les nœuds du réseau et une arête $\{u, v\}$ appartenant à E représente un lien entre deux nœuds u, v adjacent pouvant communiquer dans leur intervalle de transmission.

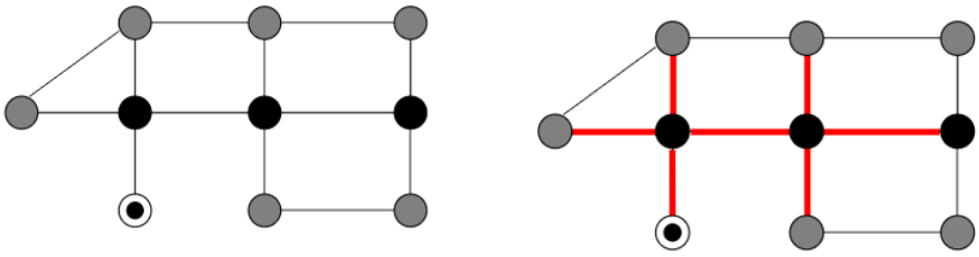

FIGURE I.11 – Ensemble dominant connecté minimum.

Un ensemble dominant minimum connecté (*Minimum Connected Dominating Set, MCDS*) [12] dans un graphe G est un MDS tel que toute paire de sommets de ce MDS est connectée. Cet ensemble joue un rôle très important dans la construction de l'épine dorsale du réseau. Le MCDS prendra en charge la retransmission des messages, puisque tout nœud, soit appartient au MCDS, soit est voisin d'un nœud du MCDS.
Tout nœud qui ne fait pas partie du MCDS pourra donc se mettre en sommeil lorsqu'il n'a pas de données à envoyer. De cette façon, les nœuds limités en ressources pourront économiser leur énergie, ce qui permettra aussi une réduction de la redondance du trafic et une augmentation de la bande pas-

sante.
Chercher l'ensemble de nœuds qui prendra en charge la liaison entre les différents nœuds du réseau revient à chercher un ensemble dominant minimum connecté dans le graphe G.

I.3.4 Les réseaux sociaux

Aujourd'hui, les réseaux sociaux jouent un rôle très important dans la diffusion d'informations et l'influence sur les individus. La taille et la popularité de ces réseaux ont largement grandit lors de la dernière décennie. Facebook, par exemple, a actuellement plus de 2 milliards d'utilisateurs actifs par mois [13]. De ce fait, beaucoup de recherches sont menées afin d'analyser différents problèmes liés aux réseaux sociaux.

L'objectif d'une campagne publicitaire, par exemple sur un réseau social, est d'influencer beaucoup de participants de ce réseau, la contrainte étant que seul un ensemble limité de personnes peut être contacté directement. Cependant, si tous les individus de ce réseau sont liés à au moins une personne de cet ensemble, la campagne de publicité peut alors atteindre tous les individus de ce réseau.
Ce problème peut être formulé en un problème de domination dans un graphe de la façon suivante :

- les membres du réseau social sont représentés par des nœuds dans un graphe G ;
- la connexion directe entre deux individus est représentée par un lien entre deux nœuds.

Chercher l'ensemble minimum de personnes qui sera accessible par tout individu du réseau, revient à chercher l'ensemble dominant minimum du graphe G.

I.3.5 Les stations radars

Ce problème a été discuté par *Berge* [2] et consiste en un certain nombre d'endroits stratégiques qui doivent être gardés sous surveillance.
Comme chaque station radio a un intervalle de diffusion limitée, un certain nombre de stations doivent être utilisées afin de couvrir toute la zone mise en surveillance. Comme l'implantation des stations radios est coûteuse, l'objectif est donc de localiser des radars pour la surveillance en un minimum d'endroits possibles.
Comment peut-on déterminer les emplacements dans lesquels on place les stations radar ? Le problème peut être formulé de la façon suivante :

On construit un graphe de telle sorte que nous représenterons les endroits qui doivent être surveillés par des sommets. Nous relions un sommet u à un sommet v, s'il est possible d'observer la station v depuis la station u. Chercher l'ensemble d'emplacements minimum dans lesquels placer les stations radars revient à trouver un ensemble dominant minimum dans ce graphe construit.

I.4 Algorithmique et complexité

I.4.1 Problème de décision

Dans la théorie de la calculabilité, un problème de décision est un problème dont la réponse est soit oui, soit non.

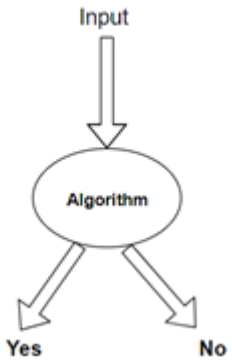

FIGURE I.12 – Problème de décision.

Le problème de domination dans un graphe est un problème de décision. On peut le formuler de telle sorte à demander si un graphe contient un sous-ensemble dominant inférieur ou égal à k.

P vs NP

La classe P comprend tous les problèmes de décision qui peuvent être reconnus en temps polynomial ; c'est-à-dire pour lesquels la décision d'une instance peut être atteinte en un temps polynomial. La classe NP est celle des problèmes pour lesquels on peut vérifier en temps polynomial si une solution proposée convient.

I.4.2 NP-complétude

Définition
On dit qu'un langage L_1 est réductible en temps polynomial à un langage L_2, noté $L_1 <_p L_2$, s'il existe une fonction calculable en temps polynomial $f : \{0,1\}^* \to \{0,1\}^*$ tel que pour tout $x \in \{0,1\}^*$, $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Définition
Un langage L est NP-complet si et seulement s'il appartient à NP et si tout langage de NP se réduit en temps polynomial à lui :
— $L \in NP$.
— Et $\forall L' \in NP, L' <_p L$

Définition
Soient L_1, L_2 deux langages récursifs. Si L_1 est NP-complet, si L_2 est dans NP et si $L_1 <_p L_2$ alors L_2 est aussi NP-complet.

Le problème de l'ensemble dominant est un problème NP-complet. Afin de vérifier ce résultat, on peut se référer à un autre problème, qui est lui-même NP-complet ; le problème de couverture par les sommets [14].

Problème : couverture par les sommets (*Vertex Cover, VC*)
Entrée : un graphe $G = (V, E)$; un entier k.
Question : existe-t-il un sous-ensemble de sommets V' tel que toute arête de E a au moins une extrémité dans V' et tel que $|V'| \leqslant k$?

Un sous-ensemble $X \subseteq V$ de sommets est une couverture par les sommets dans G, si toutes les arêtes de G sont incidentes à au moins un sommet de X. Autrement dit, X est une couverture par les sommets si et seulement si $\forall e \in E : e \cap X \neq \emptyset$.

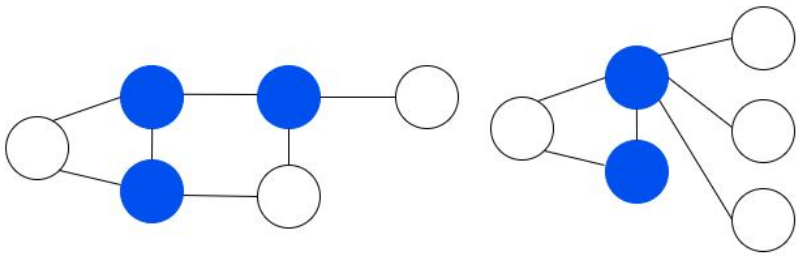

FIGURE I.13 – Exemples de couvertures par les sommets minimales.

Théorème : le problème VC est NP-complet [15].
Théorème : le problème DS est NP-complet.
Afin de prouver que le problème de l'ensemble dominant est un problème NP-complet, il est nécessaire de prouver qu'il appartient à la classe NP, et qu'il existe un autre problème NP-complet qui se réduit en temps polynomial à lui.

Le problème DS est dans NP
Étant donné une instance (G, k), on devine un sous-ensemble de sommets D de taille k. On vérifie que tous les sommets du graphe G soit appartiennent à D, soit ils sont voisins à un sommet de D. Comme cette vérification se fait en temps polynomial alors le problème $DS \in NP$.

Le problème DS est NP-complet [16]
Le problème de décision de l'ensemble dominant a été prouvé NP-complet par réduction avec le problème de couverture par sommets. Les deux problèmes sont similaires. La différence étant que le premier concerne des arcs alors que le second concerne les sommets.
Voici une preuve de ce résultat :

VC $\leqslant_p$ DS
On veut montrer qu'étant donné une instance (G, k) du problème de la couverture par sommets, on peut produire une instance (G', k') du problème de l'ensemble dominant en un temps polynomial.
On construit le nouveau graphe G' de la façon suivante :
Au départ G' contient tous les sommets et toutes les arêtes du graphe G. Pour chaque arête $\{u, v\} \in G$, ajoutons un sommet uv et les arêtes $\{u, uv\}$ et $\{v, uv\}$.

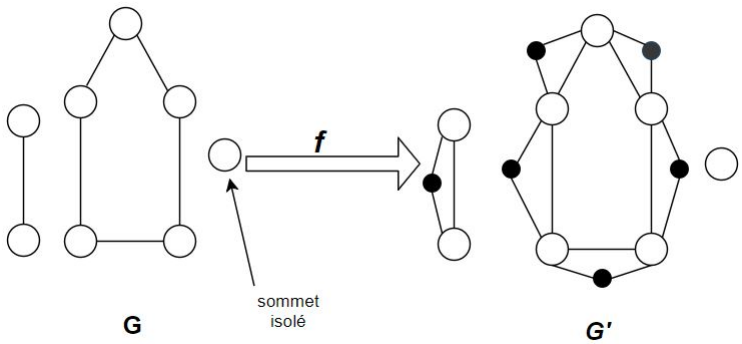

FIGURE I.14 – Graphe G' construit à partir de G.

Soit n_i le nombre de sommets isolés du graphe G et soit $k' = n_i + k$.

La figure I.14 illustre cette réduction, notons que celle-ci se fait en un temps polynomial.
Montrons maintenant qu'on a une couverture de sommets C de taille k si et seulement si G' a un ensemble dominant D de taille k'.

— **Si C est une couverture par sommets de G de taille k alors $C' = C \cup V_i$ est un ensemble dominant de G' de taille k' :**
Pour voir que C' est un ensemble dominant dans G', regardons d'abord que tous les sommets isolés de G' seront dominés.
Considérons un sommet $v \in G'$ parmi les sommets originaux de G, alors soit $v \in C$, soit v est adjacent à un sommet $u \in C$.
Considérons $uv \in G'$ un des sommets ajoutés à G pour obtenir G', alors ce sommet correspond à une arête $\{u, v\}$, ce qui implique qu'un des voisins de uv (u ou v) appartient à D. Donc le sommet uv de G' est dominé par le même sommet (u ou v) dans G'.

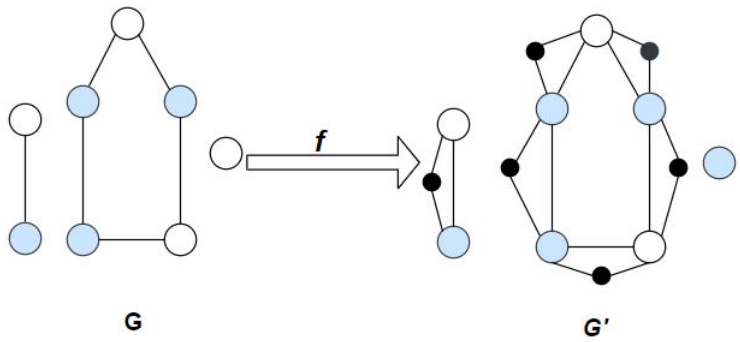

FIGURE I.15 – Graphe G' construit à partir de G.

— **Si C' est un ensemble dominant de G' de taille $k' = k+n_i$ alors G a une couverture par sommets de G de taille k :**
Soit $C'' = C' - V_i$, les sommets de l'ensemble dominant C' non isolés dans G', on peut se dire que les sommets de C'' forment une couverture par sommets du graphe G, mais ce n'est pas vrai vu qu'il pourrait y avoir des sommets de C'' qui ne sont pas dans G. Cependant, on peut prétendre qu'on n'a pas besoin des sommets de C'' pour dominer le graphe G. En particulier, pour chaque sommet uv de C' on peut le remplacer par son voisin u (ou v). Sachant que ce sommet uv domine u, v et uv dans G', si on n'utilise que u on domine toujours u, v et uv, si on remplace donc uv par u (ou v) on garde toujours la propriété de domination. Soit V' ce nouvel ensemble dominant. V' est une couverture par sommets de G car toute arête de G' touche un sommet de

V'. V' est donc une couverture par sommets de G de taille k.

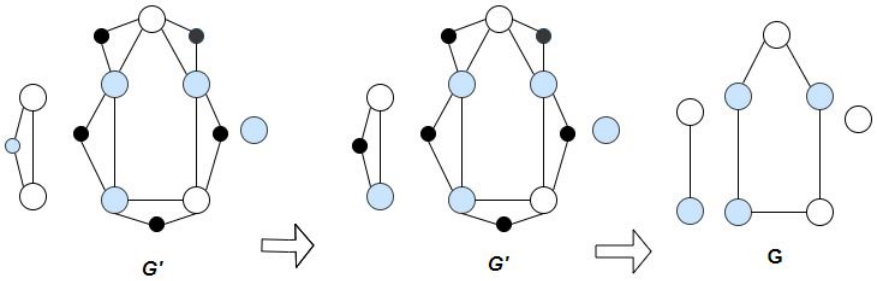

FIGURE I.16 – Graphe G construit à partir de G'.

Comme on sait que la couverture par sommets est un problème NP-complet, alors l'ensemble dominant est aussi un problème NP-complet.

I.4.3 Méthodes exactes Vs Méthodes approchées

La résolution des problèmes NP-complet peut se faire en énumérant et en vérifiant toutes les solutions possibles. Cependant, le temps de calcul nécessaire pour arriver à la solution optimale augmente rapidement quand la taille de l'instance augmente.

Les méthodes exactes garantissent une solution optimale au problème avec un temps qui peut être très grand. En revanche, dans certaines situations on peut chercher de bonnes solutions, mais sans garantie d'optimalité. Pour cela, on applique des algorithmes approchées. Ces derniers, sont des algorithmes qui permettent d'identifier une solution réalisable (pas forcement optimal) en un temps polynomial.
Le choix d'un algorithme approché est donc efficace pour calculer une solution approchée d'un problème.

Le choix entre ces deux types de méthodes pour résoudre un problème d'optimisation dépendra fortement de plusieurs critères :

- Contraintes de tailles : pour des instances petites en taille, les méthodes exactes donnent des solutions optimales en un temps raisonnable.
- Contraintes de temps : en combien de temps on espère trouver la solution ?

— Contrainte d'optimalité : arriver à trouver une solution optimale reste la meilleure des choses. Cependant peut-on accepter une solution supposée bonne pour notre problème ?

Dans ce mémoire, nous allons présenter deux types d'algorithmes approchés, les heuristiques simples et les les méta-heuristiques.

Les heuristiques :
Ce sont des algorithmes qui peuvent produire des solutions de bonne qualité sans garantie d'optimalité. Ces algorithmes sont spécifiques au problème à résoudre.

Les méta-heuristiques :
Ce sont des algorithmes génériques pouvant s'appliquer à tout problème d'optimisation, en spécialisant néanmoins certaines parties. Plusieurs méta-heuristiques s'inspirent de processus qu'on trouve dans la nature. Parmi les plus connues, on retrouve le recuit simulé, la recherche tabou, les algorithmes évolutionnaires.

Dans le quatrième chapitre de ce mémoire, nous allons présenter un algorithme génétique faisant partie de la famille des algorithmes évolutionnaires. Par la suite, nous allons comparer les qualités des résultats obtenus par les deux familles d'algorithmes approchés.

Dans le prochain chapitre, nous commencerons par découvrir certains algorithmes exacts connus pour trouver l'ensemble dominant minimum dans un graphe.

Chapitre II

Algorithmes exacts pour l'ensemble dominant minimum

II.1 Introduction

L'approche naïve pour chercher l'ensemble dominant minimum dans un graphe $G = (V, E)$, considère d'itérer sur l'ensemble des sous-ensembles de V et, de vérifier pour chaque sous-ensemble, si tout sommet du graphe est soit dans ce sous-ensemble soit adjacent à un sommet de ce sous-ensemble. Dans le pire des cas, cet algorithme nécessite d'énumérer et de vérifier tous les sous-ensembles de V. La cardinalité de cet ensemble vaut 2^n, cette opération implique donc de vérifier au plus 2^n ensemble, ce qui fait un temps exponentiel. La complexité de cet algorithme dans le pire des cas est en $O(2^n)$.

Jusqu'à maintenant, nous n'avons pas d'algorithmes polynomiaux pour résoudre les problèmes NP-difficile, ce qui est le cas pour notre problème de domination, cependant, pouvons-nous avoir une meilleure complexité que l'approche naïve ?

En première partie de ce chapitre, nous présentons l'approche conçue dans l'article [3] pour résoudre le problème en un temps exponentiel plus petit que l'approche naïve. En deuxième partie de ce chapitre, nous expliquons une autre approche pour les graphes de degré maximum plus petit ou égal à 3, les auteurs précisent que la première approche n'est pas toujours optimale pour ce type de graphe.

Dans leur article [3], les auteurs présentent des algorithmes exacts (exponentiels) pour résoudre le problème de l'ensemble dominant pour différents types de graphes. Le but des auteurs est de présenter des algorithmes qui ont un temps exponentiel rapide borné par des fonctions sous forme de (c^n) avec des constantes raisonnablement petites $c < 2$.

II.2 Algorithme général (pour tous types de graphe)

Introduction

Dans cette section, nous présentons le résultat principal du document [3] ; c'est à dire l'algorithme exact pour le MDS sur n'importe quel type de graphe, l'algorithme a une complexité au pire des cas en $O(1.93782^n)$, l'algorithme se base sur une proposition de *Reed* [17] qui permet de restreindre l'espace de recherche.

Proposition [17]

Pour chaque graphe G à n sommets et tel que $\delta(G) \geqslant 3$, G a un ensemble dominant de taille au plus égal à $3n/8$.

Idée générale

L'algorithme prend en entrée un graphe $G = (V, E)$ et un ensemble de sommets $X \subseteq V$, et retourne l'ensemble dominant minimum D. On dit que $D \subseteq V$ domine X si $X \in N[D]$.

L'idée principale de l'algorithme est de brancher en sous-problèmes et élaguer l'arbre de recherche en supprimant tous les sommets de degré un ou deux, L'objectif des branchements est d'arriver à des sous-graphes avec des sommets de degré égal à 0 ou de degré minimum égal à 3.

À chaque étape du branchement et selon les critères des sommets supprimés, on vérifie si on rajoute certains sommets à notre ensemble dominant D.

Le branchement se termine lorsqu'on atteint le cas de base ; c'est-à-dire un graphe dont tous les sommets sont de degré zéro ou au moins trois.

