Analyse d'Algorithmes et Génération Aléatoire (5I550)

M2 STL 2016/2017

Ensemble Dominant Connexe algorithmes présentées dans [Li, Thai, Wang, Yi ,Wan and Du][1]

Auteurs
Chafik Nouira
Morane Gruenpeter

Responsable :
Bình Minh Bùi Xuân

Version 0.1 du
30 octobre 2016

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Résumé

Le problème de l'Ensemble Dominant Connexe Minimal est un problème NP-difficile. Pour ce problème les auteurs de l'article "Wireless Communications and Mobile Computing" [1] proposent un nouvel algorithme glouton appelé **S-MIS**. Nous allons implémenter et analyser cet algorithme afin d'en déduire son efficacité. En parallèle nous allons analyser une version naïve pour pouvoir comparer les différentes approches. Par ailleurs, nous allons présenter un autre algorithme que nous avons nommé, **"The magic MIS"**. Ce dernier est moins efficace que le **S-MIS**, or il retourne une solution bien meilleur pour le probléme.

1.2 Problème de l'Ensemble Dominant Connexe

Étant donné un graphe géométrique G représenté par un ensemble de nœuds V et un ensemble d'arêtes E placés dans le plan cartésien, et sachant que la longueur d'une arête liant deux nœuds ne peut dépasser un seuil donné, l'un des problèmes algorithmiques NP-difficiles les plus connus est le problème de l'Ensemble Dominant Connexe servant à dominer tous les nœuds du graphe G. Plus concrètement, il s'agit de trouver un ensemble minimal de sommets $S \subset V$ tel que tout sommet $v \in V$ du graphe est soit un élément de S, soit un voisin d'un élément de S, et tel que le sous-graphe $G[S]$ induit par les sommets de S est connexe.

Il existe plusieurs domaines dans lesquels la résolution du problème de l'Ensemble Dominant Connexe est utile. En particulier dans le domaine des réseaux sans fil, où une gestion centralisée est absente. l'Ensemble Dominant Connexe s'est avéré être le meilleur candidat pour assurer et faciliter la diffusion et multidiffusion de données dans un réseau sans fil, et les nœuds faisant partie de l'ensemble sont parfait pour conserver les informations

de routage.

Dans ce rapport, nous commencerons par présenter une solution naïve au problème de l'Ensemble Dominant Connexe, nous poursuivront ensuite par l'explication de l'algorithme fourni par **Li et al.**[1], et nous finiront par présenter notre algorithme inspiré de celui de **Guha and Khuller**[2] qui combine les avantages de l'algorithme naïf et de l'algorithme de Li et al. Bien entendu, les 3 algorithmes présentés seront comparés en termes de complexité, de temps d'exécution et de qualité du résultat.

1.3 Définitions et notations

Dans les sections qui vont suivre, nous désignons par **G** le graphe à traiter, par **G** l'ensemble de nœuds du graphe et par **E** l'ensemble d'arêtes. Cela peut être abrégé comme suit : $G = (V, E)$. Dans un souci de clarté, le terme «Ensemble Dominant » sera remplacé par DS ('Dominating Set' en anglais) et le terme «Ensemble Dominant Connexe» par CDS (Connected Dominating Set). D'autres termes seront également représentés comme suit :

- UDG (Unit Disk Graph) → Graphe de disques.
- MCDS (Minimum-sized Connected Dominating Set) → Ensemble Dominant Connexe Minimal.
- MIS (Maximal Independent Set) → Ensemble Stable Maximum.
- PR (Performance Ratio)→ Ration de performance.
- ID → Identifiant
- ST-MSN (Steiner Tree with Minimum Steiner Nodes)→ Arbre de Steiner avec un minimum de nœuds de Steiner.

Chapitre 2

Etude expérimentale

2.1 Etude théorique

2.1.1 Algorithme Naïf - Ensemble Stable Glouton & Steiner

Généralement, pour résoudre un problème, il faut commencer par trouver un algorithme naïf fournissant la solution attendue. Dans le cas du problème du **CDS**, l'algorithme naïf choisi pour ce rapport se compose de 2 phases :

1. Trouver un **Ensemble Stable** : Cela consiste à calculer un sous-ensemble de sommets $S \subset V$ tel que le sous graphe G [S] induit par les sommets de S est vide (i.e $G[S]$ contient aucune arête). Autrement dit, tout sommet $u \in S$ n'a aucun voisin dans S.
   Un Ensemble Stable est toujours un DS.

2. Une fois l'Ensemble Stable obtenu, nous utilisons l'algorithme de Steiner afin de construire un arbre de Steiner passant par tous les sommets de S et qui contient le plus petit nombre de sommets possible. Bien entendu, l'arbre de Steiner sera un sous-graphe de G.
   Le problème de l'arbre de Steiner étant un problème NP-Difficile, ce dernier fournira une solution approchée.

Nous supposons que la fonction qui calcule la liste des voisins d'un sommet existe déjà et que la fonction calculant l'arbre de Steiner est fournie.

l'algorithme 1 présente la première étape de la solution naïve, celui-ci retourne un Ensemble Stable. l'algorithme 2 calcule l'arbre de Steiner sur l'Ensemble Stable retourné par l'algorithme 1.