Supposons que G' est un tel graphe avec t sommets, la proposition confirme qu'il existe un ensemble de sommet dans G' avec aux plus $3t/8$ sommets qui domine G'.

Il suffit donc qu'on teste tous les sous-ensembles de taille jusqu'à $3t/8$ pour trouver l'ensemble dominant minimum D' de G'.

En calculant le nombre de combinaison possible $C^t_{3t/8}$ et en utilisant l'approximation de Stirling suivante pour les factorielles :

$$x! \approx x^x e^{-x}\sqrt{2}$$

On a :

$C^t_{3t/8} = \binom{t}{3t/8} = \frac{t!}{((3t/8)!(5t/8)!)} = O(8^t.3^{-3t/8}.5^{-5t/8}) = O(1.93782^t)$

où $8/(3^{3t/8}.5^{5t/8}) \approx 1.9378192$.

On observe que ce test pourra se faire en $O(1.93782^t)$. Il suffit donc de rajouter les sommets de degré 0 et les sommets de D' à D afin d'obtenir l'ensemble dominant minimum.
Dans la section suivante, nous allons expliquer les appels récursifs qui s'appliquent lorsqu'on n'est pas dans le cas de base décrit-ici, c'est à dire lorsqu'ils existent des sommets de degré 1 ou 2.

II.2.1 Règles de branchement

Discutons d'abord de la gestion des sous-cas.
Lorsque le graphe contient toujours un sommet de degré un ou deux, nous choisissons un tel sommet, disons v, et nous faisons un appel récursif en distinguant quatre cas selon deux critères :

— Le degré de v.
— $v \in X$ ou non.

L'algorithme utilise le voisinage fermé de v : $N[v] = N(v) \cup \{v\}$ où $N(v)$ est le voisinage ouvert de v.

Cas A : Le sommet v est de degré 1 et $v \in V - X$:

Dans ce cas, il n'est pas nécessaire de dominer le sommet v et il existe toujours un ensemble dominant minimum pour X qui ne contient pas v. Alors un ensemble dominant minimum pour $X - \{v\}$ dans $G - \{v\}$ est aussi un ensemble dominant minimum pour X dans G, et donc nous continuons d'appliquer l'algorithme sur $G - \{v\}$ et $X - \{v\}$.

Cas B : Le sommet v est de degré 1 et $v \in X$:

Soit w le voisin unique de v. Il existe toujours un ensemble dominant minimum X qui contient w, mais ne contient pas v. Si D' est un ensemble dominant minimum pour $X - N[w]$ dans $G - \{v, w\}$ alors $D' \cup \{w\}$ est un

ensemble dominant minimum pour X dans G, et donc nous continuons avec $G - \{v, w\}$ et $X - N[w]$.

La figure II.1 montre l'application de la règle de branchement B sur un graphe à 8 sommets.

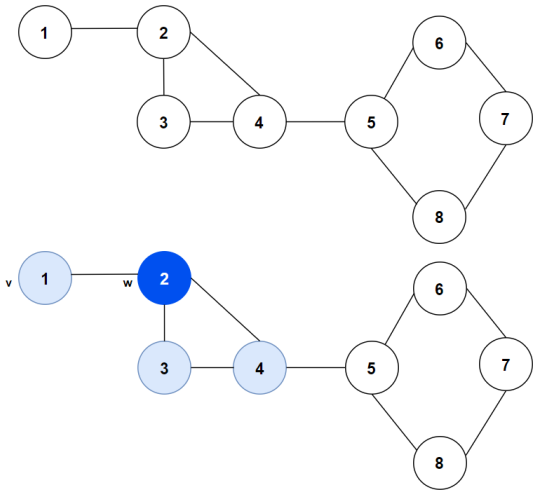

FIGURE II.1 – Illustration de la règle de branchement B.

Supposons que le sommet 1 est le sommet sélectionné par l'algorithme, l'objectif est de dominer l'ensemble $X = \{1, 2, 3, 4, 5, 6, 7, 8\}$ dans $G = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Comme le sommet $v \in X$, la règle B est appliquée, c'est à dire qu'on rajoute le sommet 2 à l'ensemble dominant D et on applique récursivement l'algorithme sur $G' = \{3, 4, 5, 6, 7, 8\}$ et $X' = \{5, 6, 7, 8\}$.

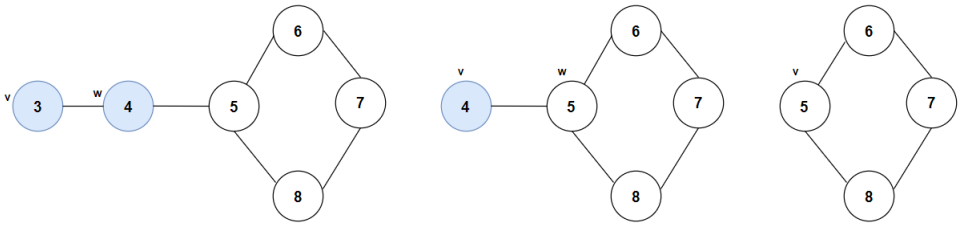

FIGURE II.2 – Illustration de la règle de branchement A.

Dans l'étape suivante, supposons que le sommet 3 est choisi par l'algorithme, on a $X = \{5, 6, 7, 8\}$ et $G = \{3, 4, 5, 6, 7, 8\}$, comme $v \notin X$, il faut appliquer la règle A, donc ne rien ajouter à l'ensemble dominant et appliquer

l'algorithme sur $G' = \{4, 5, 6, 7, 8\}$ et $X' = \{5, 6, 7, 8\}$.
La figure II.2 montre les étapes de l'algorithme jusqu'à devoir dominer que seulement les sommets $\{5, 6, 7, 8\}$.

Pour les cas où le sommet v est de degré 2, nous avons besoin du lemme suivant :

Lemme 1 :
Soit v un sommet de degré 2 dans G, et soit u_1, u_2 les deux voisins de v, donc pour tout sous-ensemble $X \subseteq V$, il y a un ensemble dominant minimum D pour X tel que l'une des règles suivantes est vraie :

— (i) $u_1 \in D$ et $v \notin D$.

— (ii) $v \in D$ et $u_1, u_2 \notin D$.

— (iii) $u_1 \notin D$ et $v \notin D$.

Preuve :

S'il existe un ensemble dominant minimum D pour X contenant u_1, alors il existe un ensemble dominant minimum D' pour X qui contient u_1 mais pas v. En fait, si $v \in D$, alors $D' = \{D - \{v\}\} \cup \{u_2\}$ est un ensemble dominant pour X tel que $|D'| \leqslant |D|$.
De la même façon, s'il existe un ensemble dominant minimum pour X qui contient u_2 alors il existe un ensemble dominant minimum pour X qui contient u_2 mais pas v.

Il nous reste donc cinq possibilités pour connaître l'appartenance de v, u_1, u_2 à un ensemble dominant minimum D pour X :
— (a) $u_1, u_2, v \notin D$.
— (b) $v \in D$ et $u_1, u_2 \notin D$.
— (c) $u_1 \in D$ et $v, u_2 \notin D$.
— (d) $u_2 \in D$ et $v, u_1 \notin D$.
— (e) $u_1, u_2 \in D$ et $v \notin D$.

En comparaison avec le lemme, on a :
— (i) est équivalent à (c) ou (e).
— (ii) est équivalent à (b).
— (iii) est équivalent à (a) ou (d).
Ce qui finalise notre preuve.

Considérons un sommet v de degré deux, selon que $v \in X$ ou non, nous branchons de différentes façons :

Cas C : Le sommet v de degré 2 et $v \in V - X$.
Soit u_1 et u_2 les deux voisins de v dans G, par le lemme 1, nous aurons trois sous-cas pour un ensemble dominant minimum :

1. **C.1 :** $u_1 \in D$ et $v \notin D$.
   Dans ce cas si D' est un ensemble dominant minimum pour $X - N[u_1]$ dans $G - \{u_1, v\}$, alors $D' \cup \{u_1\}$ est un ensemble dominant minimum pour X dans G, et donc nous continuons avec $G - \{u_1, v\}$ et $X - N[u_1]$.

2. **C.2 :** $v \in D$ et $u_1, u_2 \notin D$.
   Dans ce cas si D' est un ensemble dominant minimum pour $X - \{u_1, u_2\}$ dans $G - \{u_1, v, u_2\}$ alors $D' \cup \{v\}$ est un ensemble dominant minimum pour X dans G, et ainsi nous continuons avec $G - \{u_1, v, u_2\}$ et $X - \{u_1, u_2\}$.

3. **C.3 :** $u_1 \notin D$ et $v \notin D$.
   Dans ce cas, un ensemble dominant minimum pour X dans $G - \{v\}$ est aussi un ensemble dominant minimum pour X dans G et donc, nous continuons avec $G - \{v\}$ et X.

Cas D : Le sommet v est de degré deux et $v \in X$.
Soit u_1 et u_2 les deux voisins de v dans G, encore une fois selon le lemme 1, nous aurons trois sous-cas pour un ensemble dominant D :

1. **D.1 :** $u_1 \in D$ et $v \notin D$.
   Dans ce cas, si D' est un ensemble dominant minimum pour $X - N[u_1]$ dans $G - \{u_1, v\}$ alors $D' \cup \{u_1\}$ est un ensemble dominant minimum pour X dans G. Donc, nous continuons avec $G - \{u_1, v\}$ et $X - N[u_1]$.

2. **D.2 :** $v \in D$ et $u_1, u_2 \notin D$.
   Dans ce cas, si D' est un ensemble dominant minimum pour $X - \{u_1, v, u_2\}$ dans $G - \{u_1, v, u_2\}$, alors $D' \cup \{v\}$ est un ensemble dominant minimum pour X dans G. Donc nous continuons avec $G - \{u_1, v, u_2\}$ et $X - \{u_1, v, u_2\}$.

3. **D.3 :** $u_1 \notin D$ et $v \notin D$.
   Alors $v \in X$ implique que $u_2 \in D$, nous utilisons cela si D' est un ensemble dominant minimum pour $X - N[u_2]$ dans $G - \{v, u_2\}$ alors

$D' \cup \{u_2\}$ est un ensemble dominant minimum pour X dans G. Ainsi, nous continuons avec $G - \{v, u_2\}$, et $X - N[u_2]$.

La figure II.3, montre la première étape de l'algorithme sur un graphe à 9 sommets. Supposons que le sommet 1 est le sommet sélectionné par l'algorithme, comme le sommet 1 est de degré deux, l'application des règles de branchements D_1, D_2, et D_3 est établis.

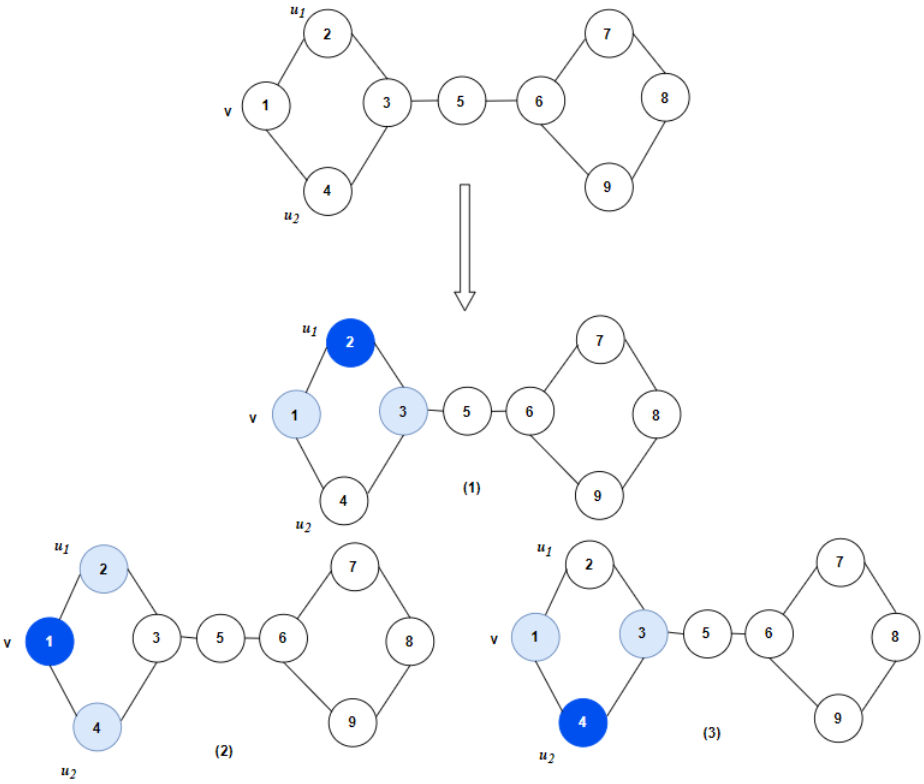

FIGURE II.3 – Illustration des règles de branchement D_1, D_2 et D_3.

Après la première étape, on a $D_1 = \{2\}$, $D_2 = \{1\}$, $D_3 = \{4\}$.

À l'étape suivante, des sommets seront supprimés et l'algorithme est exécuté récursivement sur les sous-graphes restants.
La figure II.4 montre les sommets qui doivent être dominés dans chaque branche. Par exemple, dans la branche (1) seuls les sommets $\{4, 5, 6, 7, 8, 9\}$ doivent être dominés.

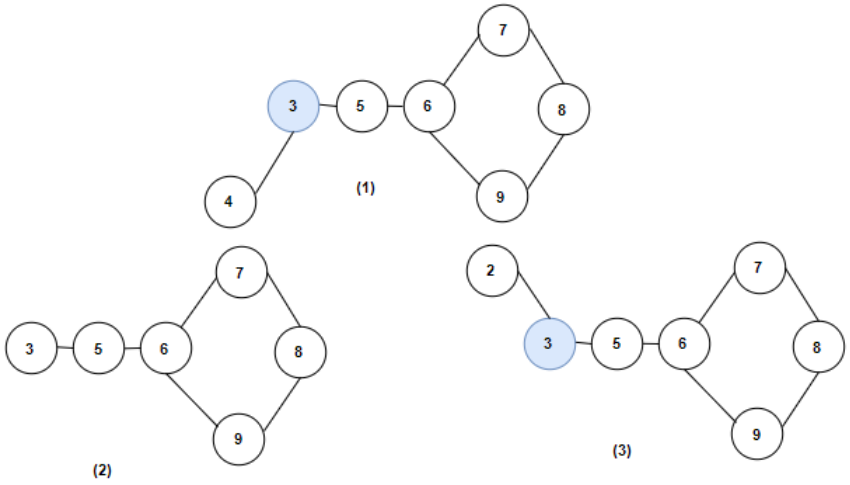

FIGURE II.4 – Illustration des règles de branchement D_1, D_2 et D_3.

II.2.2 Pseudo-code

Le pseudo-code ci-dessous, montre la gestion du cas de base, c'est à dire lorsque tous les sommets sont de degré 0 ou au moins 3 :

Algorithme : baseCase(G, X)
Entrées : un graphe G, un ensemble de sommets X
Sorties : un ensemble dominant minimum de X

si *tout les sommets de* X *sont de degré* 0 **alors**
  retourner X
fin
$X_0 = \{v \in X | d(v) = 0\}$
$X_{minDegree3} = \{v \in X | d(v) > 2\}$
subsets = ensemble des sous-ensembles de $X_{minDegree3}$ de taille 1 à 3∗ taille($X_{minDegree3}$)/8
mds = l'ensemble minimum de l'ensemble *subsets* qui est MDS de $X_{minDegree3}$
Ajouter X_0 à *mds*
retourner *mds*

Algorithme 1 : La gestion du cas de base.

Le pseudo-code ci-dessous, montre l'algorithme global pour tous types de graphe.

Algorithme : generalAlgo(G, X)
Entrées : un graphe G, un ensemble de sommets X
Sorties : un ensemble dominant minimum de X

```
1  si taille(X) = 0 alors
2  |  retourner {}
3  fin
4  Choisir v tel que d(v) = 1 ou 2
5  si v est vide alors
6  |  retourner baseCase(G, X)
7  fin
8  si d(v) = 1 alors
9  |  si v ∈ V − X alors
10 |  |  retourner generalAlgo(G − {v}, X − {v})
11 |  fin
12 |  sinon si v ∈ X alors
13 |  |  choisir w tel que w et le voisin unique de v
14 |  |  retourner {w}∪ generalAlgo(G − {v, w}, X − N[w])
15 |  fin
16 fin
17 sinon si d(v) = 2 alors
18 |  u1, u2 = voisins de v
19 |  si v ∈ V − X alors
20 |  |  C1 = {u1}∪ generalAlgo(G − {u1, v}, X − N[u1])
21 |  |  C2 = {v}∪ generalAlgo(G − {u1, v, u2}, X − {u1, u2})
22 |  |  C3 = generalAlgo(G − {v}, X)
23 |  |  retourner min(C1, C2, C3)
24 |  fin
25 |  sinon si v ∈ X alors
26 |  |  D1 = {u1}∪ generalAlgo(G − {u1, v}, X − N[u1])
27 |  |  D2 = {v}∪ generalAlgo(G − {u1, v, u2}, X − {u1, v, u2})
28 |  |  D3 = {u2}∪ generalAlgo(G − {u2, v}, X − N[u2])
29 |  |  retourner min(D1, D2, D3)
30 |  fin
31 fin
```

Algorithme 2 : Algorithme général.

II.2.3 Illustration

Soit $G = (V, E)$ le graphe de la figure II.5.
Au départ l'ensemble $X = V$.

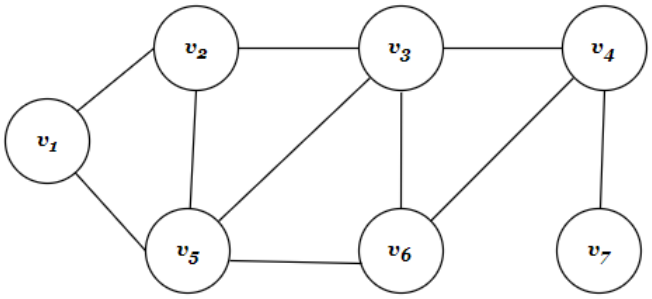

FIGURE II.5 – Graphe de départ.

Il y a un sommet de degré 1 (le sommet v_7). Puisque $v_7 \in X$, nous appliquons le cas B, dans les notations de ce cas, v correspond à v_7 et w correspond à v_4. L'algorithme spécifie qu'il existe un ensemble dominant minimum pour X (donc V ici) qui contient v_4, mais pas v_7.
Il y'aura un appel récursif sur $X' = X - N[w] = \{v_1, v_2, v_5\}$ et $G' = G - \{v, w\}$.
La figure II.6 est une représentation de l'étape suivante.
$G' = \{v_1, v_2, v_3, v_5, v_6\}$ et les sommets dans X' sont $\{v_1, v_2, v_5\}$:

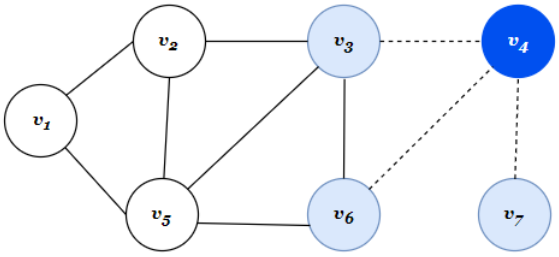

FIGURE II.6 – Première étape.

Nous sommes donc dans la configuration décrite dans l'étape précédente, avec G' et X' pour trouver D'. Si D' est un ensemble dominant minimum pour le graphe G' alors $D' \cup \{w\}$ sera un ensemble dominant minimum pour notre graphe de départ.

Comme il n'y a plus de sommets de degré 1, nous allons donc chercher un sommet de degré 2. Il existe deux : v_1 (qui est dans X') et v_6 (qui n'est pas dans X'). Nous pouvons donc choisir d'appliquer l'étape suivante sur v_1 (ce qui serait un cas D) ou v_6 (ce qui ferait un cas C). Alors, choisissons arbitrairement de travailler avec v_1. Les deux voisins de v sont notés u_1 et u_2 sur la figure II.7.

Pour trouver ce qui est D', l'ensemble dominant de G' par rapport à X', il y a trois branches à appliquer (D_1, D_2 et D_3) :

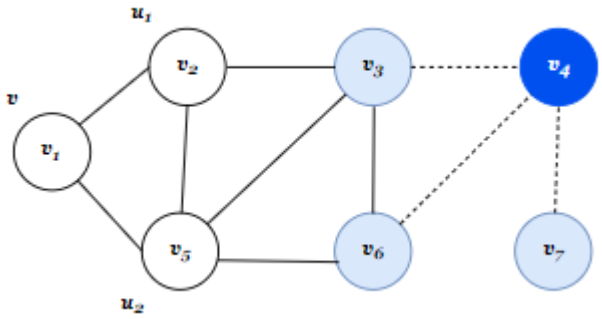

FIGURE II.7 – Deuxième étape.

- Sous-cas D_1 :

Dans ce cas, nous supposons que $u_1 \in D'$ et $v \notin D'$ et que si D'' est un ensemble dominant minimum pour $X'' = X' - N[u_1]$ dans $G'' = G' - \{u_1, v\}$, alors $D'' \cup \{u_1\} = \{v_2\}$ sera notre D'. Il y aura donc un appel récursif sur G'' et X''. Nous ne continuons pas la récursivité dans cette branche car X'' est vide mais nous rappelons que puisque D'' est vide, nous avons $D' = u_1 = v_2$, donc $D = \{v_2, v_4\}$ est la meilleure solution dans cette branche. Cette sous-branche nous apprend qu'on peut avoir $|D| = 2$.

- Sous-cas D_2 :

Dans ce cas, nous supposons que $v \in D'$ et $u_1, u_2 \notin D'$ et si D'' est un ensemble dominant minimum pour $X'' = X' - \{u_1, u_2, v\}$ dans $G'' = G' - \{u_1, u_2, v\}$ alors $D'' \cup v$ sera notre D'. Il y aura donc un appel récursif sur G'' et X''. Mais aussi, X'' étant vide, cela signifie qu'on peut avoir $|D| = 2$ avec $D = \{v_1, v_4\}$ qui est également une solution optimale du problème de départ.

- Sous-cas D_3 :

Dans ce cas, nous supposons que $u_1, v \notin D'$. Alors comme $v \in X'$, ça implique que u_2 doit appartenir à D' afin de dominer v. Si D'' est un ensemble dominant pour $X'' = X' - N[u_2]$ dans $G'' = G' - \{v, u_2\}$, alors $D'' \cup \{u_2\}$ serait notre D'. Il y aura donc un appel récursif sur G'' et X'', mais aussi, X'' étant vide, cela signifie que nous pouvons avoir $|D| = 2$ avec $D = \{v_5, v_4\}$ qui est aussi une solution optimale du problème de départ.

II.2.4 Analyse de la complexité

Théorème
Un ensemble dominant minimum d'un graphe G à n sommets peut être calculé en un temps exponentiel égal à $O(1,93782^n)$.

Preuve
Pour analyser le temps d'exécution de l'algorithme, notons $T(n)$ le nombre d'appels dans le pire des cas parmi les appels récursifs effectués par l'algorithme pour un graphe à n sommets.
L'ensemble des appels récursifs effectués par l'algorithme forme un arbre. La racine de cet arbre représente le premier appel. Chaque nœud a un ou trois fils (un fils pour les cas A et B, et trois fils pour les cas C et D). En ce qui concerne les feuilles de l'arbre , elles correspondent à la règle : pas de sommet de degré 1 ou 2.

Dans les cas A et B, nous avons la récurrence associée $T(n) \approx T(n-1)$, dans le cas C on a $T(n) \approx T(n-1) + T(n-2) + T(n-3)$, et dans le cas D on a $T(n) \approx 2 * T(n-2) + T(n-3)$.

Le pire cas des cas se présente lorsque $T(n) \approx T(n-1) + T(n-2) + T(n-3)$. Pour résoudre les récurrences de ce type, on sait par des méthodes mathématiques qu'elles satisfont $T(n) \in O(c^n)$ où c est l'unique racine réelle positive du polynôme $x^3 = x^2 + x + 1$.
La seule racine positive est $1,8393$. Ainsi, le pire cas est en $0(1,8393^n)$.

Par conséquent, la partie la plus coûteuse de l'algorithme est la phase de vérification de tous les sous-ensembles de taille au plus $3t/8$ où $t \leqslant n$.
Comme déjà discuté, ceci peut être effectué en $O(1.93782^n)$.

II.3 Graphes de degré maximum au plus égal à 3

Les graphes de degré maximum $\leqslant 3$ ont des nombres de domination élevés, le premier algorithme ne se comporte pas bien sur ces graphes.
Une explication possible pour ce problème est que l'algorithme général peut dans certains cas, chercher un MDS de taille $\gamma(G) - 1$ même si un MDS de taille plus petite a été trouvé auparavant dans la phase de recherche.

Dans cette partie, on présente un algorithme plus précis pour les graphes de degré maximum $\leqslant 3$. L'algorithme est basé sur la technique d'élagage d'un arbre de recherche, et une propriété des ensembles dominants minimums dans un graphe de degré maximum $\leqslant 3$ présentée dans le lemme suivant :

Lemme 2
Soit $G = (V, E)$ un graphe tel que $\Delta(G) \leqslant 3$, alors il existe un ensemble dominant minimum D avec les deux propriétés suivantes :

— (i) Chaque composant connexe de $G[D]$ est soit un sommet isolé, soit une arête isolée.

— (ii) Si deux sommets $x, y \in D$ forment une arête isolée dans $G[D]$, alors x et y ont un degré trois dans G, et $N(x) \cap N(y) = \emptyset$.

Pour rappel, un sommet isolé est un sommet qui n'a pas de voisins, alors qu'une arête isolée est une arête qui n'est adjacente à aucune autre arête.

Preuve
Soit D l'ensemble dominant minimum de G contenant le nombre maximum de sommets isolés dans $G[D]$.
Si $G[D]$ a un sommet x de degré trois, alors $D - \{x\}$ est un ensemble dominant minimum plus petit de G (contradiction), donc le degré maximum de $G[D]$ est $\leqslant 2$.
Supposons maintenant que $G[D]$ contient un sommet y de degré deux.
Si le degré de y dans G est égal à deux : alors $D - \{y\}$ est un ensemble dominant plus petit de G (contradiction).
Sinon, Soit z l'unique voisin de y dans G qui n'est pas dans D. Si $z \in N[D - \{y\}]$, alors $D - \{y\}$ est un ensemble dominant plus petit de G (une autre contradiction).
Enfin, si $z \notin N[D - \{y\}]$ alors $D_1 := (D \cup \{z\}) - \{y\}$ est un autre ensemble dominant de G avec un plus grand nombre de sommets isolés dans $G[D_1]$ que dans $G[D]$, cette contradiction conclut la preuve de la propriété (i).

Pour prouver la propriété (ii), montrons d'abord que si deux sommets $x, y \in D$ sont adjacents, alors ils sont de degré trois dans G.
Supposons que y est de degré inférieur à trois dans G. Il est clair que y ne peut pas être de degré 1, sinon, $D - \{y\}$ est un ensemble dominant minimum (contradiction).
Supposons maintenant que y est de degré deux et que $z \neq x$ soit le deuxième voisin de y :
- Si $z \in N[D - \{y\}]$, alors $D - \{y\}$ est un ensemble dominant de taille plus petite que D (contradiction).
- Si $z \notin [D - \{y\}]$, alors $D_2 := (D - \{y\}) \cup \{z\}$ est un ensemble dominant minimum avec un plus grand nombre de sommets isolés dans $G[D_2]$ que dans $G[D]$ (une autre contradiction).

Enfin, prouvons que $N(x) \cap N(y) = \emptyset$ dans G :
Par contradiction, supposons que $N(x) \cap N(y) \neq \emptyset$.
Si $N[x] \subseteq N[y]$, alors $D - \{x\}$ est un ensemble dominant et si $N[y] \subseteq N[x]$, alors $D - \{y\}$ est aussi un ensemble dominant. Dans les deux cas, cela contredit notre

hypothèse de départ. Par conséquent, $N(x) = \{y, w, u\}$ avec $N(x) - N(y) = \{w\}$ et $N(x) \cap N(y) = \{u\}$.
- Si $w \in N[D - \{x\}]$, alors $D - \{x\}$ est un ensemble dominant plus petit (contradiction).
- Si $w \notin N[D - \{x\}]$, alors $D_3 := (D - \{x\}) \cup \{w\}$ est un ensemble dominant minimum de G avec un plus grand nombre de sommets isolés dans $G[D_3]$ que dans $G[D]$ (une autre contradiction).

Maintenant, nous construisons un algorithme d'arbre de recherche en prenant en considération la restriction de l'espace de recherche garanti par le lemme 2, c'est-à-dire que pour tout graphe $G = (V, E)$ tel que $\Delta(G) \leqslant 3$, seuls les ensembles de sommets $D \subseteq V$ qui satisfont les propriétés du lemme 2 doivent être inspectés.

II.3.1 Règles de branchement

L'idée principale est de brancher dans des sous-cas jusqu'à ce que nous obtenions des graphes de degré maximum égal à 2, et pour de tels graphes, un ensemble dominant minimum peut être calculé en temps linéaire. Ceci est possible, car chaque composant connexe de degré maximum égal à 2 est soit un chemin soit un cycle. L'algorithme reçoit en entrée le graphe G et à chaque étape, selon la branche, il choisit quel sommet rajouter à l'ensemble dominant D et, récursivement brancher sur certains sous-graphes plus petits.

$G = (V, E)$ et $D = \emptyset$ correspondent à la racine de l'arbre de recherche, chaque nœud de l'arbre de recherche correspond à un sous-graphe induit $G[V']$ de G et un ensemble dominant partiel $D \subseteq V - V'$ de G, D est choisi auparavant de faire partie de l'ensemble dominant dans la branche du nœud courant. Chaque nœud feuille de l'arbre est un sous-graphe induit $G[V']$ de G de degré maximum égal à 2.

Pour chaque nœud interne de l'arbre de recherche, l'exécution de l'algorithme se déroule comme suit : l'algorithme choisit un sommet x voisin d'un sommet y de degré 3, tel que x a le plus petit degré. Puis, selon les propriétés de x, il se branche dans divers sous-cas.
Supposons que $(G[V'], D)$ corresponds à un nœud de l'arbre de recherche, et que $\Delta(G[V']) \leqslant 2$, donc l'ensemble dominant minimum D' de $G[V']$ sera calculé en temps polynomial, et $D \cup D'$ sera retourné comme MDS de cette branche. À chaque niveau de l'arbre de recherche, l'algorithme choisit l'ensemble D de taille minimale entre les différentes branches.

Cet algorithme va choisir un sommet x de degré 3 une seule fois et seulement si tous les sommets du graphe d'entrée sont de degré 3, pour cette raison ce cas n'est pas étudié dans l'analyse de la complexité de l'algorithme. Voici les différents autres sous-cas de l'algorithme :

Cas 1 : x est un sommet de degré 1.

Soit y de degré 3 le voisin de x, et z_1, z_2 les deux autres voisins de y, clairement on choisit $y \in D$ et on branche de la façon suivante :

A : $y \in D$ est un sommet isolé dans $G[D]$, on rajoute y à D et on continue avec $G - N[y]$. Comme $d(y) = 3$, le nombre d'appel récursif est égal à $T(n-4)$.

B : $y, z_i \in D$, $i \in \{1,2\}$ est une arête isolée dans $G[D]$, on rajoute y, z_i, $i \in \{1,2\}$ à D et on continue avec $G - (N[y] \cup N[z_i])$. Par le lemme 2, $d(z_i)$ doit être égal à 3. Le nombre d'appel récursif dans cette branche est au plus égal à $2T(n-6)$.

Dans le graphe de la figure II.8, (A) correspond à l'application de la règle A sur notre graphe de départ, $(B.1)$ et $(B.2)$ représentent l'application de la règle B.

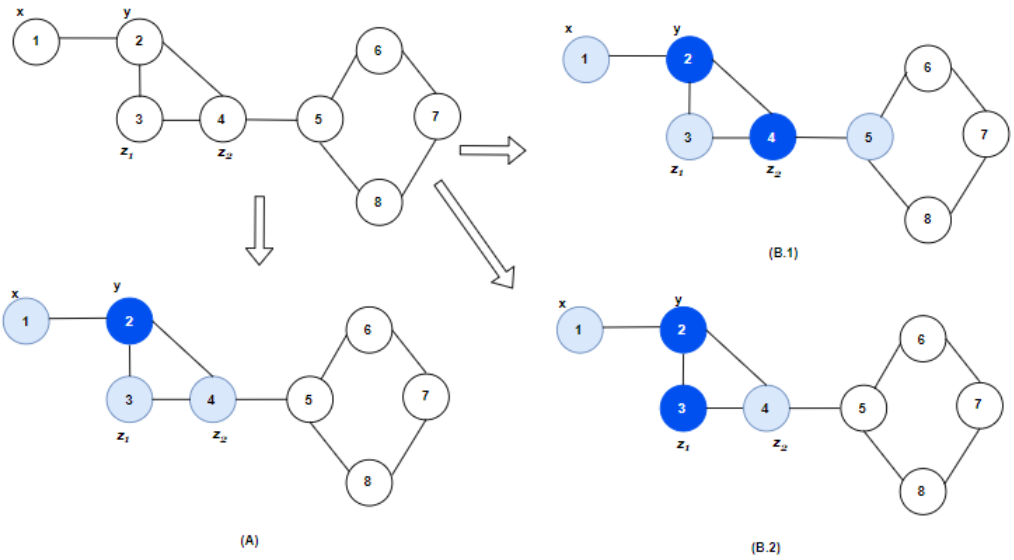

FIGURE II.8 – Première étape.

Puisque $(B.1)$ et $(B2)$ ne remplissent pas les critères du lemme 2, ($d(z_1) = 2$ et $N(y_1) \cap N(z_2) \neq \emptyset$), ces branches ne seront pas explorées, seule la branche (A) sera explorée pour déterminer l'ensemble dominant du graphe de départ. Il faut noter que l'ensemble dominant retourné par la branche (A) ne représente que l'ensemble dominant du graphe depuis cette branche, l'algorithme va devoir itérer sur plusieurs branches pour déterminer l'ensemble dominant minimum du graphe de départ. Dans notre exemple de la figure II.8, l'algorithme va aussi brancher depuis le sommet 3, puisque le sommet 3 est le voisin de degré minimum d'un

sommet de degré 3 (le sommet 4), ainsi depuis le sommet 6 ou 7, car ces deux sommets sont voisins de degré minimum d'un sommet de degré 3 (le sommet 5).

Cas 2 : x est un sommet de degré 2.

Soit y_1, y_2 les deux voisins de x, soit y_1 le sommet de degré 3.
sous-cas 2.1 : y_1 et y_2 sont adjacents : donc il existe un MDS non contenant x, donc soit $y_1 \in D$, soit $y_2 \in D$.
→ *Cas 2.1.1 :* $\mathrm{d}(\mathrm{y}_2) = 2$.
Donc soit y_1 est un sommet isolé dans $G[D]$ et on le rajoute à D, soit l'arête $y_1, z \in D$ avec z le troisième voisin de y_1. Le nombre d'appel récursif dans cette branche est au plus égal à $T(n-4) + T(n-6)$.

→ *Cas 2.1.2 :* $\mathrm{d}(\mathrm{y}_2) = 3$.
Soit z_i le troisième voisin de y_i pour $i = \{1, 2\}$, Alors soit $y_1 \in D$ et on continue avec $G - N[y_1]$, soit $y_2 \in D$ et on continue avec $G - N[y_2]$, ou soit on choisit $y_i, z_i \in D$ et on continue avec $G - (N[y_i] \cup N[z_i])$. Le nombre d'appel récursif dans cette branche est au plus égal à $2T(n-4) + 2T(n-6)$.

Dans la figure suivante, on a $d(x) = 2$, $d(y_1) = 3$ et comme y_1, y_2 sont adjacents, nous appliquons le sous-cas 2.1.2 à notre graphe de départ.

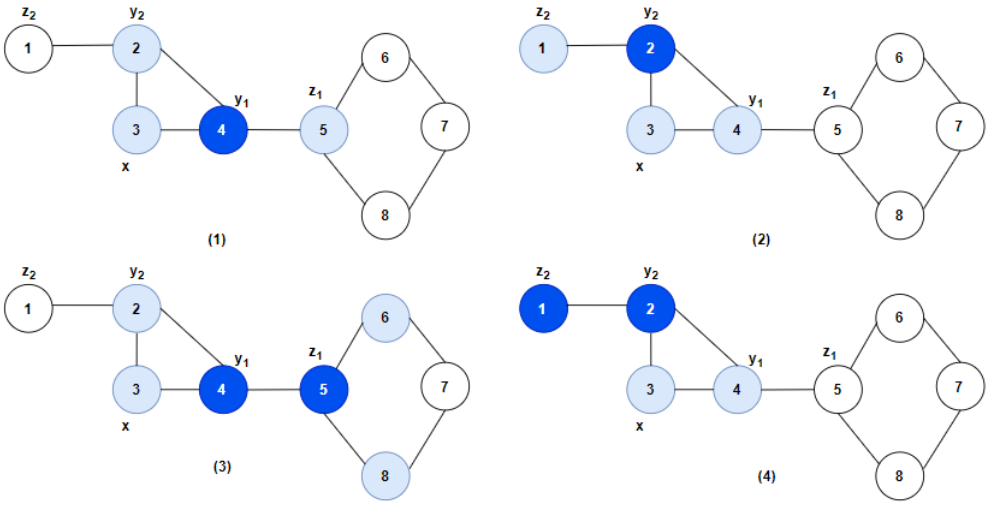

FIGURE II.9 – Première étape.