Algorithm 1 Algorithme INDEPENDENT SET

```
procedure IndependentSet(G)                          ▷ Un graphe de points
    Initialize domSet
    while isEmpty(V) = false do
        max ← 0
        pmax ← V[0]                  ▷ Ici nous récupérons le premier noeud dans v
        for p in V do
            if size(voisins(v)) > max then
                max ← size(voisins(v))
                pmax ← p
            domSet ← domset ∪ pmax
            V ← V − (voisins(v) ∪ pmax)
    return domSet
```

Algorithm 2 Algorithme Naïf - Ensemble Stable Glouton & Steiner

```
procedure CDSNaif(G) returns CDS(G)
    DS ← MIS(G)
    CDS ←Steiner(DS,G)
    return CDS
```

2.1.2 Algorithme Li et al.

Plusieurs solutions algorithmiques ont été proposées au problème du CDS minimal, parmi ces solutions nous pouvons citer l'algorithme de Guha et Khuller[2] et celui de Wu et Li[3] qui a une complexité en $O(n)$ mais qui retourne un CDS de grande taille. L'algorithme que nous présentons dans cette section est celui de Li et al., il se compose de 2 phases principales, la première sert à calculer un MIS, et la deuxième permet de rajouter au MIS les nœuds de Steiner afin de connecter les nœuds entre eux et construire un CDS.

Construction MIS

Un MIS est un Ensemble Stable de taille maximale.
Afin de construire un MIS valide, deux propriétés fondamentales doivent être satisfaites :

1. Dans n'importe quel **UDG**, la taille de chaque MIS est majorée par $(3.8 * opt) + 1.2$ où *opt* est la taille du **MCDS** dans le **UDG** concerné.
2. N'importe quelle paire de sous-ensembles complémentaires du MIS a une distance exacte de 2 voisins.

L'algorithme procède comme suit :

1. Colorer tous les nœuds du graphe en blanc et mettre leur état à 'inactif'.

2. Récupération d'un nœud aléatoire (qu'on appellera m) du graphe.
3. Colorer m en noir et mettre son état à 'inactif' puis le rajouter au MIS.
4. Colorer en gris tous les voisins de m.
5. Mettre à 'actif' l'état de tous les voisins des voisins de m (Sauf m).
6. Tant qu'il existe des nœuds blancs dans le graphe faire
   - Récupérer dans n le best actif (nœud actif ayant un maximum de voisins blancs).
   - Colorer n en noir, mettre son état à 'inactif' et le rajouter au MIS.
   - Colorer en noir tous les voisins de n.
   - Mettre à actif tous les voisins des voisins de n. (Sauf les nœuds noirs)
7. Retourner le MIS.

Ci-dessous le pseudocode de l'algorithme proposé par Guha et Khuller[2] en tant que première phase afin de calculer un MIS valide avant de l'utiliser pour calculer le **CDS**.

Algorithm 3 Algorithme VALID - MIS

```
procedure VALIDMIS(G)
    Initialize res
    for v in V do
        setColor(v, WHITE)
        setActif(v, false)
    m ← getRandom(v)
    setColor(m, BLACK)
    setActif(m, false)
    res.add(m)
    for v in Voisins(m) do
        setColor(v, GREY)
    for v in Voisins(m) do
        for b in Voisins(m) do
            setActif(b, true)
    while exists v in V and color(v) == WHITE do
        n ← bestActif(v)                ▷ sommet actif avec max de voisins blancs
        setColor(n, BLACK)
        setActif(n, false)
        res.add(n)
        for p in voisins(n) do
            setColor(p, GREY)
            setActif(voisins(p), true)
    Return res
```

On peut également prendre en entrée un Ensemble Stable au lieu d'un MIS en utilisant l'algorithme 1 dans la section précédente.

The S-MIS algorithm

Dans cette section nous présentons la deuxième phase de l'algorithme proposé par **Li et al** [1]., appelé l'algorithme de **S-MIS**. Ce dernier prend le **MIS** en entrée et produit un **CDS** borné par $(4.8 + 1n5)opt + 1.2$ où opt est la taille du **MCDS**. Il se déroule comme suit :

1. Colorer en noir tous les nœuds de **MIS** et leurs affecter des identifiants uniques.
2. Colorer en gris tous les autres nœuds du graphe.
3. Pour i de 5 à 2
4. Tant qu'il existe des nœuds gris ayant au moins i voisins noirs (avec au moins i identifiants distincts)
   • Colorer le nœud gris en bleu.
5. Rajouter tous les nœuds bleus au **MIS**
6. Retourner le **MIS**

Afin de rendre la lecture du pseudo-code plus compréhensible, nous considérons existantes les fonctions utilitaires suivantes :
— Les fonctions servants à modifier la couleur et l'identifiant d'un nœud.
— La fonction qui retourne les nœuds gris ayant i voisins noirs avec i identifiants distincts.
— La fonction qui retourne les voisins noirs d'un nœud gris donné.
— La fonction qui retourne les nœuds avec l'id minimum dans une liste donnée.
— La fonction qui retourne les nœuds noirs ayant le même identifiant qu'un nœud noir donné.
L'algorithme S-MIS décrit ci-dessus peut donc être formulé comme suit :

Algorithm 4 Algorithme SMIS

```
procedure SMIS(MIS, G) returns MCDS(MIS,G)
    Initialize blueNodes
    for p in V do
        setColor(p, GREY)
    i ← 0
    for p in MIS do
        setColor(p, BLACK)
        setID(p, i)
        i ← i + 1
    for i = 5 to 2 do  ▷ Ici nous récupérons la liste des noeuds gris ayant au moins i voisins noirs avec au moins i identifiants distincts
        greyNodes := listGreyNodesWithIBlackNeighbors(V, MIS, i)
        for g in greyNodes do
            blackNeighbors ← getBlackNeighbors(g)
            if allDomSameID(blackNeighbors) then
                continue
            setColor(g, BLUE)
            blueNodes ← blueNodes ∪ g
            idMin ← minIdBlackNeighbor(blackNeighbors)
            Initialize list
            for b in blackNeighbors do  ▷ Ici nous ajoutons les noeuds noirs avec le même id
                list ∪ blackNodesWithSameID(b)
                setID(b, idMin)
            for x in list do
                setID(x, idMin)
    return MIS ∪ blueNodes
```

2.1.3 Bonus - The Magic MIS

Nous présentons ici notre algorithme que nous avons décidé d'appeler **The Magic MIS**, mais attention, même s'il y a le mot **MIS** dans son nom, c'est un **MCDS** qui est retourné, d'où le mot **Magic**. Une autre explication du nom est que c'est un algorithme inspiré de celui de **Guha et Khuller**[2]. En gros, il se base entièrement sur un algorithme de construction d'un MIS afin de retourner un MCDS. Magic ! Celui-ci retourne donc un **CDS** beaucoup plus petit en taille qu'un CDS calculé par l'algorithme naïf ou l'algorithme **S-MIS**.