Cette fois, c'est la branche (4) qui ne sera pas considérée parmi les appels récursifs, puisque $d(z_2) < 3$.

sous-cas 2.2 : y_1 et y_2 ne sont pas adjacents.
→ *Cas 2.2.1 :* $\mathrm{d}(\mathrm{y}_2) = 2$.
Soit z_{11}, z_{12} les autres voisins de y_1, et soit z_2 l'autre voisin de y_2.

1. Cas A : $x \in D$ est un sommet isolé dans $G[D]$, on rajoute x à D et on continue avec $G - N[x]$. Comme $d(x) = 2$, le nombre d'appel récursif dans cette branche est égal à $T(n-3)$.

2. Cas B : $y_i \in D$ pour $i = 1, 2$ est un sommet isolé dans $G[D]$, on rajoute y_i à D et on continue avec $G - N[y_i]$. Comme $d(y_1) = 3$ et $d(y_2) = 2$, le nombre d'appel récursif dans cette branche est au plus égal à $T(n-3) + T(n-4)$.

3. Cas C : y_1, z_{1j} pour $j \in \{1, 2\}$ est une arête isolée dans $G[D]$, on rajoute y_1,z_{1j} pour $j \in \{1, 2\}$ à D et on continue avec $G - (N[y_1] \cup N[z_{1j}])$. Par le lemme 2, $d(z_{1j})$ doit être égal à 3. Le nombre d'appel récursif dans cette branche est au plus égal à $2T(n-6)$.

Dans la figure suivante, comme $d(x) = 2$, $d(y_1) = 3$, $d(y_2) = 2$ et comme y_1, y_2 ne sont pas adjacents, nous appliquons le sous-cas 2.2.1.

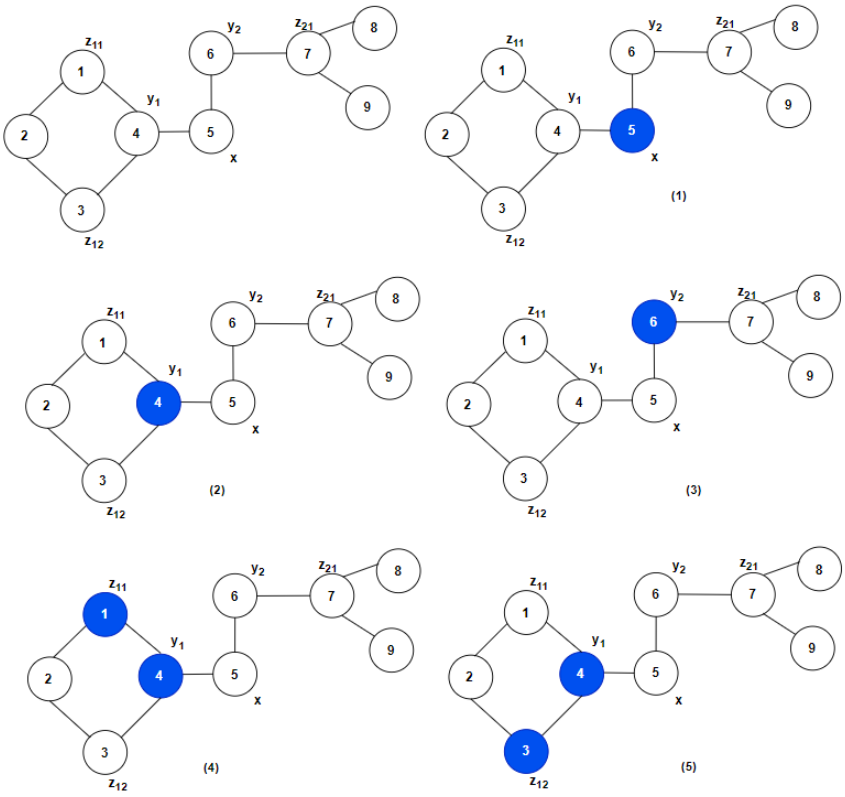

FIGURE II.10 – Première étape.

Puisque $d(z_{11}) \neq 3$ et $d(z_{12}) \neq 3$, les branches 4 et 5 ne seront pas explorées.

→ *Cas 2.2.2 :* $\mathrm{d}(\mathrm{y}_2) = 3$.

Soit z_{11} et z_{12} les deux autres voisins de y_1, et z_{21} z_{22} les deux autres voisins de y_2.

1. Cas A : $x \in D$ est un sommet isolé dans $G[D]$, on rajoute x à D et on continue avec $G - N[x]$, le nombre d'appel récursif dans cette branche est égal à $T(n-3)$.

2. Cas B : $y_i \in D$ pour $i = 1, 2$ est un sommet isolé dans $G[D]$, on rajoute y_i à D et on continue avec $G - N[y_i]$. Comme $d(y_1) = 3$ et $d(y_2) = 3$, le nombre d'appel récursif dans cette branche est au plus égal à $2T(n-4)$.

3. Cas C : y_i, z_{ij} pour $j \in \{1, 2\}$ est une arête isolée dans $G[D]$, on rajoute y_i, z_{ij}, $i, j \in \{1, 2\}$ à D et on continue avec $G - (N[y_i] \cup N[z_{ij}])$. Par le lemme 2, $d(z_{ij})$ doit être égal à 3. Le nombre d'appel récursif dans cette branche est au plus égal à $4T(n-6)$.

La figure II.11 et II.12, montrent les différentes branches obtenues après avoir appliqué le cas 2.2.2 sur un graphe à 12 sommets :

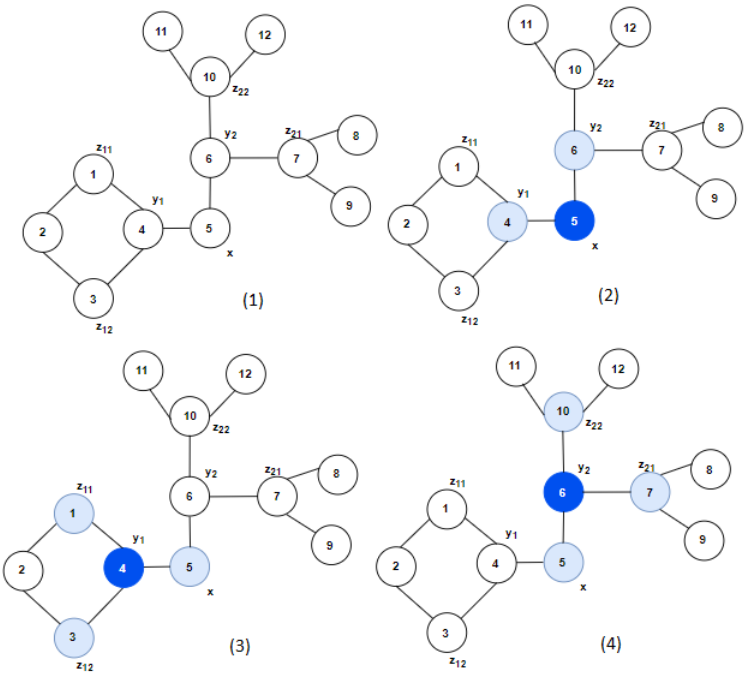

FIGURE II.11 – Première étape-1.

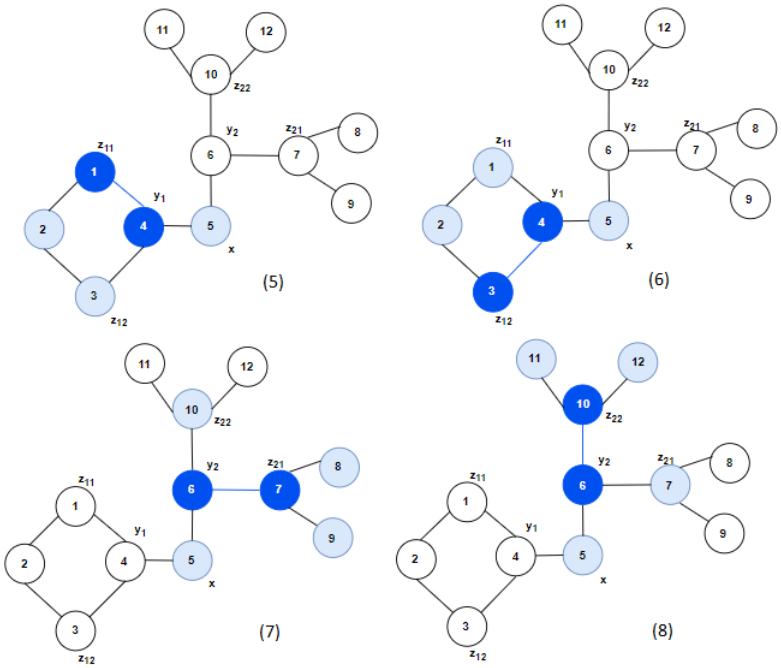

FIGURE II.12 – Première étape-2.

Cette fois-ci, deux branches ne seront pas explorées, il s'agit de la branche 5 et la branche 6.

II.3.2 Pseudo-code

Le pseudo-code de la figure suivante, montre l'algorithme spécifique pour les graphes de degré maximum au plus égal à 3. Pour raison de lisibilité, le code de vérification des conditions du lemme 2 n'est pas introduit.

Algorithme : mdsForGraphAtMost3Degree(G)
Entrées : un graphe G
Sorties : un ensemble dominant minimum
$mdsList = \{\}$
$listOfNeighbors$ = {liste des voisins de degré minimum des sommets de degré 3}
si $isCubic(G)$ **alors**
  retourner mdsfromCubicGraph(G)
fin
pour tous les $x_i \in listOfNeighbors$ **faire**
  si $d(x) = 1$ **alors**
    y= voisin de x avec $d(y) = 3$
    z_1, z_2 : les deux autres voisins de y
    D_1 = mdsForGraphAtMost3Degree $(G - N[y]) \cup \{y\}$
    D_2 = mdsForGraphAtMost3Degree $(G - \{N[y] \cup N[z_1]\}) \cup \{y, z_1\}$
    D_3 = mdsForGraphAtMost3Degree $(G - \{N[y] \cup N[z_2]\}) \cup \{y, z_2\}$
    Ajouter min(D_1, D_2, D_3) à $mdsList$.
  fin
  sinon si $d(x) = 2$ **alors**
    Choisir y_1, y_2 voisins de x avec $d(y_1) = 3$
    si *voisin*(y_1, y_2) **alors**
      si $d(y_2) = 2$ **alors**
        D_1 = mdsForGraphAtMost3Degree $(G - N[y_1]) \cup \{y_1\}$
        Choisir z tel que z est le troisième voisin de y_1
        D_2 = mdsForGraphAtMost3Degree $(G - \{N[y_1] \cup N[z]\}) \cup \{y_1, z\}$
        Ajouter min(D_1, D_2) à $mdsList$
      fin
      sinon si $d(y_2) = 3$ **alors**
        Soit z_1 = troisième voisin de y_1
        Soit z_2 = troisième voisin de y_2
        D_1 = mdsForGraphAtMost3Degree $(G - N[y_1]) \cup \{y_1\}$
        D_2 = mdsForGraphAtMost3Degree $(G - N[y_2]) \cup \{y_2\}$
        D_3 = mdsForGraphAtMost3Degree $(G - \{N[y_1] \cup N[z_1]\}) \cup \{y_1, z_1\}$
        D_4 = mdsForGraphAtMost3Degree $(G - \{N[y_2] \cup N[z_2]\}) \cup \{y_2, z_2\}$
        Ajouter min (D_1, D_2, D_3, D_4) à $mdsList$
      fin
    fin
    sinon
      si $d(y_2) = 2$ **alors**
        D_1 = mdsForGraphAtMost3Degree $(G - N[x]) \cup \{x\}$
        D_2 = mdsForGraphAtMost3Degree $(G - N[y_1]) \cup \{y_1\}$
        D_3 = mdsForGraphAtMost3Degree $(G - N[y_2]) \cup \{y_2\}$
        D_4 = mdsForGraphAtMost3Degree $(G - \{N[y_1] \cup N[z_{11}]\}) \cup \{y_1, z_{11}\}$
        D_5 = mdsForGraphAtMost3Degree $(G - \{N[y_1] \cup N[z_{12}]\}) \cup \{y_1, z_{12}\}$
        Ajouter min$(D_1, D_2, D_3, D_4, D_5)$ à $mdsList$
      fin
      sinon si $d(y_2) = 3$ **alors**
        D_1 = mdsForGraphAtMost3Degree $(G - N[x]) \cup \{x\}$
        D_2 = mdsForGraphAtMost3Degree $(G - N[y_1]) \cup \{y_1\}$
        D_3 = mdsForGraphAtMost3Degree $(G - N[y_2]) \cup \{y_2\}$
        D_4 = mdsForGraphAtMost3Degree $(G - \{N[y_1] \cup N[z_{11}]\}) \cup \{y_1, z_{11}\}$
        D_5 = mdsForGraphAtMost3Degree $(G - \{N[y_1] \cup N[z_{12}]\}) \cup \{y_1, z_{12}\}$
        D_6 = mdsForGraphAtMost3Degree $(G - \{N[y_2] \cup N[z_{21}]\}) \cup \{y_2, z_{21}\}$
        D_7 = mdsrorGraphAtMost3Degree $(G - \{N[y_2] \cup N(z_{22})\}) \cup \{y_2, z_{22}\}$
        Ajouter min$(D_1, D_2, D_3, D_4, D_5, D_6, D_7)$ à $mdsList$
      fin
    fin
  fin
fin
retourner min($mdsList$)

Algorithme 3 : Algorithme pour les graphes de degré maximum $\leqslant 3$.

II.3.3 Illustration

Soit $G = (V, E)$ le graphe de la figure II.13. Selon l'algorithme, on recherche d'abord dans G les sommets de degré 3. Comme il y en a plusieurs, choisissons le sommet v_2. Alors $x = v_1$ puisque v_1 est le voisin de v_2 avec un degré minimum.

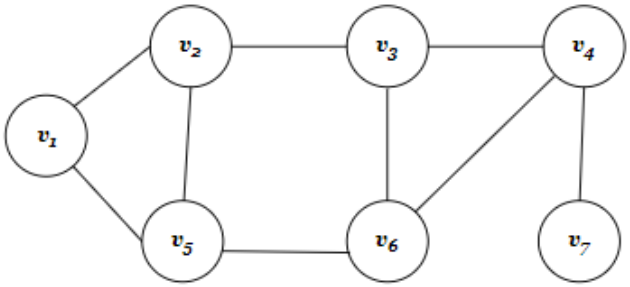

FIGURE II.13 – Graphe de départ.

On a $d(v_1) = 2$, puisque $y_1 = v_2$ et $y_2 = v_5$ sont adjacents et voisins de x et comme $d(y_2) = 3$, nous exécutons le cas 2.1.2 pour ce graphe. Soit $z_1 = v_3$ et $z_2 = v_6$ les deux autres voisins de y_1 et y_2 respectivement. Maintenant, nous traversons quatre branches différentes :

1.1 Nous ajoutons $y_1 = v_2$ à D et faisons un appel sur $G' = G - N[y_1]$. Ceci atteint un nœud feuille (graphe représenté sur la figure II.14, puisque ce nouveau graphe G' n'a pas de sommet de degré 3, et comme c'est un chemin nous ajoutons v_4 à D et retournons $D = \{v_2, v_4\}$ comme l'ensemble dominant minimum de cette branche.

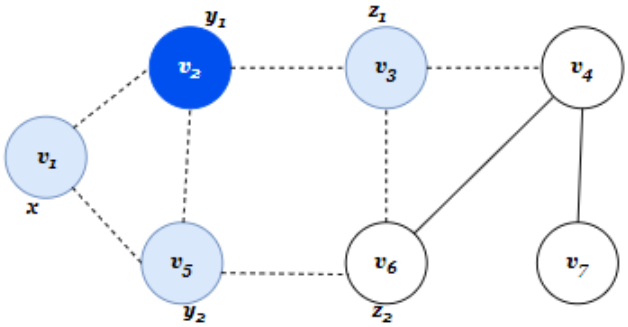

FIGURE II.14 – Première branche.

1.2 Nous ajoutons $y_2 = v_5$ à D et faisant un appel sur $G' = G - N[y_2]$. Ceci atteint aussi un nœud feuille (graphe sur la figure II.15), puisque ce nouveau graphe G' n'a pas de sommet de degré 3, et comme c'est un chemin, nous ajoutons v_4 à D et retournons $D = \{v_5, v_4\}$ comme l'ensemble dominant minimum de cette branche.

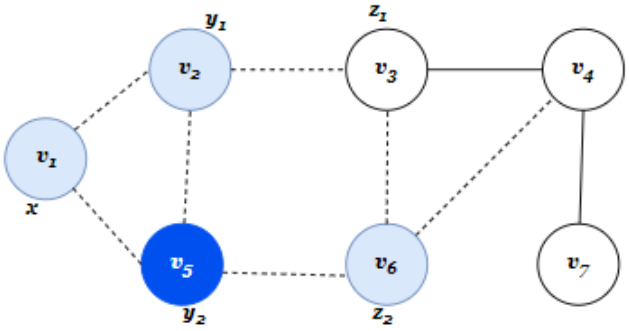

FIGURE II.15 – Deuxième branche.

1.3 Nous ajoutons l'arête $\{y_1, z_1\}$ égale à $\{v_2, v_3\}$ à D, et faisons un appel récursif sur $G' = G - \{N[y_1] \cup N[z_1]\}$. Ceci atteint un cas de base, puisque le nouveau graphe n'a pas de sommet de degré 3. En effet, c'est un sommet isolé (graphe dans la figure II.16). Pour cela, nous ajoutons v_7 à D et retournons $D = \{v_2, v_3, v_7\}$ comme l'ensemble dominant minimum de cette branche.

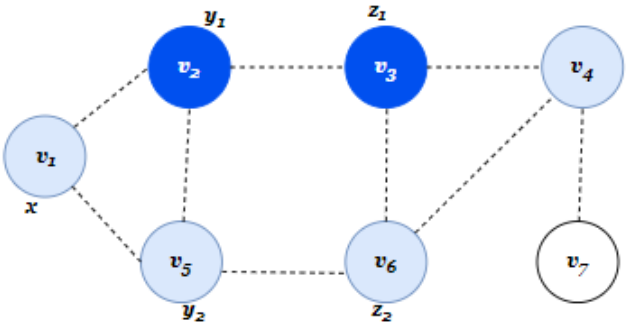

FIGURE II.16 – Troisième branche.

1.4 Nous ajoutons l'arête $\{y_2, z_2\}$ égale à $\{v_5, v_6\}$ à D et faisons un appel récursif sur $G' = G - \{N[y_2] \cup N[z_2]\}$. Ceci atteint aussi un cas de base. En effet, c'est aussi un sommet isolé (graphe dans la figure II.17). Pour cela, nous ajoutons v_7 à D et retournons $D = \{v_5, v_6, v_7\}$ comme l'ensemble dominant minimum de cette branche.