**Voir annexes pour les scores fournis par chaque algorithme sur un même graphe de 1000*

points.

Autre détail **très important** : La taille de la solution fournie par The Magic MIS dépend entièrement du nœud de départ, en gros le nœud pris aléatoirement dans **V** et à partir duquel nous allons construire notre CDS. Il suffit donc de tester avec tous les nœuds de **V** et de récupérer le meilleur score.
Par exemple, en testant The Magic MIS avec le graphe fourni dans le canevas du **TME3 de AAGA**, et en essayant avec tous les nœuds comme nœud de départ, les tailles des solutions fournies étaient entre 124 et 136. L'algorithme naïf et l'algorithme de Li et al. ont trouvé respectivement 154 et 165.

De plus, en termes de temps d'exécution, The Magic MIS n'est pas très loin de l'algorithme S-MIS. Par exemple, sur un graphe contenant 1000 points, l'algorithme naïf trouve une solution en 14 secondes, le S-MIS en 0.1 secondes et The Magic MIS en 1 seconde.

The Magic MIS combine donc les avantages de l'algorithme naïf et du S-MIS tout en fournissant une solution bien meilleure. Et, cerise sur le gâteau, il n'a pas besoin d'un Ensemble Stable de départ !
Ci-dessous le pseudocode décrivant l'algorithme MIS Magique, vous pourrez constater la ressemblance frappante avec l'algorithme $VALID-MIS(G)$ présenté dans la section 2.1.2. La seule différence c'est que celui-ci, dans la boucle while, au lieu de prendre comme bestActif un nœud BLANC actif et ayant un maximum de voisins blancs, prend obligatoirement un nœud GRIS avec un maximum de voisins blancs, ce qui construit directement la solution.

Algorithm 5 Algorithme MAGIC - MIS

```
procedure MagicMis(G) returns MCDS(G)
    Initialize res
    for v in V do
        setColor(v, WHITE)
    m ← getRandom(V)
    res.add(m)
    for v in voisins(m) do
        setColor(v, GREY)
    for v in voisins(m) do
        for b in voisins(v) do
            setActif(b)
    while exists v in V and color(v) == WHITE do
        n ← bestActif(V)            ▷ nœud GRIS actif avec max de voisins blancs
        res.add(n)
        for p in voisins(n) do
            setColor(p, GREY)
            setActif(voisins(p), true)
    return MIS ∪ blueNodes
```

2.2 Complexités des algorithmes

Dans cette section, nous allons nous intéresser à la complexité de chacun des 3 algorithmes présentés.

2.2.1 Algorithme Naïf - Ensemble Stable Glouton & Steiner

Pour un graphe contenant un seul nœud, l'algorithme naïf a un temps d'exécution constant, puisqu'il retourne directement la solution. Si le graphe est de taille importante, l'algorithme naïf commence par calculer un ensemble stable. Pour se faire, 2 approches sont possibles :

1. En utilisant l'approche gloutonne, cette dernière parcourt la liste des nœuds en supprimant les voisins du point courant, ce dernier est également supprimé après être rajouté à l'ensemble stable. Cette action est répétée jusqu'à ce qu'on ait plus de nœuds à traiter, à la fin l'ensemble stable est retourné. L'approche gloutonne se fait en 2 boucles, elle a une complexité en $O(n^2)$.
2. En utilisant l'approche implémentée pour le projet du TME 2 de AAGA, cette

dernière consiste à améliorer un ensemble stable donné par la première approche. L'amélioration consiste à chercher les paires de nœuds remplaçables par un seul en gardant un DS valide, elle utilise 3 boucles imbriquées, elle a donc une complexité en $O(n^3)$.

Une fois l'ensemble stable calculé, il suffit de le prendre en entrée dans la fonction Steiner qui consiste à calculer un Arbre de Steiner passant par tous les nœuds de l'ensemble. L'algorithme de Steiner commence par construire un graphe complet des Plus Courts Chemins, et se base dessus pour calculer un Arbre Couvrant Minimum par la suite. Il a une complexité en $O(n^3)$. L'approche naïve a donc une complexité en $O(n^3)$ + $O(n^3)$, soit $O(n^3)$.

2.2.2 Algorithme Li et al.

L'approche proposée par Li et al. se compose aussi de 2 phases :

1. La première phase consiste à calculer un MIS en utilisant l'approche représentée par le pseudocode de VALIDMIS(G) de la section 2.1.2.1. Comme nous pouvons le constater, cette approche contient 2 boucles imbriquées parcourant l'ensemble des nœuds, elle a donc une complexité en $O(n^2)$
2. La deuxième phase consiste à utiliser l'algorithme de S-MIS présenté dans la section 2.1.2.2, ce dernier prend en entrée le MIS et trouve dans V un minimum de nœuds (qui ne se trouvent pas dans le MIS) qui auront pour rôle de connecter les nœuds du MIS afin d'avoir un CDS. Le résultat final sera le MIS + les nœuds choisis pour connecter les nœuds MIS (Nœuds bleus dans le pseudocode). Cet algorithme a une complexité meilleur cas en $O(nlogn)$, puisqu'il parcourt presque jamais tous les points dans deux boucles imbriquées, cependant, sa complexité pire cas est en $O(n^2)$ pour certains cas précis.

2.2.3 Bonus - The Magic MIS

Comme indiqué dans la section 2.1.3, l'algorithme The Magic MIS se base entièrement sur l'algorithme de construction de MIS, à quelques différences près. Il a donc une complexité en $O(n^2)$.

2.3 Implémentation

Le langage d'implémentation qui a été choisi pour cette étude expérimentale est Java. Nous avons opté aussi pour une classe MyPoint représentant un point avec une instance de Point, une couleur, un identifiant et un état (Actif ou Inactif). Ce choix s'explique par

le fait que la classe Point de Java a seulement deux attributs de type int correspondant à ses coordonnées x et y.

Le fichier canevas du TME 3 de AAGA et les sources du projet Arbre de Steiner de CPA ont été d'une grande utilité, puisqu'ils nous ont servi de base pour effectuer les différents tests, réutiliser certaines fonctions déjà implémentées et surtout avoir un aspect visuel des solutions fournies par les différents algorithmes.

Estimant que l'algorithme naïf n'est pas très intéressant d'un point de vue expérimental, nous présentons ci-dessous l'implémentation de l'algorithme de S-MIS puis l'algorithme Magic MIS.

• Implémentation de l'Algorithme de S-MIS

```
public ArrayList<MyPoint> SMIS(ArrayList<MyPoint> points, ArrayList<MyPoint>
    MIS) {
        ArrayList<MyPoint> res = (ArrayList<MyPoint>) MIS.clone();
        ArrayList<MyPoint> neighbourBlack = new ArrayList<MyPoint>();
        ArrayList<MyPoint> listGreyNodeIDom = new ArrayList<MyPoint>();
        int i = 0;
        for(MyPoint p : points) {
                p.setCouleur(Color.GREY);
        }
        for(MyPoint p : MIS) {
                p.setCouleur(Color.BLACK);
                p.setId(i);
                i++;
        }

        for(i = 5; i > 1; i--) {
                listGreyNodeIDom = listGreyNodesContainsIDom(points, MIS, i)
                    ;
                for(MyPoint p : listGreyNodeIDom) {
                        neighbourBlack = getListDomGreyNode(p, MIS);
                        if(idEqualAllBlackNode(neighbourBlack)) continue;
                        p.setCouleur(Color.BLUE);
                        int idMin = idMinNeighbourBlack(neighbourBlack);
                        ArrayList<ArrayList<MyPoint>> list = new ArrayList<
                            ArrayList<MyPoint>>();
                        for(MyPoint tmp : neighbourBlack) {
                                list.add(getBlackNodesWithEqId(MIS,tmp.getId
                                    ()));
                                tmp.setId(idMin);
                        }
                        for(ArrayList<MyPoint> tmp_list : list){
                                for(MyPoint p1 : tmp_list) p1.setId(idMin);
                        }
                        res.add(p);
                }
```

```
        }
        return res;
}
```

• Implémentation de l'Algorithme The Magic MIS

```
public ArrayList<MyPoint> MIS(ArrayList<Point> points, int toz){
        ArrayList<MyPoint> mypoints = new ArrayList<MyPoint>();
        for(Point p : points) {
                mypoints.add(new MyPoint(p, Color.WHITE, false));
        }
        MyPoint m = mypoints.get(toz);
        ArrayList<MyPoint> dominators = new ArrayList<MyPoint>();
        dominators.add(m);

        ArrayList<MyPoint> mNeighbours = myPointNeighbours(m, mypoints, 55);
        for(MyPoint p : mNeighbours) {
                p.setCouleur(Color.GREY);
                ArrayList<MyPoint> v = myPointNeighbours(p, mypoints, 55);
                for(MyPoint q : v)  {
                        if(q == m) continue;
                        q.setActif(true);
                }
        }

        while(containsWhite(mypoints)) {
                MyPoint bestActif = maxWhiteNeighbours(mypoints);
                dominators.add(bestActif);
                ArrayList<MyPoint> neighbours = myPointNeighbours(bestActif,
                    mypoints, 55);
                for(MyPoint p : neighbours) {
                        p.setCouleur(Color.GREY);
                        ArrayList<MyPoint> v = myPointNeighbours(p, mypoints
                            , 55);
                        for(MyPoint q : v) {
                                if(q == bestActif) continue;
                                q.setActif(true);
                        }
                }
        }
        return dominators;
}
```

2.4 Etude pratique

Pour l'étude de l'efficacité des algorithmes en fonction du temps d'exécution et en fonction de la qualité du résultat, nous avons créé une classe java qui exécute directement les

différents algorithmes et imprime en sortie dans un fichier ***data.txt*** les données résultantes de l'exécution.

Dans cette section nous commençons par décrire la structure de données utilisée et la manière de générer notre base de tests. Puis nous allons décrire les résultats sur le temps d'exécution et le la solution minimale retrouvée. Finalement dans la section 3 nous allons interpréter ces résultats en définissant les avantages et les inconvénients de chaque algorithme.

Les tests suivants ont été effectués sur une machine personnelle avec la configuration ci-après : 8 Go de RAM, Processeur Intel Core i3-2310M CPU @ 2.10GHz x 8, une carte graphique GT520M (NVIDIA GeForce) et un système Windows 10 / 64 bits.

2.4.1 Structures de données et Génération des Testbeds

Structure de données

Compte tenu de la structure utilisée en TME, nous avons aussi utilisé une $ArrayList < Point >$, ce qui est considéré comme la liste des sommets. Sur cette structure nous avons utilisé le seuil 55 pour la distance maximale entre deux points voisins.