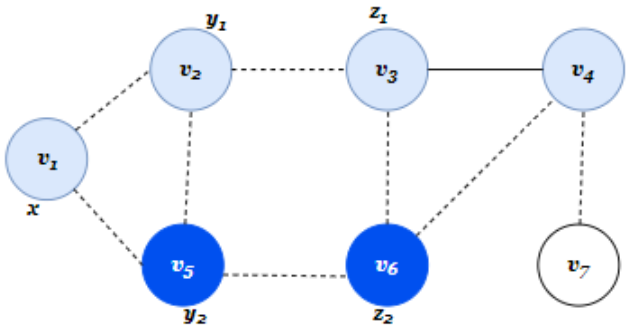

FIGURE II.17 – Quatrième branche.

À partir des 4 branches, l'ensemble dominant minimum est retourné par les branches 1.1 et 1.2. Ainsi, les deux solutions $\{v_2, v_4\}$, $\{v_5, v_4\}$ représentent des ensembles dominants minimums du graphe de départ.

Notons, que ces solutions ne représentent que des solutions depuis le branchement du sommet $x = v_1$, l'algorithme va aussi itérer sur les autres sommets voisins d'un sommet de degré trois. Dans notre exemple, nous avons illustrés qu'une seule itération de l'algorithme.

II.3.4 Analyse de la complexité

L'analyse de la complexité se fait de la même façon que l'algorithme vu dans la section précédente, au total on obtient la relation de récurrence suivante : $T(n) \approx T(n-3) + 2T(n-4) + 4T(n-6)$ et qui correspond au cas 2.2.2, en résolvant cette occurrences, $1,51433$ est l'unique racine réelle positive du polynôme $x^6 = x^3 + 2x^2 + 4$. Le pire cas de $T(n)$ est en $O(1,51433^n)$.

Chapitre III

Algorithmes exacts basés sur le problème du Set Cover

III.1 Introduction

L'approche discutée dans l'article [4], se caractérise par la réduction du problème d'ensemble dominant minimum au problème de couverture par ensembles (*Set Cover, SC*) [5]. Ce dernier, fait aussi partie des problèmes NP-complet [15].

Définition
Étant donné un ensemble d'éléments $U = \{1, 2, ...n\}$, et une famille s de sous-ensembles de U, une couverture par ensembles de U consiste à couvrir tous les éléments de U avec une sous-famille de s la plus petite possible. On dit qu'un élément e est couvert par un ensemble S si $e \in S$.

La version de décision associée est la suivante :
Problème : couverture par ensembles (*Set Cover, SC*)
Entrée : un entier k, un ensemble U fini et un sous-ensemble s de l'ensemble des parties de U.
Question : existe-il un sous-ensemble C de s, de taille inférieure à k, tel que l'union des éléments présents dans C est égal à U ?

Exemple
Considérons un ensemble de cinq éléments à couvrir : $U = \{1, 2, 3, 4, 5\}$. Soit la collection des sous-ensembles $s = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$. On essaye de couvrir tous les éléments de U avec le nombre minimum de sous-ensembles de s.
Pour notre instance (s, U), $C_1 = \{\{1, 2, 3\}, \{3, 4\}, \{4, 5\}\}$ est une couverture de U, puisque chaque élément de U est dans au moins un des sous-ensembles de C_1. Cependant, la couverture qui utilise le moins de sous-ensembles est celle des sous-ensembles $\{\{4, 5\}, \{1, 2, 3\}\}$, car elle contient deux sous-ensembles et il n'est pas

possible de couvrir l'ensemble U avec un seul sous-ensemble.

Réduction

Nous pouvons réduire le problème de l'ensemble dominant minimum à un problème de couverture par ensembles en convertissant une instance $G = (V, E)$ du premier problème à une instance (s, U) du deuxième. Cette réduction se fait de la façon suivante : on introduit un ensemble S_i pour chaque sommet v_i du graphe G, chaque ensemble S_i contiendra le voisinage fermé d'un sommet v_i du graphe G, c'est-à-dire, $s := \{N[v] | v \in V\}$, et $U := V$. Nous pouvons donc résoudre le problème d'ensemble dominant minimum sur un graphe $G = (V, E)$ à n sommet en utilisant un algorithme de couverture par ensembles s'exécutant sur une instance (s, U) avec n ensembles et un univers de taille n.

Exemple

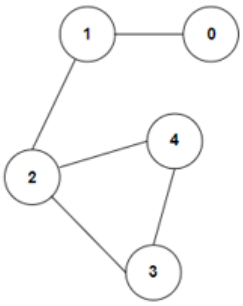

FIGURE III.1 – Graphe de départ.

L'instance du problème de couverture par ensembles associée au problème d'ensemble dominant minimum du graphe G de la figure III.1 est la suivante :

$U = \{0, 1, 2, 3, 4\}$ et $s = \{\{0, 1\}, \{0, 1, 2\}, \{1, 2, 3, 4\}, \{2, 3, 4\}, \{2, 3, 4\}\}$.

On observe que la couverture par ensembles $\{\{1, 2, 3, 4\}, \{0, 1\}\}$ correspond bien à une solution optimale du problème d'ensemble dominant minimum sur le graphe associé. En effet, les deux sous-ensembles $\{1, 2, 3, 4\}, \{0, 1\}$ correspondent bien aux sommets 2 et 0 du graphe G. Ces deux sommets forment bien un ensemble dominant minimum dans le graphe G.

III.2 L'approche naïve

III.2.1 Introduction

L'algorithme trivial présenté dans l'article [4], permet de sélectionner simplement le plus grand sous-ensemble $S \in s$ et considère deux instances : une dans laquelle nous prenons l'ensemble S dans la couverture de l'ensemble U et l'autre dans laquelle nous le rejetons. À chaque niveau, l'algorithme sélectionne simplement le plus grand sous-ensemble S de notre instance et branche différemment. Dans la branche où le sous-ensemble S est pris dans la couverture, on enlève S de s et tous les éléments de S de l'univers U, nous supprimons donc pour tout $S' \in s \backslash \{S\}$ tous les éléments de S. Dans l'autre branche, où S n'est pas choisi on retire seulement S de s. Ensuite, l'algorithme résout récursivement les sous-problèmes générés et renvoie la plus petite couverture par ensembles possible. L'algorithme s'arrête quand l'ensemble s est vide. Ensuite, il vérifie si les solutions retournées par les appels récursifs correspondent bien à des couvertures d'ensembles.

Exemple

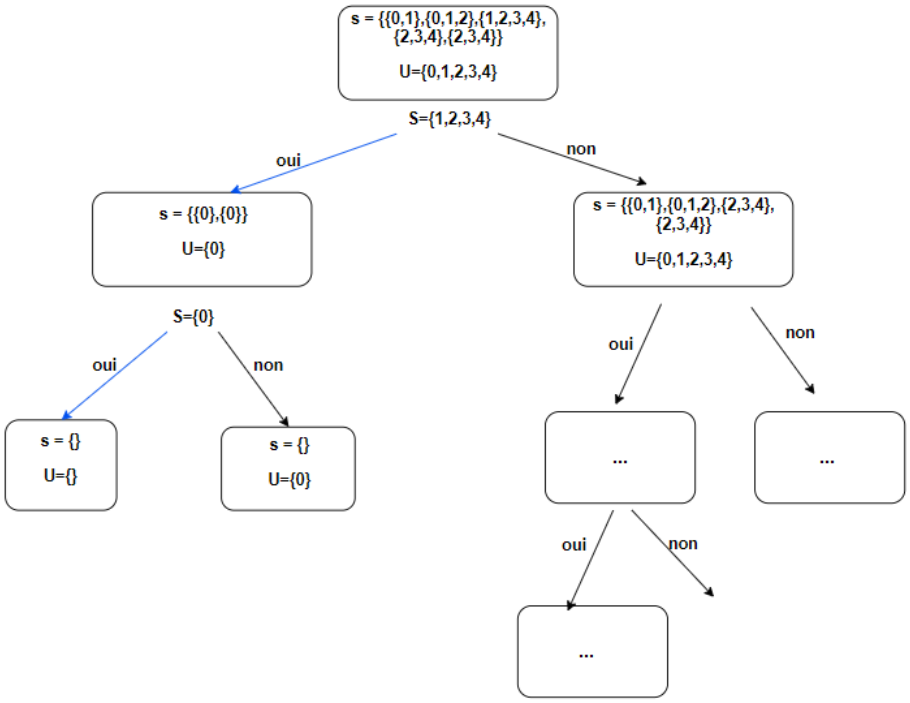

FIGURE III.2 – Arbre généré par les appels récursifs.

Dans la figure III.2, on présente l'exécution de l'algorithme trivial sur l'instance :
$U = \{0, 1, 2, 3, 4\}$ et $s = \{\{0, 1\}, \{0, 1, 2\}, \{1, 2, 3, 4\}, \{2, 3, 4\}, \{2, 3, 4\}\}$.

Dans la première itération, $S = \{1, 2, 3, 4\}$ est le sous-ensemble de s de taille maximale, dans la première branche qui correspond à la condition oui, S est sélectionné dans la solution, et l'algorithme est exécuté récursivement sur la nouvelle instance : $U = \{0\}$ et $s = \{\{0\}, \{0\}\}$, dans l'autre branche, S n'est pas choisi dans la couverture et l'algorithme est exécuté récursivement sur la nouvelle instance $U = \{0, 1, 2, 3, 4\}$ et $s = \{\{0, 1\}, \{0, 1, 2\}, \{2, 3, 4\}, \{2, 3, 4\}\}$.

À chaque niveau de l'arbre, le même traitement est fait, le cas de base est lorsque s est vide. $\{\{0, 1\}, \{1, 2, 3, 4\}\}$, représentent la solution optimale de cette instance.

Il est intuitif que l'exécution de l'algorithme trivial est exponentiel vu qu'on traverse un arbre binaire et qu'on considère tous les sous-ensembles de s ; puisque $|s|$ est égal au nombre de sommet de notre graphe de départ. Notre algorithme a donc une complexité en $O(2^n)$.

III.2.2 Pseudo-code

Le pseudo-code de l'algorithme trivial est présenté ci-dessous :

Algorithme : trivialSC(s, U)
Entrées : une collection de sous-ensembles s, un ensemble d'éléments U
Sorties : une couverture de U qui utilise le moins de sous-ensembles

```
si s est vide alors
    si U est vide alors
        retourner {}
    fin
    retourner null
fin
Soit S ∈ s : le sous-ensemble avec la cardinalité maximale
C1 = {S} ∪ trivialSC({S'\S | S' ∈ s\{S}}, U\S)
C2 = trivialSC(s\{S}, U)
retourner min(C1, C2)
```

Algorithme 4 : Algorithme de couverture par ensembles trivial.

III.3 Algorithme amélioré

L'algorithme amélioré présenté dans l'article [4], utilise une technique dite brancher et réduire (*Branch and Reduce*). Cette technique est un paradigme de programmation basé sur la technique de branchement. Elle consiste en une série de règles de réduction, une série de règles de branchement et une procédure pour décider quelle règle de branchement appliquer sur une instance donnée. Les règles de réduction transforment une instance avec certaines propriétés spécifiques en une instance équivalente plus petite en temps polynomial. Un tel algorithme, applique d'abord de manière exhaustive les règles de réduction. Lorsque l'instance ne satisfait plus aux propriétés d'une des règles de réduction, l'algorithme décide quelle règle de branchement à appliquer. La règle de branchement sélectionnée génère alors une série d'instances de problèmes plus petits qui seront résolus récursivement.

Dans leur article [4], les auteurs présentent deux algorithmes améliorés pour résoudre le problème de la couverture par ensembles. La différence entre les deux est que le deuxième algorithme inclus plus de règles de réduction que le premier. Dans cette section nous allons présenter le premier algorithme qui contient trois règles de réduction.

III.3.1 Règle d'élément unique

S'il existe un élément e avec une fréquence qui vaut 1, c'est-à-dire qu'il n'y a qu'un seul ensemble S dans s qui contient e, alors nous devons obligatoirement ajouter S à notre solution afin de couvrir l'élément e, puisque aucun autre ensemble ne couvrira l'élément e.

Par exemple, soit $U = \{1, 2, 3, 4\}$ et $s = \{\{1, 2\}, \{1, 2, 3\}, \{3, 4\}\}$, comme la fréquence de l'élément 4 est égale à 1, nous devons donc ajouter l'ensemble $\{3, 4\}$ à la solution. L'algorithme fait un appel récursif sur $s = \{\{1, 2\}, \{1, 2\}\}$ et $U = \{1, 2\}$.

Règle de réduction 1
Si $\exists$ un élément $e \in U$ d'une fréquence égale à 1 **Alors** :
Retourner $\{R\} \cup trivialSC(\{R' \backslash R | R' \in s \backslash \{R\}\}, U \backslash R)$ où R est l'ensemble unique qui contient e.

III.3.2 Éliminer les sous-ensembles inclus

Il est évident que s'il y a deux ensembles identiques dans s, on peut en retirer un sans risquer de laisser un élément non couvert. S'il y a deux ensembles Q et R tels que $R \subseteq Q$, alors effectivement Q couvre aussi les éléments couverts par R. Donc nous supprimons R de s et nous continuons avec les ensembles restants.

Soit $U = \{1, 2, 3, 4\}$ et $s = \{\{1, 2\}, \{1, 2, 3\}, \{3, 4\}\}$. Nous pouvons supprimer l'ensemble $\{1, 2\}$, puisqu'il est inclus dans $\{1, 2, 3\}$. Nous continuons donc, avec $U = \{1, 2, 3, 4\}$ et $s = \{\{1, 2, 3\}, \{3, 4\}\}$ sans risque de laisser certains éléments non couverts.

Règle de réduction 2
Si $\exists$ deux ensembles $Q, R \in s$ tel que $R \subseteq Q$ **Alors**
Retourner $trivialSC(s \backslash \{R\}, U)$

III.3.3 Arrêter quand la cardinalité des ensembles est au plus égale à deux

Si la cardinalité de tous les ensembles est au plus égale à 2, nous pouvons arrêter le branchement et trouver un SC minimum en temps polynomial. Cette solution requiert de réduire le problème du SC au problème de couverture par arêtes (*Edge Cover, EC*) [18].

Définition
Une couverture par arêtes d'un graphe G est un ensemble d'arêtes C, tel que chaque sommet de G est incident à au moins une arête de C.
La plus petite couverture par arêtes de G, est une couverture par arêtes utilisant le moins d'arêtes possible.
La figure III.3, montre des exemples de couvertures par arêtes minimaux.

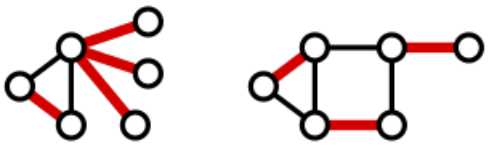

FIGURE III.3 – Différentes couvertures par arêtes minimaux [19].

Définition
Un couplage M de $G = (V, E)$ est un sous-ensemble des arêtes E deux à deux non adjacentes, c'est à dire que chaque sommet dans V est incident à au plus une arête de M. Intuitivement, nous pouvons dire que deux arêtes de M n'ont pas de sommet en commun.

Un couplage M est maximum s'il contient le plus grand nombre possible d'arêtes. Ainsi, pour tout autre couplage M', $|M| >= |M'|$.
Dans la figure III.4, on observe des couplages maximaux pour différents graphes.

L'algorithme d'*Edmonds* [20] peut trouver un couplage maximum en temps polynomial, la complexité de cet algorithme est en $O(N^4)$.

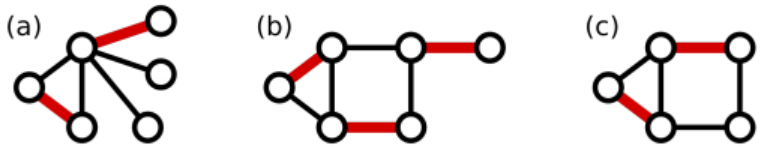

FIGURE III.4 – Différents couplages maximaux [21].

La plus petite couverture par arêtes dans le graphe G' peut être calculée en temps polynomial en cherchant un couplage maximum M dans G', et pour tout sommet u non couvert par l'ensemble M, nous rajoutons à M une arête arbitraire incidente à u.
Finalement, l'ensemble M représentera une couverture par arêtes du graphe G' et donc une couverture par ensembles de l'instance (s, U).

Réduction
Afin de réduire une instance (s, U) du problème de la couverture par ensembles en une instance du problème de la couverture par arêtes, nous procédons comme suit :
Considérons le graphe $G' = (V, E)$, tel que pour tout élément $e_i \in U$, nous rajoutons un sommet v_i dans G', dès lors, nous avons $|V| = |U|$, et rajoutons une arête $\{u, v\}$ à G' pour tout sous-ensemble $S = \{u, v\}$ de s.
La plus petite couverture par arêtes dans le graphe construit G' correspond bien à la plus petite couverture par ensemble de U dans s.

Exemple
Soit $U = \{1, 2, 3, 4, 5\}$ et $s = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{3, 5\}, \{4, 1\}, \{4, 5\}\}$.
Afin de calculer la plus petite couverture par ensembles de l'instance (s, U), nous cherchons un couplage maximum dans le graphe construit de la figure III.5.
Un possible couplage maximum du graphe G' est l'ensemble $C = \{\{1, 2\}, \{3, 5\}\}$.
Comme le sommet 4 n'est incident à aucune arête de l'ensemble C, nous rajoutons à C une des arêtes incidentes au sommet 4 dans G', prenons par exemple l'arête $\{1, 4\}$. L'ensemble $C = \{\{1, 2\}, \{3, 5\}, \{1, 4\}\}$ représente une plus petite couverture par arêtes dans G', et aussi une plus petite couverture par ensembles de l'instance (s, U).

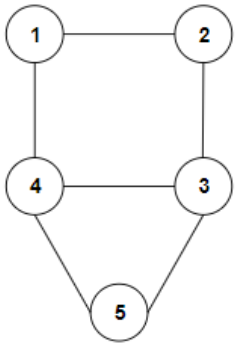

FIGURE III.5 – Graphe construit G'.

Règle de réduction 4 **:**
Soit $S \in s$ l'ensemble avec cardinalité maximale
Si $|S| \leqslant 2$ **Alors**
retourner la plus petite couverture par ensembles calculé en temps polynomial utilisant l'algorithme du couplage maximum.

III.3.4 Pseudo-code

Algorithme : improvedSC(s, U)
Entrées : une collection de sous-ensembles s, ensemble d'éléments U
Sorties : une couverture de U qui utilise le moins de sous-ensembles

si *s est vide* **alors**
  retourner {}
fin
Soit $S \in s$: le sous-ensemble avec la cardinalité maximale
si $\exists e \in U$ *tel que* $f(e) = 1$ **alors**
  retourner $\{R\} \cup$ *improvedSC* $(R' \backslash R | R' \in s \backslash \{R\}, U)$ *où* $e \in R$
fin
sinon si $\exists Q, R \in s$ *tel que* $R \subseteq Q$ **alors**
  retourner improvedSC$(S \backslash \{R\}, U)$
fin
sinon si $|S| \leqslant 2$ **alors**
  retourner *MSC calculé en temps polynomial par l'algorithme Maximum Matching*
fin
$C_1 = \{S\} \cup$ *trivialSC*$(\{S' \backslash S | S' \in s \backslash \{S\}\}, U \backslash S)$
$C_2 =$ *trivialSC*$(s \backslash \{S\}, U)$
retourner *min(C_1, C_2)*

Algorithme 5 : Algorithme de couverture par ensembles amélioré.

III.3.5 Analyse de la complexité

La complexité de l'algorithme amélioré est en $O(1.5169^n)$, l'analyse de cette complexité dans l'article [4], est faite par une technique dite : mesurer et conquérir (*Mesure and Conquer*) [22]. Dans cette analyse, une mesure non-standard est soigneusement choisie pour être utilisée. Ceci est en contraste avec les analyses classiques qui reposent sur des mesures simples, tel que le nombre de sommets dans un graphe. La preuve de cette analyse est au-delà de la portée de ce mémoire. Le détail complet se trouve dans l'article [4].

Chapitre IV

Méthodes approchées

IV.1 Introduction

Dans les chapitres précédents, nous avons vu quelques algorithmes de complexité exponentielle pour résoudre le problème d'ensemble dominant minimum. Malheureusement, en pratique, lorsque les instances sont grandes, il est quasi-impossible d'obtenir une solution optimale. Pour cette raison, il existe des algorithmes approchés pouvant produire de bonnes solutions en un temps raisonnable (polynomial).

Dans la première partie de ce chapitre, nous allons présenter 3 heuristiques proposées dans l'article [6].
Les trois heuristiques sont :

- l'algorithme *greedy* ;
- l'algorithme *greedy random* ;
- l'algorithme *greedy reverse*.

Dans ces trois heuristiques, deux notions sont réutilisées : la couverture, et le poids. Le premier terme est utilisé pour indiquer qu'un sommet fait partie des sommets dominés par l'ensemble dominant D. Ainsi, pour un graphe G, nous disons qu'un sommet v est couvert par l'ensemble D si v appartient au voisinage fermé de $G[D]$. Le deuxième terme indique le nombre de sommets qu'un sommet peut encore couvrir. Pour un sommet v_i, cette somme est initialisée à $d(v_i) + 1$ et elle se réduit au cours des itérations jusqu'à aboutir à la valeur 0. Quand le poids de chaque sommet atteint la valeur zéro, le graphe est dominé.

Les deux premiers algorithmes fonctionnent de la manière suivante :
Au départ l'ensemble D est vide, à chaque itération nous ajoutons un sommet à D jusqu'à ce que ce dernier devient un ensemble dominant.
Pour le troisième algorithme, le processus fonctionne dans le sens inverse, c'est à dire qu'au départ D contient tous les sommets du graphe et à chaque itération on enlève un sommet de D à condition que D reste toujours un ensemble dominant.

Dans la deuxième partie de ce chapitre, nous allons présenter un algorithme génétique. L'algorithme va faire évoluer simultanément un ensemble de solutions un certain nombre d'itérations. L'idée principale consiste à utiliser l'ensemble de solutions dans le but de créer de nouvelles solutions meilleures.

IV.2 L'algorithme greedy

Initialement, l'ensemble dominant D est vide et à chaque itération de l'algorithme, on ajoute un sommet à D jusqu'à ce que D devient un ensemble dominant. Le sommet sélectionné pour appartenir à D est choisi à condition qu'il couvre le nombre maximum de sommets non couverts à l'itération précédente. Si plusieurs sommets répondent à ce critère, le sommet à ajouter est choisi au hasard parmi ces sommets.

IV.2.1 Pseudo-code

La figure 6 présente le pseudo-code de l'algorithme *greedy*, dans une première étape, le poids de chaque sommet v_i est initialisé à la valeur $d(v_i) + 1$, puisqu'au départ chaque sommet v_i peut dominer $d(v_i) + 1$ sommets.
Dans le pseudo-code, $weight_i$ dénote le poids du sommet v_i, c'est à dire le nombre de sommets précédemment non couverts par le sommet v_i, et qui seraient couverts lors de l'ajout de ce sommet à D.
La variable $covered_i$ est vraie si le sommet v_i est couvert par l'ensemble D.

Algorithme : greedy(G)
Entrées : un graphe G
Sorties : un ensemble dominant D

```
1  D = ∅
2  pour tous les v_i ∈ G faire
3  |   weight_i = 1 + d(v_i)
4  |   covered_i = false
5  fin
6  répéter
7  |   v = chooseVertex(weight)
8  |   si v ≠ −1 alors
9  |   |   ajouter v à D
10 |   |   adjustWeights(G, weight, covered, v)
11 |   fin
12 jusqu'à v = −1;
13 retourner D
```

Algorithme 6 : Greedy.

À chaque itération, et après l'ajout d'un sommet à l'ensemble D, un ajustement de poids des sommets est établi. La figure 7, présente le pseudo-code de l'opération de l'ajustement des poids des sommets.

Algorithme : adjustWeights(D, *weight*, *covered*, v_i)
Entrées : l'ensemble D, le vecteur *weight*, le vecteur *covered*, l'indice du sommet v_i

```
1  weight_i = 0
2  pour tous les v_j voisin de v_i tel que weight_j > 0 faire
3  |   si !covered_i alors
4  |   |   weight_j − −
5  |   fin
6  |   si !covered_j alors
7  |   |   covered_j = true
8  |   |   weight_j − −
9  |   |   pour tous les v_k voisin de v_j faire
10 |   |   |   si weight_k > 0 alors
11 |   |   |   |   weight_k − −
12 |   |   |   fin
13 |   |   fin
14 |   fin
15 fin
16 covered_i = true
```

Algorithme 7 : AdjustWeights.

Le pseudo-code suivant présente l'opération de recherche du sommet au poids maximum :

Algorithme : chooseVertex(*weight*)
Entrées : le vecteur *weight*
Sorties : un sommet de G qui couvre le maximum de sommets pas encore couverts

```
1 M = max_{1≤i≤n} weight_i
2 si M = 0 alors
3 |   retourner −1
4 fin
5 sinon
6 |   S = {v_i | weight_i = M}
7 |   retourner aléatoirement un élément de S
8 fin
```

Algorithme 8 : ChooseVertex.

IV.2.2 Illustration

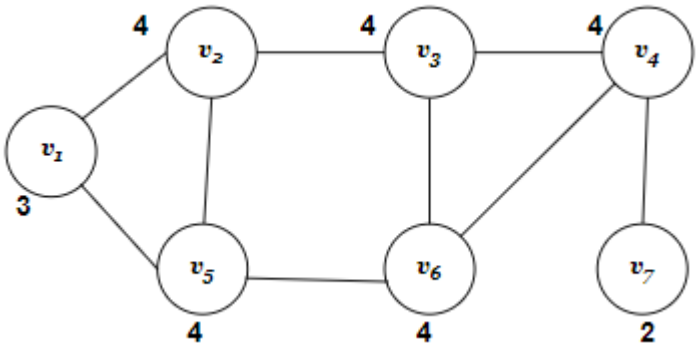

FIGURE IV.1 – Graphe de départ.

Soit $G = (V, E)$, le graphe de la figure IV.1. Initialement, chaque sommet est non couvert. Les chiffres devant les sommets représentent le poids de chaque sommet. Puisqu'il existe plusieurs sommets qui ont un poids maximum, nous choisissons un sommet au hasard à partir de l'ensemble $\{v_2, v_3, v_4, v_5, v_6\}$, prenons aléatoirement v_2. Nous ajoutons v_2 à D, par conséquent et après la première étape, nous avons $D = \{v_2\}$.

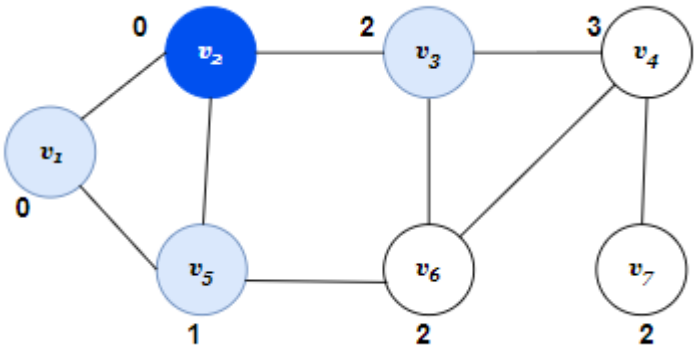

FIGURE IV.2 – Première étape.

Après l'ajustement de cette itération, le poids de plusieurs sommets sera modifié, le sommet v_1 par exemple n'aura plus de voisins à couvrir.

Dans la deuxième itération, le sommet v_4 est le sommet avec le poids maximum, nous le rajoutons à D. Après l'ajustement de cette itération, nous avons $D = \{v_2, v_4\}$.
Dans le graphe de la figure IV.3, tous les sommets sont couverts. Puisque tous les sommets ont un poids égal à 0, l'algorithme se termine et renvoie l'ensemble dominant $D = \{v_2, v_4\}$.

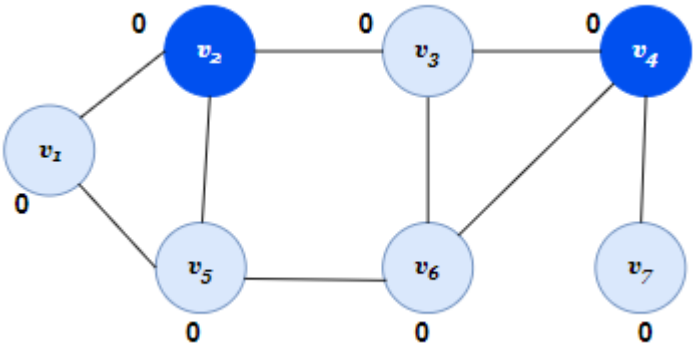

FIGURE IV.3 – Deuxième étape.

L'ensemble $D = \{v_2, v_4\}$ représente bien une solution optimale pour le graphe de départ. Cependant, si dans la première étape, l'algorithme commence par sélectionner le sommet v_3 pour appartenir à D, nous aurons toujours un ensemble dominant de taille 3, ce qui n'est pas la solution optimale pour notre graphe de départ.

IV.2.3 Analyse de la complexité

Pour un graphe G à n sommets et m arêtes, chaque appel à $chooseVertex$ est au plus en $O(n)$. Le temps total passé en $adjustWeights$ tout au long de l'exécution de l'algorithme est en $O(m)$. Si d est la taille de l'ensemble dominant trouvé par l'algorithme, la complexité totale de l'algorithme est en $O(nd + m) \leqslant O(n^2)$.

IV.3 L'algorithme greedy random

Cet algorithme est similaire à *greedy*, la seule différence étant la façon dont les sommets sont ajoutés à l'ensemble dominant. La probabilité qu'un sommet soit choisi à une itération donnée est proportionnel au nombre des sommets supplémentaires qu'il peut couvrir. Le pseudo-code suivant présente le seul changement. La complexité de l'algorithme est également similaire à *greedy*.

IV.3.1 Pseudo-code

Algorithme : chooseVertex(weight)
Entrées : le vecteur weight
Sorties : un sommet choisi avec une certaine probabilité
1 **retourner** v_i *avec une probabilité égale à* $weight_i / \sum_j weight_j$

Algorithme 9 : ChooseVertex.

IV.4 L'algorithme greedy reverse

Cet algorithme fonctionne à l'inverse de l'algorithme *greedy*. Initialement, D contient tous les sommets du graphe. À chaque itération, nous retirons un sommet de D à condition que ce sommet ne soit pas le seul à couvrir un des sommets du graphe, en d'autres termes ; D sera toujours un ensemble dominant après la suppression de ce sommet.
À chaque itération, les sommets sont triés selon l'ordre croissant de leur degré. Le sommet avec le plus petit degré, qui ne couvre pas d'une façon unique un des sommets du graphe est choisi pour être retiré de l'ensemble D.

IV.4.1 Pseudo-code

La figure 10 présente le pseudo-code de l'algorithme *greedy reverse*.
La variable $uniquely_i$ est vraie si le retrait de v_i de l'ensemble D aurait pour résultat que D ne serait plus un ensemble dominant.

Algorithme : greedy reverse(G)
Entrées : un graphe G
Sorties : un ensemble dominant

```
1  trier les sommets de G par ordre croissant de leur degré
2  D = {V}
3  pour tous les v_i ∈ G faire
4  |  coveredby_i = d(v_i) + 1
5  |  si coveredby_i = 1 alors
6  |  |  uniquely_i = true
7  |  fin
8  |  sinon
9  |  |  uniquely_i = false
10 |  fin
11 fin
12 start = 0
13 répéter
14 |  v = chooseVertex(D, uniquely, start)
15 |  si v ≠ −1 alors
16 |  |  Supprimer v de D
17 |  |  adjust(D, v, coveredby, uniquely)
18 |  fin
19 jusqu'à v = −1;
20 retourner D
```

Algorithme 10 : Greedy reverse.

À chaque itération, et après la suppression d'un sommet de l'ensemble D, un ajustement de poids des sommets est établi. La figure 11 présente le pseudo-code de l'ajustement des poids. Pendant cette opération, le vecteur *uniquely* est mis à jour.

Algorithme : adjust(D, v_i, *coveredby*, *uniquely*)
Entrées : l'ensemble D, l'indice du sommet v_i, le vecteur *coveredby*, le vecteur *uniquely*

```
1  coveredby_i − −
2  pour tous les v_j voisin de v_i faire
3  |  si coveredby_i = 1 and v_j ∈ D alors
4  |  |  uniquely_j = true
5  |  fin
6  |  coveredby_j − −
7  |  si coveredby_j = 1 alors
8  |  |  si v_j ∈ D alors
9  |  |  |  uniquely_j = true
10 |  |  fin
11 |  |  sinon
12 |  |  |  pour tous les v_k voisin de v_j faire
13 |  |  |  |  si v_k ∈ D alors
14 |  |  |  |  |  uniquely_k = true
15 |  |  |  |  fin
16 |  |  |  fin
17 |  |  fin
18 |  fin
19 fin
```

Algorithme 11 : Adjust.

IV.4.2 Analyse de la complexité

Le temps total requis pour la méthode *chooseVertex* au pire des cas est en $O(n)$ à chaque appel. Ainsi, dans le pire des cas, le temps requis par *chooseVertex* tout au long de l'algorithme est en $O(nd)$ où $d = |D|$.
Le temps total passé par *Adjust* est en $O(m)$. Le temps requis pour le tri est en $O(nlogn)$. Ainsi, le temps total au pire des cas est en $O(nd+m+nlogn) \leqslant O(n^2)$.

IV.5 L'algorithme génétique

IV.5.1 Principe général

Les algorithmes génétiques [23] font partie de la famille des algorithmes évolutifs (*evolutionary algorithms*). Afin de résoudre les problèmes d'optimisations, les algorithmes de cette famille s'inspirent du principe d'évolution de Darwin [24].

Les algorithmes génétiques étant fortement liés aux phénomènes biologiques, ils imitent les 3 mécanismes suivants :

— **La sélection naturelle** : est le mécanisme qui favorise la reproduction et la survie des individus les mieux adaptés à l'environnement.

— **La reproduction par croisement** : est le fait qu'un individu hérite ses caractéristiques de ses parents, de sorte que le croisement de deux individus bien adaptés à leur environnement aura tendance à créer un nouvel individu bien adapté à l'environnement.

— **La mutation** : est l'apparition ou la disparition aléatoire de certaines caractéristiques, permettant ainsi d'introduire de nouvelles capacités d'adaptation à l'environnement.

Contrairement à un certain nombre d'approches qui manipulent une seule solution par itération, les algorithmes génétiques manipulent un groupe de solutions admissibles à chaque itération du processus de recherche. Après la génération de la population initiale, et à travers les itérations, l'algorithme tente d'améliorer la qualité des solutions de la population en ayant recours aux mécanismes cités auparavant.
Ces mécanismes sont inspirés de la nature par les algorithmes génétiques afin de faire évoluer les populations de solutions. Un algorithme génétique est donc, basé sur une population d'individus dont chacun représente une solution candidate du problème. Le degré d'adaptation d'un individu à l'environnement est exprimé par la valeur de la fonction objective correspondante (*fitness function*).

La figure suivante, représente les principales étapes d'un algorithme génétique :

1 Initialiser une population de solution
2 **tant que** *critères d'arrêt non atteints* **faire**
3 | Sélectionner des combinaisons de la population
4 | Créer de nouvelles combinaisons par recombinaison et mutation
5 | Mettre à jour la population
6 **fin**
7 **retourner** la meilleure combinaison ayant appartenu à la population

Algorithme 12 : Algorithme génétique général.

IV.5.2 L'algorithme génétique pour l'ensemble dominant minimum

Dans leur article [25], les auteurs présentent un algorithme génétique pour résoudre le problème d'ensemble dominant minimum. Dans cette section, nous allons découvrir en grandes lignes leur approche.

Représentation

Afin de représenter les individus de la population, un codage binaire est utilisé. Chaque individu est représenté par un vecteur à n bits, où n représente le nombre de sommets d'un graphe G et chaque composante du vecteur est associée à un sommet de G, cette composante vaut 1 si et seulement si ce sommet appartient à D. La valeur 1 associée à l'indice i du vecteur, implique que le sommet i appartient à l'ensemble dominant. Par exemple, les deux vecteurs 101010 et 000101 représentent deux solutions associées au graphe de la figure IV.4.

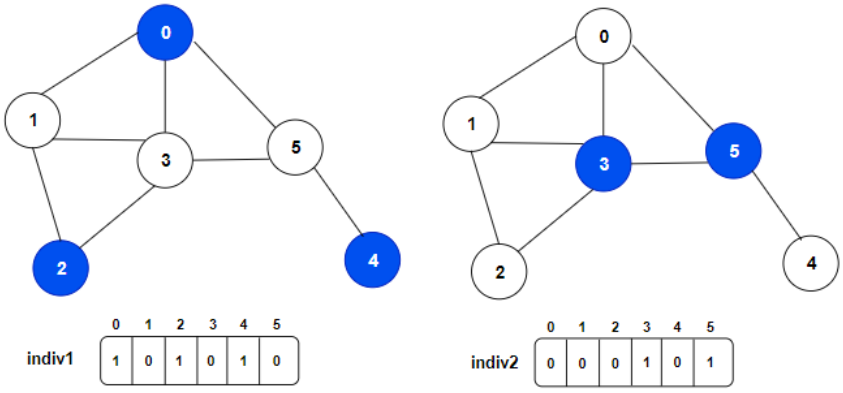

FIGURE IV.4 – Représentation d'individus.