Génération aléatoire des points du graphe

Nous avons utilisé *LocalThread.random* de la bibliothèque *java.util* afin de générer alétoirement les points dans le graphe. Le nombre de points est passé en paramètre sur la méthode locale et permet de créer des graphes de tailles différentes. Cette méthode retourne une liste de listes de points, donc une $ArrayList < ArrayList < Point >>$

2.4.2 Résultats

Comparaison - Temps d'exécution

Afin de visualiser correctement les différences entre les algorithmes, nous avons lancé 100 tests sur chaque taille de graphe. Dans la table 2.1 une moyenne a été calculée pour chaque algorithme et chaque grandeur de graphe.

TABLE 2.1 – Temps d'exécution moyen par taille du graphe

Taille du graphe	SMIS	Naïf	Magique
500	0.014 secondes	3.6 secondes	0.48 secondes
1000	0.032 secondes	20 secondes	2 secondes
1500	0.054 secondes	60 secondes	4.7 secondes
2000	0.087 secondes	136.3 secondes	5.8 secondes
2500	0.129 secondes	260.8 secondes	13.6 secondes

À partir de ces données nous avons déduit deux comparaisons sur le temps d'exécution. La figure 2.1 représente la comparaison d'une moyenne pour exécuter une instance de x nœuds dans chaque algorithme. La figure 2.2 représente la comparaison d'exécution de plusieurs instances. Celle-ci se base sur une base de tests de 100 instances. Il est important de souligner que chaque instance de test a été calculée par chaque algorithme.

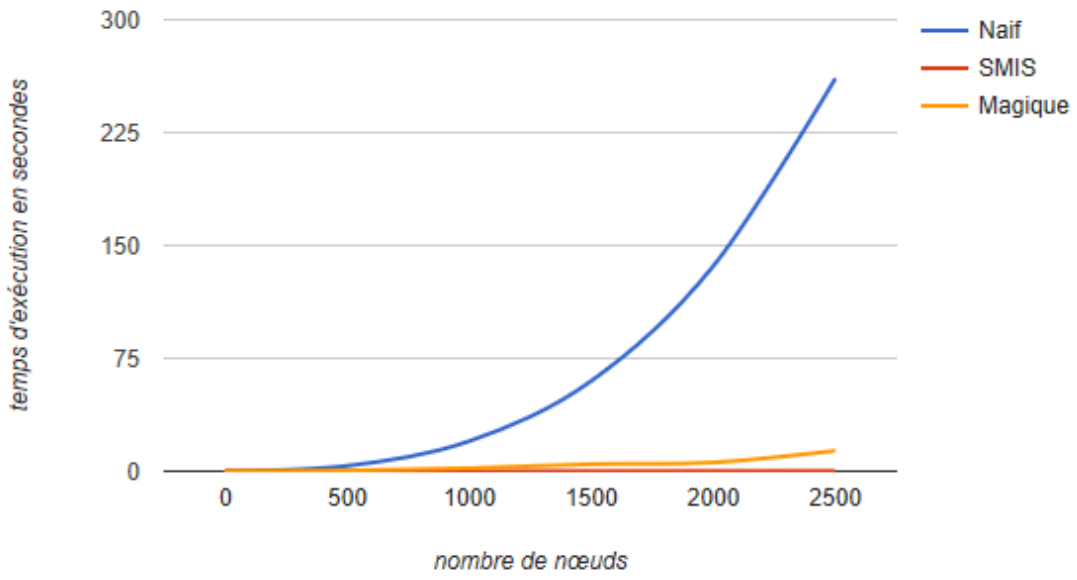

FIGURE 2.1 – Temps d'exécution moyen par taille du graphe

Nous constatons dans les figures 2.1 et 2.2 ci-dessous que le temps d'exécution pour l'algorithme Naïf est le plus lent, 20 secondes sur un graphe de 1000 points. En revanche, l'algorithme de Li et al. est le plus rapide. comme décrit précédemment, cet algorithme a une complexité $O(nlogn)$ et en $O(n^2)$ au pire cas et retourne un résultat assez rapidement.

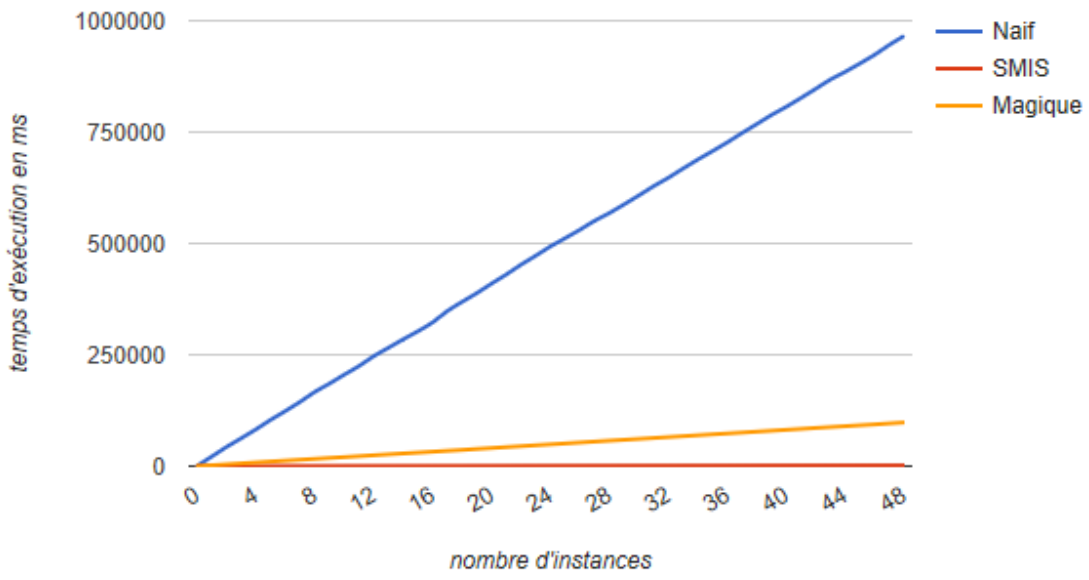

FIGURE 2.2 – Temps d'exécution par nombre d'instances pour graphe de 1000 nœuds

Ci-dessous le tableau récapitulant le nombre d'opérations effectuées par chaque algorithme en prenant en entrée un ensemble de points de taille N.