Population initiale

La population initiale contient 100 individus, ils sont générés de façon aléatoire, tel que les bits du vecteur sont mis à 1 avec une probabilité égale à 0.3. Si un individu ne représente pas un ensemble dominant, la solution est modifiée en utilisant l'heuristique *greedy* avec une probabilité p_h égale à 0.6, dans les autres cas la solution est modifiée en ajoutant des sommets au hasard jusqu'à ce que la solution soit valide.
Dans une deuxième étape, un algorithme de minimisation est appliqué sur les individus afin d'essayer de réduire la valeur de la cardinalité de l'ensemble dominant. Cette deuxième étape se déroule de la façon suivante :
Si un sommet $v \in D$ et ses voisins, sont couverts par d'autres sommets de D, nous retirons le sommet v de l'ensemble D.

L'évaluation des individus

Afin d'évaluer les individus de la population, une fonction objective est définie pour produire un score pour chaque individu de la population. La valeur de la fonction objective d'un individu est égale à la cardinalité de l'ensemble dominant généré. Dans notre exemple de la figure IV.4, la valeur de la fonction objective associée à l'individu 1 est égale à 3, quant à l'individu 2, son score associé est égal à 2. Ce score représente la cardinalité de l'ensemble dominant minimum du graphe.

Sélection

En général, cette étape consiste à sélectionner les meilleurs individus de la population pour l'étape de croisement. Plusieurs méthodes de sélection existent. L'approche utilisée dans l'article [26], consiste à sélectionner deux individus de la population pour la phase de croisement, ces deux individus sont sélectionnés de la manière suivante : nous formons deux groupes d'individus, chaque groupe composé de deux individus tirés au hasard de la population, et avec une probabilité égale à 0.8, c.-à-d. à 80% des cas, on choisit depuis chaque groupe l'individu avec la meilleure fitness, dans les autres cas, on choisit l'autre individu du groupe.
Il est clair que cette stratégie de génération ne permet pas une construction rapide de nouvelles générations. Pour cette raison, une deuxième stratégie est implémentée et elle se déroule de la façon suivante : à chaque itération, la meilleure moitié de la population est sélectionnée pour construire de nouveaux individus, de cette façon plus d'individus sont créés.

Croisement

L'objectif de cette étape est d'utiliser les individus sélectionnés dans l'étape de sélection afin de créer de nouveaux individus. Il existe plusieurs méthodes de croisement :

— Le croisement simple (*one-point crossover*) : consiste à fusionner les particularités de deux individus à partir d'un pivot, afin d'obtenir un ou deux enfants.

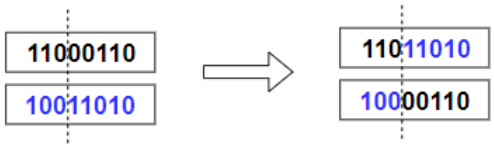

FIGURE IV.5 – Un croisement simple.

— Le double croisement (*two-point crossover*) : repose sur le même principe que le croisement simple, sauf qu'il y a deux pivots.

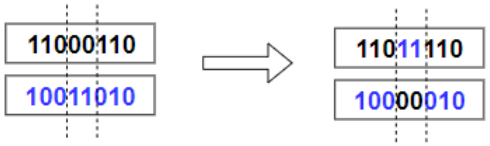

FIGURE IV.6 – Un double croisement.

— Le croisement uniforme (*uniform crossover*) : cette méthode est l'approche utilisée dans l'article [26], le croisement se déroule comme suit : soit p_1 et p_2 les deux individus sélectionnés dans l'étape de sélection et soit $f(p_1)$, $f(p_2)$ leur score associé. Les bits de l'individu généré sont hérités de p_1 avec une probabilité égale à $f(p_2)/f(p_1) + f(p_2)$, et sont hérités de p_2 avec une probabilité égale à $f(p_1)/f(p_1) + f(p_2)$..

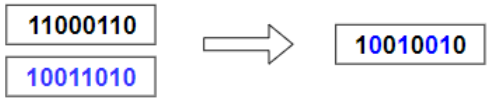

FIGURE IV.7 – Un croisement uniforme.

La génération de nouveaux individus n'est pas toujours dû à la recombinaison par croisement, mais selon une probabilité p_c égale à 0.7. Dans les autres cas, la génération est faite aléatoirement afin de diversifier la population.

Mutation

Cette étape permet de modifier de façon aléatoire certains composants de l'individu. Dans notre contexte, un simple bit flip avec une probabilité égale à 0.02 est appliqué sur les nouveaux individus.

Voici un exemple de mutation sur un individu :

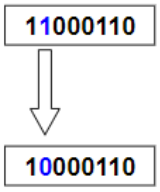

FIGURE IV.8 – Opération de mutation.

Mise à jour de la population
Une fois que de nouveaux individus sont créés dans l'étape de croisement/mutation, il faut sélectionner les individus qui vont continuer à participer dans l'amélioration de notre population.

Dans le cas de l'algorithme où un seul individu est créé pour intégrer la population à chaque itération, le nouvel individu va remplacer l'individu qui a le plus mauvais score. la figure 13 montre le pseudo-code de cet algorithme. La deuxième version de l'algorithme génétique se caractérise par la sélection de la moitié de la population pour la phase de croisement/mutation , un ensemble d'individus intègre la population à chaque itération.

IV.5.3 Pseudo-code

Entrées : le nombre d'itération maximum MAXGEN
Sorties : un ensemble dominant D

```
1  I := génération de la population initiale
2  F := le score du meilleur individu de I
3  b := meilleur individu de la population I
4  tant que gen ⩽ MAXGEN faire
5  |  si p ⩽ p_c alors
6  |  |  selectionner p_1 et p_2 de I
7  |  |  child := crossover(p_1, p_2)
8  |  |  child := mutation(child)
9  |  fin
10 |  sinon
11 |  |  child := randomGeneration()
12 |  fin
13 |  si p ⩽ p_h alors
14 |  |  child := heuristicRepair(child)
15 |  fin
16 |  sinon
17 |  |  child := randomRepair(child)
18 |  fin
19 |  child := minimise(child)
20 |  si isUnique(child) alors
21 |  |  remplacer le mauvais individu de I par child
22 |  |  si f(child) < F alors
23 |  |  |  F := f(child)
24 |  |  |  b := child
25 |  |  fin
26 |  |  gen++
27 |  fin
28 fin
```

Algorithme 13 : La première version de l'algorithme génétique.

Chapitre V

Analyse empirique

V.1 Introduction

Dans les précédents chapitres, nous avons vu en détails quelques algorithmes résolvant le MDS. Dans ce chapitre, nous allons faire une analyse empirique de ces différents algorithmes.
Dans un premier temps, et sur différents ensembles de graphes, nous allons comparer les performances des algorithmes exacts en fonction du temps. Dans une deuxième étape, nous allons comparer les heuristiques entre elles en fonction de la qualité des résultats. Ces deux comparaisons vont nous permettre de connaître les algorithmes les plus adaptés pour nos instances.

V.2 Environnement

L'ensemble de données
Pour bien mener cette analyse empirique, un ensemble d'instances différentes de graphes est utilisés.
Ces différents ensembles de graphes sont :

- Un premier ensemble de graphes contenant 12293433 graphes générés grâce à l'outil *GraPHedron* [27].
- Un deuxième ensemble de graphes contenant 176798 graphes de degré maximum plus petit ou égal à 3, générés grâce à l'outil *geng* [28].
- Un troisième ensemble provenant de la base de données *House of Graphs* [29], contenant des graphes qui ont un nombre de domination entre 8 et 10.
- Un dernier ensemble de graphes, contenant des graphes réels de taille large, étudié dans de nombreux domaines tel que les télécommunications réseaux [30–32].

Les deux premiers ensembles sont sous la forme d'un fichier texte. Chaque ligne de ces deux fichiers représente une instance d'un graphe, elle est sous la forme suivante n *triangleup* d, où n représente le nombre de sommet, g est une chaîne de 0 et 1 représentant le triangle supérieur de la matrice d'adjacence du graphe, et d représente le nombre de domination pour le graphe.
La chaîne de texte 3 110 1 par exemple, représente le graphe de la figure suivante :

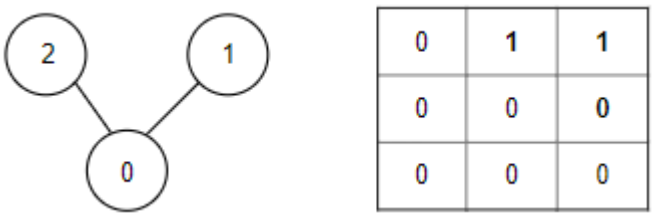

FIGURE V.1 – Un graphe à 3 sommets, la matrice d'adjacence équivalente.

Le troisième ensemble est utilisé pour comparer les algorithmes sur des graphes qui ont un nombre de domination plus élevé que le premier ensemble.
Le quatrième ensemble concerne des graphes réels de grande taille, cet ensemble contient des graphes provenant des réseaux sociaux *Google+* et *Pokec*. Les fichiers de cet ensemble sont sous le format *Dimacs* [33] et ont été utilisés dans beaucoup d'études, chaque fichier de cet ensemble représente un graphe pouvant atteindre 50000 sommets.

Les trois premiers ensembles sont utilisés pour tester les algorithmes exacts, alors que le quatrième est utilisé pour tester la qualité des heuristiques implémentées.

Système
Les tests ont été effectués sur une machine ayant les caractéristiques suivantes :

- AMD Ryzen 1700.
- 16 Go de mémoire vive.
- 256 Go SSD.
- Windows 10 64 bits.
- JDK 1.8.

Implémentation
Le langage Java a été choisi pour implémenter tous les algorithmes décrits dans ce mémoire. Le code source de l'implémentation se trouve dans un dépôt git à l'adresse suivante : `https://github.com/BoualiZakariae/MinDominatingSet`.

V.3 Algorithmes exacts

Dans cette section, nous allons présenter les résultats de comparaison des algorithmes exacts sur 2 différents types de graphes. Le premier type concerne les graphes généraux, le deuxième type concerne les graphes de degré maximum plus petit ou égal à 3.

V.3.1 Tous types de graphes

Pour rappel, les algorithmes exacts qu'on a présentés pour tous types de graphes sont les suivants :

— L'algorithme général (chapitre 2).
— L'algorithme trivial (chapitre 3).
— L'algorithme amélioré (chapitre 3).

Le tableau suivant, montre la distribution des graphes du premier ensemble selon l'ordre des graphes. Les graphes de cet ensemble peuvent atteindre l'ordre 10 :

N	4	5	6	7	8	9	10
Nombre de graphes	11	34	156	1044	12346	24668	12005168

La figure V.2 montre le temps total requis pour que les trois algorithmes calculent les ensembles dominants minimaux des graphes du premier ensemble :

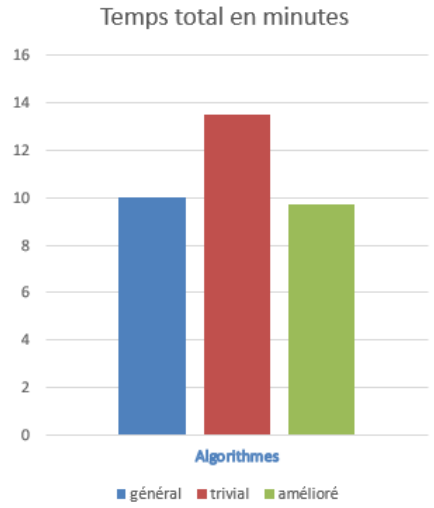

FIGURE V.2 – Comparaison des algorithmes exacts en fonction du temps.

Nous pouvons déjà remarquer que l'algorithme général montre des performances

presque équivalentes à l'algorithme amélioré, l'algorithme général prend 10 minutes pour traiter la totalité des graphes du premier ensemble, alors que l'algorithme amélioré prend 9 minutes et 40 secondes. L'algorithme trivial basé sur le Set Cover donne le pire résultat avec 13 minutes et 30 secondes.

Temps moyen d'exécution
La figure V.3 montre le temps moyen d'exécution pour chaque algorithme selon l'ordre des graphes. L'axe des abscisses représente l'ordre des graphes du premier ensemble, alors que l'axe des ordonnées représente le temps moyen en millisecondes. L'algorithme amélioré montre les meilleurs résultats que les autres algorithmes.

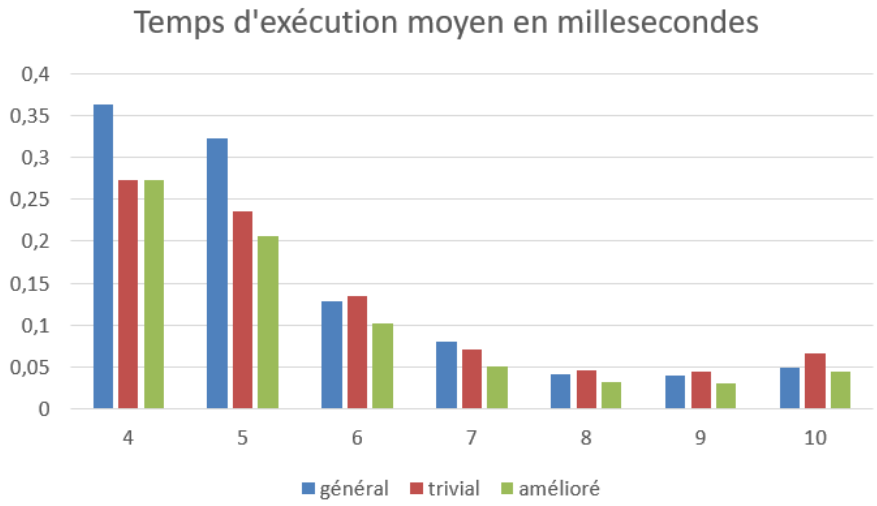

FIGURE V.3 – Comparaison des algorithmes exacts en fonction du temps d'exécution moyen.

D'après les résultats théoriques, l'algorithme amélioré devrait donner de meilleures performances. L'explication possible aux résultats des figures V.2, V.3 est la nature des graphes testés ; la taille des ensembles dominants minimums des graphes du premier ensemble n'est pas grande. Pour cette raison, le troisième ensemble des graphes est testé, cet ensemble contient 9 graphes qui ont un nombre de domination entre 8 et 10. Le tableau suivant affiche les résultats de la comparaison entre l'algorithme général et l'algorithme amélioré sur ce deuxième ensemble :

N	$\gamma(G)$	Identifiant	Algorithme général	Algorithme Amélioré
24	8	1391	862 ms	265 ms
26	8	968	2378 ms	1096 ms
30	8	1049	10468 ms	7639 ms
30	9	19959	22873 ms	8493 ms
32	8	1284	9381 ms	5762 ms
34	9	3345	3276833 ms	74135 ms
34	9	3347	2148155 ms	67658 ms
34	10	3357	- ms	77175 ms
34	10	3359	- ms	84343 ms

La première colonne représente le nombre de sommet du graphe, la deuxième et troisième colonne représentent respectivement la taille de l'ensemble dominant minimum et l'identifiant du graphe dans la base de données *House Of Graph*. La colonne 4 et 5 affichent le temps de traitement des graphes en millisecondes.
Cette fois-ci, et d'une façon claire, nous pouvons remarquer que l'algorithme amélioré affiche les meilleurs résultats. Nous remarquons aussi que l'algorithme amélioré peut trouver la solution optimale en un temps raisonnable pour certains graphes, où l'algorithme général échoue totalement (les deux graphes avec l'identifiant 3357, 3359).

V.3.2 Les graphes de degré maximum plus petit ou égal à trois

Dans le chapitre deux, nous avons vu en détails un algorithme pour les graphes de degré maximum plus petit ou égal à trois. Dans cette section, nous allons comparer les performances de cet algorithme aux algorithmes exacts pour tous types de graphe.
Pour faire cette analyse, les graphes du deuxième ensemble contiennent des graphes de degré maximum égal à trois. Ces graphes peuvent atteindre l'ordre 13. Le tableau suivant montre le nombre de graphes selon leur ordre :

N	5	6	7	8	9	10	11	12	13
Nombre de graphe	23	62	150	424	1165	3547	10946	36327	124137

Dans cette section, nous allons donc comparer les performances des quatre algorithmes exacts implémentés, c-à-d :

— L'algorithme général (chapitre 2).
— L'algorithme spécifique (chapitre 2).
— L'algorithme trivial (chapitre 3).
— L'algorithme amélioré (chapitre 3).

La figure V.4 montre les statistiques du deuxième test sur le deuxième ensemble :

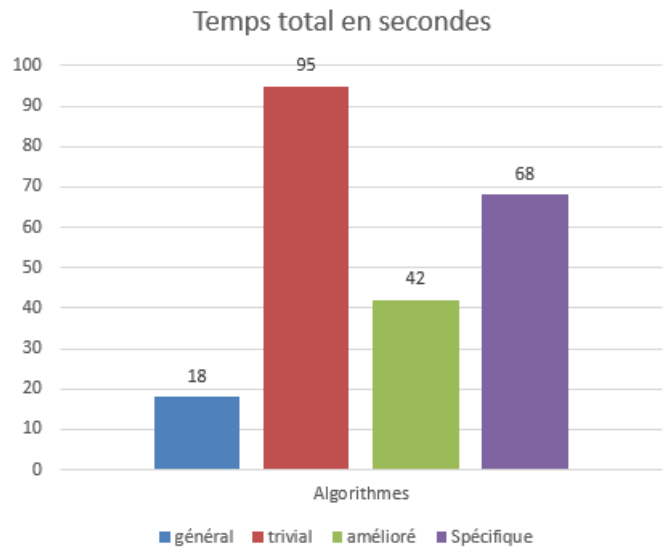

FIGURE V.4 – Comparaison des algorithmes exacts sur des graphes de degré maximum plus petit ou égal à trois.

Nous remarquons que l'algorithme général montre les meilleures performances par rapport aux autres algorithmes. À L'inverse, l'algorithme spécifique pour les graphes de degré maximum $\leqslant 3$ montre des performances faibles. La cause possible est que les graphes de l'ensemble testé n'ont pas de nombre de domination élevé. Pour cette raison, un ensemble de graphes ne contenant que des sommets de degré 3 est construit.

N	$\gamma(G)$	Algorithme général	Algorithme trivial	Algorithme Amélioré	Algorithme spécifique
22	6	150 ms	131 ms	76 ms	141 ms
32	9	55186 ms	12407 ms	5231 ms	15216 ms
38	11	- ms	289534 ms	90230 ms	527563 ms

Nous remarquons cette fois, que l'algorithme spécifique montre des performances meilleures que l'algorithme général. Cependant, c'est l'algorithme amélioré basé sue le Set Cover qui donne les meilleures performances. L'algorithme général échoue totalement pour le graphe à 38 sommets.