Le temps d'exécution est calculé en se basant sur une machine ayant un processeur 1 GHz et effectuant donc 10^9 opérations par seconde, cela facilite la tâche si l'on veut calculer le temps d'exécution sur une autre machine ayant un processeur d'une puissance différente.

TABLE 2.2 – Ordre de grandeur du temps nécessaire à l'exécution des algorithmes

N	**Algorithme Naif**		**Algorithme S-MIS**		**The Magique**	
	temps d'exécution	nb opération	temps d'exécution	nb opération	temps d'exécution	nb opération
10	10 μs	10^3	100 ns	10	1 μs	10^2
10^2	1 s	10^6	2 μs	$2(10^2)$	1 ms	10^3
10^3	10 s	10^9	30 μs	$3(10^3)$	10 ms	10^4
10^4	2.7 heures	10^{12}	400 μs	$4(10^4)$	1 s	10^5
10^6	316 ans	10^{18}	60 ms	$6(10^6)$	2.8 heures	10^7

Comparaison - Qualité du résultat

La taille du **CDS** resultant est à minimiser, car nous cherchons le sous-ensmeble minimal qui correspond au critères d'un CDS. Nous pouvons constater avec la figure 2.3 que l'algorithme S-MIS, malgré son efficacité en temps d'exécution, nous retourne le CDS de plus grande taille.

En revanche, l'algorithme The Magic MIS est un très bon compromis en temps de calcul pour un résultat bien plus adéquat à notre problème. Ce dernier retourne un CDS de plus petite taille.

Avec la figure 2.3 nous présentons le résultat obtenu du calcul de la taille moyenne d'un CDS pour 100 instances de 1000 noeuds.

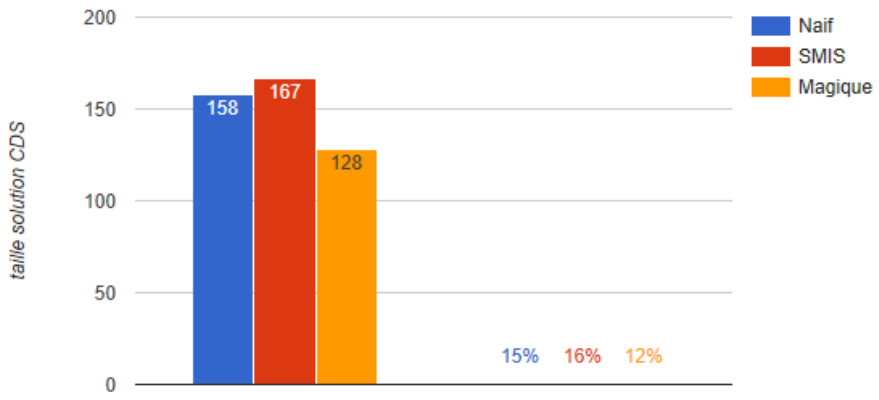

FIGURE 2.3 – taille moyenne d'un CDS retourné par algorithme sur une instance de 1000 nœuds

Avec la figure 2.4 nous comparons le résultat calculé sur 5 grandeurs d'instances différentes (500 nœuds, 1000 nœuds, 1500 nœuds, 2000 nœuds et 2500 nœuds). Sur chaque grandeur nous avons fait le calcul sur 100 instances.

Ce test confirme que la qualité du résultat de l'algorithme de **Li et al**.(S-MIS) retourne le pire résultat. L'algorithme **naïf** retourne un meilleur résultat que le **S-MIS** à partir d'une grandeur comprise entre 500 et 1000 noeuds. Finalement, l'algorithme **'The Magic MIS'** retourne un CDS beaucoup plus petit et donc la meilleur solution parmi les trois algorithmes. Cette solution représente 12% des noeuds du graphe.

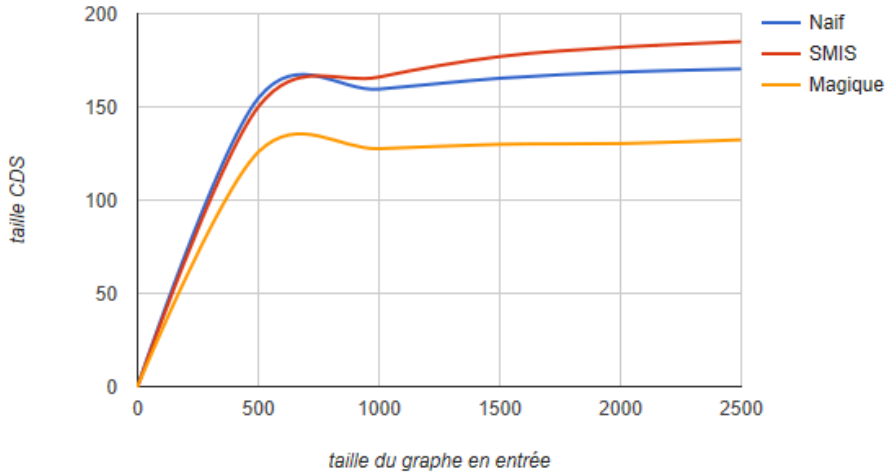

FIGURE 2.4 – Taille du CDS minimal par rapport à la taille du graphe en entrée

Chapitre 3

Discussion

3.1 Algorithme Li et al.

3.1.1 Avantages

Comme nous avons constaté avec les tests présnetés à la section 2.4.2, en termes de temps d'exécution, l'algorithme de Li et al. est de loin le plus rapide, tandis que l'algorithme naïf est beaucoup moins rapide, mais il a le mérite de fournir une meilleure solution s'il prend en entrée un ensemble stable optimisé, ce qui explique son temps d'exécution. Pour finir, nous constaterons que notre algorithme 'The Magic MIS' combine les avantages des 2 autres algorithmes, il a une complexité en $O(N^2)$, ce qui est correct, et offre une solution largement meilleure que les autres.

3.1.2 Inconvénients

L'inconvénient majeur de l'algorithme de S-MIS est qu'il fournit un résultat loin de l'optimal, certes il est très rapide, mais dans le cas du problème du CDS nous pensons que la qualité du résultat est beaucoup plus importante que le temps d'exécution. Bien entendu, il faut le temps d'exécution soit raisonnable. Un autre inconvénient est qu'il prend en entrée un MIS afin de lui rajouter des nœuds (appelés nœuds bleus dans l'article et nœuds de Steiner en général) servant à connecter les nœuds du MIS. Or, un MIS est un Ensemble Stable Maximal, ce qui veut dire que c'est un sous-ensemble contenant le maximum de nœuds du graphe à condition que ces derniers ne soient pas voisins entre eux. Ce n'est pas vraiment le meilleur ensemble de départ, et par conséquent ce n'est pas forcément l'approche la plus intéressante pour résoudre le problème du CDS. Cela a été prouvé par l'algorithme The Magic MIS, qui lui, utilise une approche complètement différente et retourne une solution bien meilleure en temps très raisonnable.

Chapitre 4

Conclusion

Le problème de l'Ensemble Dominant Connexe nous a permis de mettre en évidence les points forts et les points faibles de l'algorithme proposé par **Li et al.**

Nous avons pu étudier et analyser en détails le fonctionnement et la complexité de cet algorithme.Cela nous a amené à constater que cet algorithme s'avère être beaucoup plus rapide par rapport à l'algorithme naïf. Cependant, la taille du CDS résultant est plus grande, donc moins optimale.

Dans le cas d'un problème NP-Difficile et surtout avec les problèmes d'optimisation, la rapidité n'est pas toujours l'essentiel. En effet, l'algorithme naïf, en prenant en entrée un ensemble stable optimisé, fournit un meilleur résultat que l'algorithme de **Li et al**.

Nous avons également pu étudier notre algorithme que nous avions appelé **'The Magic MIS'**, ce dernier n'a même pas besoin de prendre un Ensemble Stable ou un MIS en entrée.Il fournit une solution beaucoup plus optimisée que les 2 autres algorithmes. Afin d'atteindre son objectif, une seule condition est à remplir : son nœud de départ doit être bien choisi.

Malgré le fait qu'il soit très rapide, nous ne sommes pas vraiment convaincues par l'algorithme de **Li et al**. En effet, dans le problème de l'Ensemble Dominant Connexe, nous pensons que la qualité du résultat est très importante, vu que c'est un résultat qui peut être utilisé dans un contexte réel et pratique, par exemple dans les calculs liés au routage pour les réseaux mobiles ad hoc. Nous jugeons donc qu'il vaut mieux perdre en temps d'exécution afin d'améliorer la qualité du résultat plutôt que de faire l'inverse et dégrader la qualité de ce dernier.

Pour conclure, les résultats obtenus avec **'The Magic MIS'** sont bien meilleurs et il est en complexité $O(N^2)$. Le temps d'exécution n'est pas beaucoup plus élevé que celui de l'algorithme de **Li et al**. Il représente une bonne performance avec la meilleure solution en sortie.

Bibliographie

[1] Yingshu Li, My T. Thai, Feng Wang, Chih-Wei Yi, Peng-Jun Wan, and Ding-Zhu Du. On greedy construction of connected dominating sets in wireless networks : Research articles. *Wirel. Commun. Mob. Comput.*, 5(8) :927–932, December 2005.

[2] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4) :374–387, April 1998.

[3] 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*, DIALM '99, pages 7–14, New York, NY, USA, 1999. ACM.

Chapitre 5

Annexes

Nous présentons dans cette section 3 captures d'écran des solutions fournies par chacun des algorithmes vus dans ce rapport en prenant en entrée le même graphe contenant 1000 nœuds.