Après nos tests sur plusieurs types de graphe, nous déduisons que l'algorithme général montre de bonnes performances lorsque la taille de l'ensemble dominant n'est pas grande. Cependant lorsque la taille de l'ensemble dominant minimum est grande, c'est l'algorithme amélioré qui montre les meilleures performances.

V.4 Approches heuristiques

Dans cette section, nous allons présenter les résultats issus des tests des heuristiques implémentées. L'objectif est d'observer les performances des heuristiques sur des graphes de tailles différentes.

V.4.1 Premier test

Ce test concerne les graphes de taille large, l'ordre de cet ensemble varie de 11 à 864 sommets. Ces graphes ont été utilisés dans beaucoup d'études, spécialement dans le problème de coloration de graphe.
Ce premier test est exécuté sur un ensemble contenant 57 graphes. Les deux tableaux suivants présentent les résultats de ce test :

Graphe	N	Greedy	GreedyRev	GreedyRand	GeneticAlgo1	GeneticAlg2
anna	130	12	12	18	12	12
david	87	2	2	4	2	2
queen5_5	25	3	3	5	3	3
queen6_6	36	4	4	6	3	3
queen7_7	49	5	5	6	4	4
queen8_12	96	7	7	9	6	6
queen8_8	64	5	6	7	5	5
queen9_9	81	5	7	7	5	5
queen10_10	100	6	8	11	5	6
queen11_11	121	7	7	12	5	6
queen12_12	144	7	10	12	7	7
queen13_13	169	7	11	13	7	7
queen14_14	196	8	12	14	8	8
queen15_15	225	9	14	16	9	9
queen16_16	256	10	14	16	9	9
myciel3	11	3	3	5	3	3
myciel4	23	4	4	6	4	4
myciel5	47	5	5	10	5	5
myciel6	95	6	6	14	6	6
myciel7	191	7	7	12	8	8
miles250	128	23	23	30	22	22
miles500	128	13	10	19	9	9
miles750	128	10	7	9	6	6
miles1000	128	5	4	6	4	4
miles1500	128	2	2	4	2	2
mulsol.i.1	197	2	2	4	2	2
mulsol.i.2	188	2	2	3	2	2
mulsol.i.3	184	2	2	4	2	2
mulsol.i.4	185	2	2	2	2	2
mulsol.i.5	186	2	2	3	2	2

Graphe	N	Greedy	GreedyRev	GreedyRand	GeneticAlgo	GeneticAlg2
huck	74	9	9	11	9	9
games120	120	14	15	23	13	13
zeroin.i.1	211	2	2	4	2	2
zeroin.i.2	211	2	2	4	2	2
zeroin.i.3	206	2	2	3	2	2
school1	385	17	18	24	15	15
school1_nsh	352	15	17	28	15	15
fpsol2.i.1	496	2	2	4	2	2
fpsol2.i.2	451	2	2	4	2	2
fpsol2.i.3	425	2	2	5	2	2
homer	561	92	92	127	91	91
jean	80	11	11	19	10	10
le450_15a	450	29	33	54	24	23
le450_15b	450	31	33	52	27	27
le450_15c	450	12	17	26	12	12
le450_15d	450	13	17	28	12	13
le450_25a	450	45	53	67	37	37
le450_25b	450	37	42	57	31	30
le450_25c	450	15	19	30	13	14
le450_25d	450	16	19	28	14	14
le450_5a	450	31	38	59	33	30
le450_5b	450	30	41	55	30	32
le450_5c	450	20	27	35	24	20
le450_5d	450	20	27	36	23	21
inithx.i.1	864	2	2	3	2	2
inithx.i.2	645	2	2	4	2	2
inithx.i.3	621	2	2	4	2	2

Les deux premières colonnes représentent le nom du graphe et son nombre de sommets. Les colonnes restantes, représentent le nombre de domination trouvé par chaque algorithme. *GeneticAlgo1* représente la première version de l'algorithme génétique, c-à-d l'algorithme où un seul individu intègre la population à chaque itération, *geneticAlgo2* représente la deuxième version de l'algorithme génétique.

Pour les 30 graphes du premier tableau, nous remarquons que les deux versions de l'algorithme génétique donnent de très bons résultats en trouvant la meilleure solution 29 fois pour *geneticAlgo1*, et 27 fois pour *geneticAlgo2*. *Greedy* trouve la meilleure solution 20 fois et *greedy reverse* 15 fois. Quant à *greedy random*, il donne le pire score puisqu'il ne trouve la meilleure solution qu'une seule fois.

Dans les résultats du deuxième tableau, les mêmes remarques sont observées. Cette fois-ci, sur les 27 graphes, *geneticAlgo2* montre les meilleures performances en trouvant la meilleure solution 23 fois, *geneticAlgo1* 22 fois, l'algorithme *greedy* trouve la meilleure solution 17 fois, *greedy reverse* seulement 11 fois. Tandis que, *greedy random* ne la trouve en aucune fois.

Les figures suivantes montrent d'une façon plus claire cette comparaison :

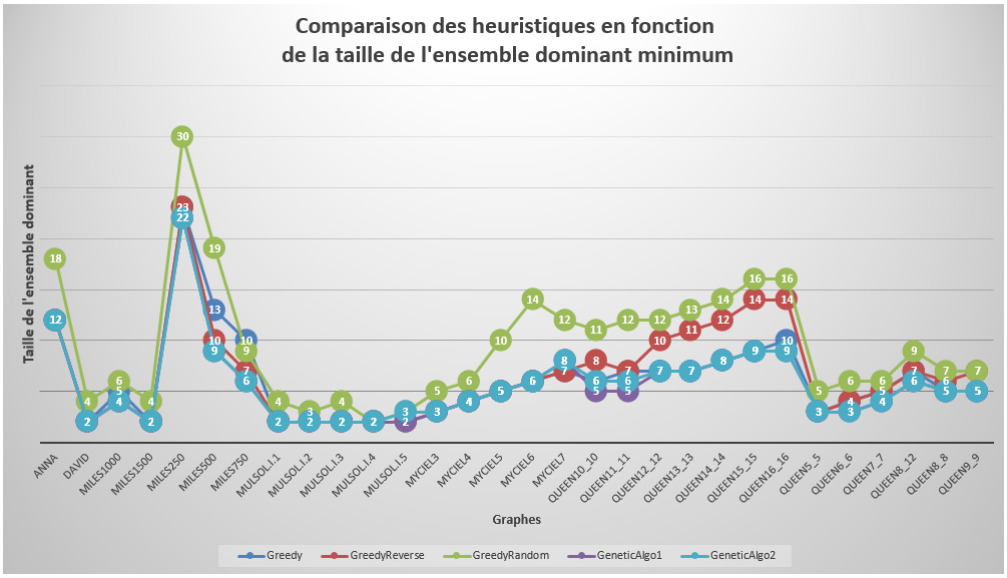

FIGURE V.5 – Comparaison-1 des heuristiques en fonction de la taille du MDS.

Dans la figure V.5, *greedy random*, n'affiche pas des résultats aussi proches de la solution optimale que les autres algorithmes. Cette remarque est aussi observée pour l'algorithme *greedy reverse* pour certains graphes.

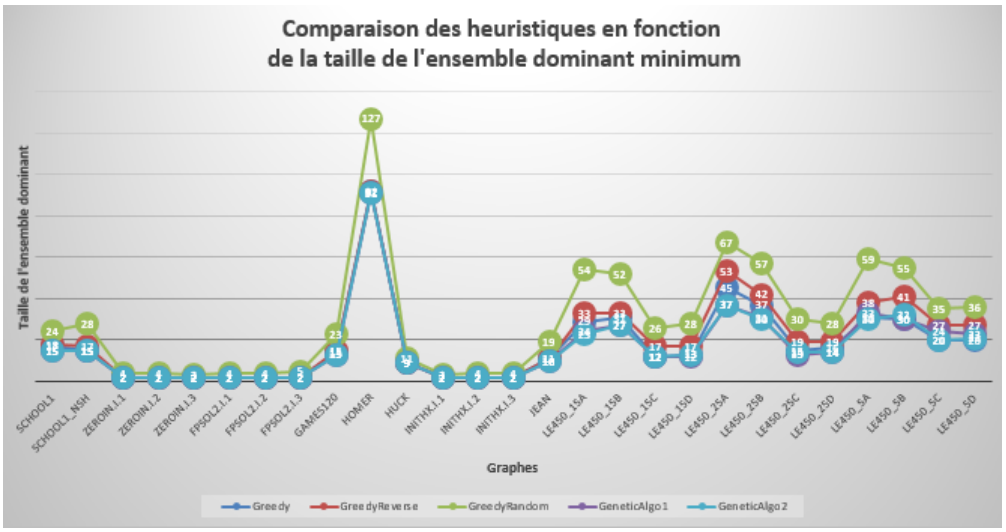

FIGURE V.6 – Comparaison-2 des heuristiques en fonction de la taille du MDS.

Dans cette deuxième comparaison, et globalement, *geneticAlgo1* et *geneticAlgo2* affichent des résultats équivalents sur la plupart des instances. *Greedy random* affiche les mêmes résultats que la première comparaison.

V.4.2 Deuxième test

Les graphes utilisés dans ce test, sont des graphes réels issus des réseaux sociaux *Google+* et *Pokec*, ces graphes sont très larges et peuvent atteindre 50000 sommets.
Le tableau suivant, présente les résultats de comparaison des 5 algorithmes sur les graphes de *Google+* :

Algorithme	gplus_200	gplus_500	gplus_2000	gplus_10000	gplus_20000	gplus_50000
Greedy	19(0%)	42(0%)	177(4.1%)	896(4%)	1809(5.4%)	4844(6%)
Greedy Reverse	19 (0%)	42 (0%)	171 (0.5%)	876(1.7%)	1760(2.5%)	4751(4%)
Greedy Random	37 (95%)	125 (197%)	419(146%)	2065(139%)	4064(136%)	10407(127%)
GeneticAlgo1	19 (0%)	42(0%)	198 (16%)	2350(172%)	5633 (228%)	14460 (216%)
GeneticAlgo2	19 (0%)	42(0%)	179(5.2%)	1723(100%)	5308(209%)	-(-%)
Solution optimale	19	42	170	861	$\geqslant$ 1716	$\geqslant$ 4566

La valeur entre les parenthèses représente l'écart en pourcentage entre la valeur optimale et la valeur trouvée par l'algorithme.
Nous remarquons que l'algorithme *greedy reverse* donne les meilleurs résultats pour tous les graphes de cet ensemble. L'algorithme génétique trouve la solution optimale pour les graphes de taille 200 et 500, mais échoue pour les graphes qui dépassent 2000 sommets.

Le tableau suivant, présente les résultats de comparaison des 5 algorithmes sur les graphes de *Pokec+* :

Algorithme	pokec_500	pokec_2000	pokec_10000	pokec_20000	pokec_50000
Greedy	16(0%)	75 (0%)	413(0%)	926(0.5%)	2770(2.3%)
Greedy Reverse	16(0%)	75(0%)	423(2.4%)	946(2.7%)	2818(4.1%)
Greedy Random	38 (137%)	271(261%)	1481 (258%)	3020(227%)	8235(204%)
GeneticAlgo1	16(0%)	75(0%)	1463(254%)	4423(380%)	12986 (379%)
GeneticAlgo2	16(0%)	75(0%)	672(62.7%)	4208(356%)	-(-%)
Solution optimale	16	75	413	921	$\geqslant$ 2706

Cette fois-ci, c'est l'algorithme *greedy* qui donne les meilleurs résultats. L'algorithme génétique échoue encore une fois, mais pour les graphes qui dépassent 10000 sommets.

D'après ces derniers tests, nous remarquons que les algorithmes *greedy* peuvent donner les meilleurs résultats pour des graphes très larges. Cependant, ce n'est pas le cas pour l'algorithme génétique.

Chapitre VI

Conclusion

Dans ce mémoire, nous nous sommes intéressés aux méthodes exactes et aux approches heuristiques pour résoudre le problème de l'ensemble dominant minimum.

Dans le premier chapitre, et après avoir rappelé certaines notions fondamentales de la théorie des graphes, nous avons introduit le problème de l'ensemble dominant minimum et certaines de ses applications.

Dans le deuxième chapitre, nous avons présenté certaines méthodes exactes pour résoudre le problème de l'ensemble dominant, à savoir, l'algorithme exact général pour tous types de graphes et l'algorithme spécial pour les graphes de degré maximum $\leqslant 3$. Dans le troisième chapitre, nous avons présenté un algorithme exact basé sur le problème de la couverture par ensembles. Cet algorithme réduit le problème de l'ensemble dominant en un problème de couverture par ensembles.

Dans le quatrième chapitre, nous avons expliqué différentes heuristiques pour résoudre le problème de l'ensemble dominant. La première partie concernait les heuristiques de type *greedy*, alors que la deuxième partie concernait une méta-heuristique de type population. Nous avons implémenté trois heuristiques et un algorithme génétique.

Dans le cinquième chapitre, nous avons fait une analyse empirique des différents algorithmes, nous avons comparé les algorithmes exacts en fonction du temps, et les approches heuristiques en fonction de la qualité des résultats. Plusieurs types de graphes ont été sélectionnés pour faire cette analyse. Les résultats qu'on a obtenus confirment les résultats théoriques présentés dans ce mémoire, à savoir que l'algorithme général ne se comporte pas bien lorsque les graphes ont un nombre de domination élevé. L'algorithme spécial pour les graphes de degré maximum $\leqslant 3$ montre de meilleures performances par rapport à l'algorithme général lorsqu'il s'agit de ce type de graphes.

Les tests effectués dans le cinquième chapitre, montrent aussi que l'algorithme amélioré basé sur le *Set Cover* donne les meilleurs résultats lorsqu'il s'agit de graphes avec un nombre de domination élevé. L'algorithme amélioré donne une solution optimale pour certains graphes où l'algorithme général échoue totalement.

Concernant la comparaison des heuristiques, nous avons constaté que l'algorithme génétique avec ces deux versions montre les meilleures performances pour les graphes de taille $\leqslant 2000$. Cependant, pour les graphes qui dépassent cette taille, c'est les heuristiques de type *greedy* qui montrent les meilleures performances.

En résumé, nous pouvons affirmer que l'algorithme amélioré basé sur le problème de la couverture par ensembles est la meilleure approche exacte pour les graphes de petite tailles. Cependant, lorsque la taille des instances est grande et telle qu'elle ne dépasse pas 2000, les deux versions de l'algorithme génétique sont les plus recommandées. En outre, les algorithmes *greedy* sont les plus recommandés pour les graphes très larges qui dépassent 2000 sommets.

Bibliographie

[1] Wikipedia. Minimum dominating set. https://fr.wikipedia.org/wiki/Ensemble_dominant, 2010 (accessed October 3, 2018).

[2] Stephen Hedetniemi, Peter Slater, and Teresa W Haynes. *Fundamentals of domination in graphs*. CRC press, 2013.

[3] Fedor V Fomin, Dieter Kratsch, and Gerhard J Woeginger. Exact (exponential) algorithms for the dominating set problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 245–256. Springer, 2004.

[4] Johan MM Van Rooij and Hans L Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17) :2147–2164, 2011.

[5] Wikipedia. Set cover problem. https://en.wikipedia.org/wiki/Set_cover_problem, 2010 (accessed October 3, 2018).

[6] Laura A Sanchis. Experimental analysis of heuristic algorithms for the dominating set problem. *Algorithmica*, 33(1) :3–18, 2002.

[7] Reinhard Diestel. *Graph theory*. Springer Publishing Company, Incorporated, 2018.

[8] Chao Shen and Tao Li. Multi-document summarization via the minimum dominating set. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 984–992. Association for Computational Linguistics, 2010.

[9] Alen Doko, Maja Štula, and Ljiljana Šerić. Using tf-isf with local context to generate an owl document representation for sentence retrieval.

[10] Feng Wang, Hongwei Du, Erika Camacho, Kuai Xu, Wonjun Lee, Yan Shi, and Shan Shan. On positive influence dominating sets in social networks. *Theoretical Computer Science*, 412(3) :265–269, 2011.

[11] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 7–14. ACM, 1999.

[12] Bevan Das and Vaduvur Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Communications, 1997. ICC'97 Montreal,*

Towards the Knowledge Millennium. 1997 IEEE International Conference on, volume 1, pages 376–380. IEEE, 1997.

[13] www.journaldunet.com. Nombre d'utilisateurs de facebook dans le monde. https://www.journaldunet.com/ebusiness/le-net/1125265-nombre-d-utilisateurs-de-facebook-dans-le-monde/, 2018 (accessed November 26, 2018).

[14] Wikipedia. Vertex cover problem. https://en.wikipedia.org/wiki/Vertex_cover, 2010 (accessed October 8, 2018).

[15] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[16] Dave Mount. Np-completeness proof : Dominating set. http://www.cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect21-np-clique-vc-ds.pdf//, 2018 (accessed November 29, 2018).

[17] Bruce Reed. Paths, stars and the number three. *Combinatorics, Probability and Computing*, 5(3) :277–295, 1996.

[18] Michael R. Garey and David S. Johnson. *Computers and Intractability ; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[19] Wikipedia. Edge cover problem. https://en.wikipedia.org/wiki/Edge_cover, 2010 (accessed December 8, 2018).

[20] Jack Edmonds. Paths, trees and flowers. *CANADIAN JOURNAL OF MATHEMATICS*, pages 449–467, 1965.

[21] Wikipedia. Matching. https://en.wikipedia.org/wiki/Matching_(graph_theory), 2010 (accessed December 8, 2018).

[22] Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM)*, 56(5) :25, 2009.

[23] JH Holland and D Goldberg. Genetic algorithms in search, optimization and machine learning. *Massachusetts : Addison-Wesley*, 1989.

[24] David Roger Oldroyd. Darwinian impacts : An introduction to the darwinian revolution. 1982.

[25] Anupama Potluri and Alok Singh. Two hybrid meta-heuristic approaches for minimum dominating set problem. pages 97–104, 12 2011.

[26] J.E Beasley and P.C Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2) :392 – 404, 1996.

[27] Hadrien Mélot. Facet defining inequalities among graph invariants : The system graphedron. *Discrete Applied Mathematics*, 156(10) :1875–1891, 2008.

[28] Brendan D McKay and Adolfo Piperno. Nauty and traces users guide (version 2.5). *Computer Science Department, Australian National University, Canberra, Australia*, 2013.

[29] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. House of graphs : a database of interesting graphs. *Discrete Applied Mathematics*, 161(1-2) :311–314, 2013.

[30] Lubos Takac and Michal Zabovsky. Data analysis in public social networks. In *International Scientific Conference and International Workshop Present Day Trends of Innovations*, volume 1, 2012.

[31] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability : second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

[32] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3) :036104, 2006.

[33] Rutgers University DIMACS Project. Coloring problems dimacs graph format. `http://prolland.free.fr/works/research/dsat/dimacs.html`, 1993 (accessed January 15, 2019).