Test de l'algorithme naïf

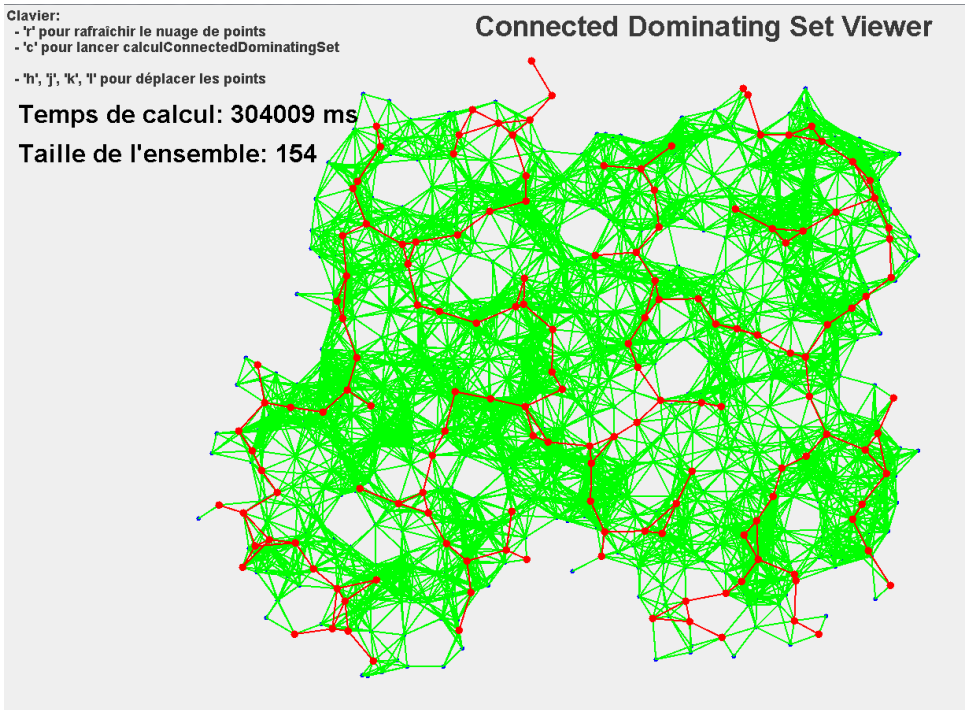

FIGURE 5.1 – Capture d'écran de l'algorithme naïf sur une instance de 1000 nœuds

Test de l'algorithme S-MIS

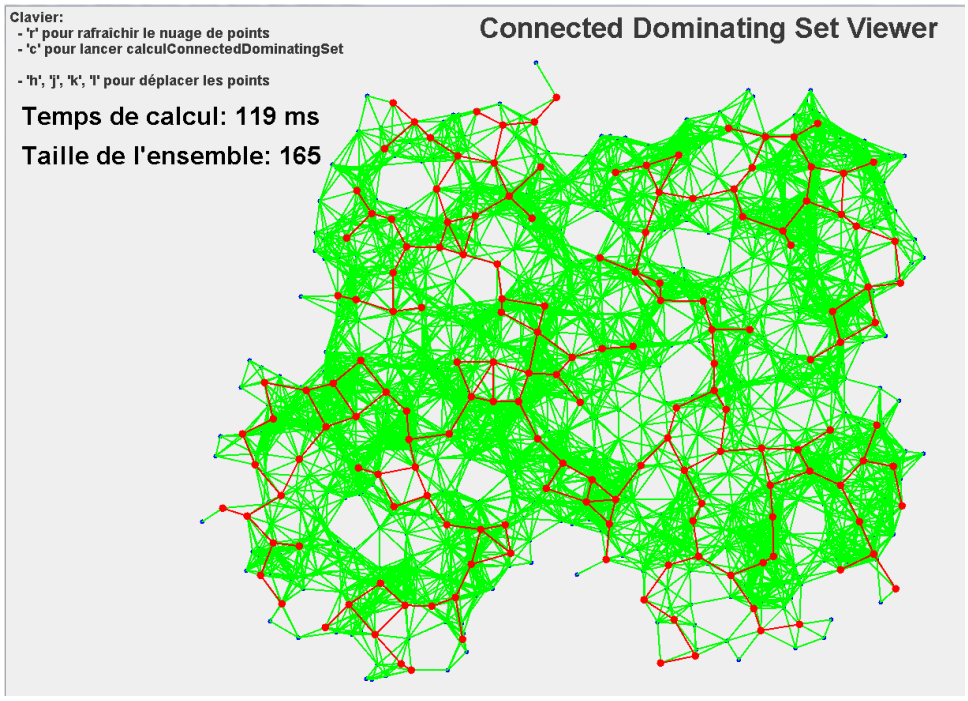

FIGURE 5.2 – Capture d'écran de l'algorithme SMIS sur une instance de 1000 nœuds

Test de l'algorithme "The Magique MIS"

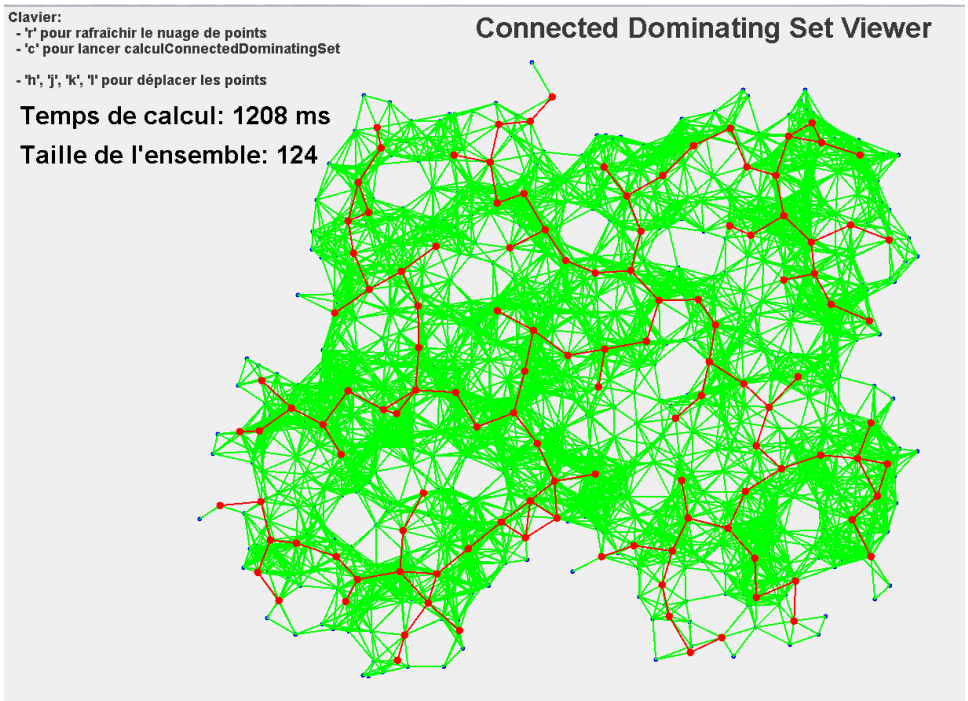

FIGURE 5.3 – Capture d'écran de l'algorithme "The Magic MIS" sur une instance de 1000 nœuds