Rapport
Ensemble dominant connexe minimum

Tanguy Retail
Anastasios Doumoulakis

30 Octobre 2016

Table des matières

Résumé

Nous présentons dans ce rapport une étude du problème d'ensemble dominant connexe pour un graphe géométrique au travers de l'analyse de l'article de Li, Thai, Wang, Yi, Wan, et Du. intitulé "Greedy construction of connected dominating sets in wireless networks" [1]. Ce rapport décrit dans un premier temps le problème de l'ensemble dominant connexe et les difficultés rencontrées lorsque l'on essaie de trouver une solution. On présente ensuite les solutions proposées dans l'article pour résoudre ce problème et quelques critiques.

Dans la partie suivante on propose des améliorations aux algorithmes de l'article pour trouver un ensemble dominant connexe plus efficacement et on présente les performances et résultats des différents algorithmes étudiés. Enfin, on compare les différentes méthodes en prenant en compte les résultats obtenus et la complexité.

1 Introduction

Nous décrivons ici le problème que l'on essaie de résoudre, un exemple d'application réelle que peut avoir ce type de problèmes ainsi que les structures de données utilisées dans les différents algorithmes.

Le problème consiste à trouver dans un graphe géométrique un sous-ensemble dominant connexe, soit un graphe $G = (V, E)$ où V est l'ensemble des sommets du graphe et E représente l'ensemble des arrêtes. Formellement, résoudre le problème du sous-ensemble dominant connexe signifie trouver un ensemble D qui suit la définition suivante.

Définition : Un ensemble D est dominant connexe sur un graphe V si :

$$D = \{D \subseteq V |\ \forall v \in V\ \forall d \in D,\ (v \in D \vee v \in n(D)) \wedge d \in n(D \setminus \{d\})\}$$

On note par $n(D)$ l'ensemble des sommets qui sont voisins des sommets appartenant à D. On cherche donc à trouver un sous-ensemble D de V de taille minimale tel que tous les points qui ne sont pas dans D sont voisins d'au moins un point appartenant à D. Un tel sous-ensemble est *dominant* mais il n'est pas forcément *connexe*. Pour qu'il soit *connexe* il faut en plus que tout points de D soit voisin d'au moins un autre point appartenant à D.

Un tel problème a des applications principalement dans le routage des MANET (Mobile ad hoc Networks). Des réseaux de ce type ont de nombreuses correspondances à des réseaux réels tel qu'un réseau de capteurs transmettant des informations à d'autres capteurs qui sont dans leur rayon de transmission [fig. 1]. Trouver une bonne solution efficacement au problème de l'ensemble dominant connexe minimum permet donc de transmettre un message à un ensemble de capteurs d'un réseau en utilisant un minimum de transmetteurs.

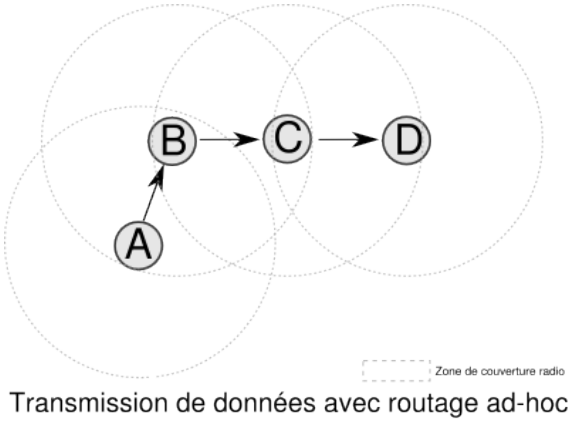

FIGURE 1 – Une représentation d'un réseau de type MANET

La structure de données utilisée durant les tests et l'implémentation des algorithmes pour représenter le graphe $G = (V, E)$ est l'*ArrayList* de Java. On considère qu'un graphe géométrique 2D est constitué d'une liste de points placés sur le plan et d'un entier $d \in \mathbb{N}^*$ tel qu'il existe une arrête $e \in E$ entre $a, b \in V$ si et seulement si $dist(a, b) \leqslant d$ avec $dist(a, b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$ la distance euclidienne. Avec une telle structure de données, on peut alors représenter un sous-ensemble de V par une autre liste L de type *ArrayList* tel que $\forall l \in L$ on a $l \in V$.

Le problème de l'ensemble dominant minimum est NP-complet [2], donc la recherche d'un tel ensemble qui est en plus connexe est également NP-complet. On présente dans la prochaine partie les différents algorithmes existant pour trouver un ensemble qui respecte toutes les propriétés énoncées précédemment.

2 État de l'art

2.1 Quelques notations

Dans cet article nous allons utiliser à de nombreuses reprises des termes récurrents, on propose donc certaines notations qui seront respectées tout le long du rapport. Sauf indication contraire, nous travaillons sur des graphes géométriques en 2D utilisant la structure de données décrite dans la partie précédemment.

On définit donc un graphe de la façon suivante, $G = (V, n)$ le graphe avec V un ensemble de points définis par une abscisse et une ordonnée, on a donc $\forall v \in V,\ v = (x, y)$ avec $x, y \in \mathbb{N}^*$. De plus l'entier $n \in \mathbb{N}^*$ définit le seuil en dessous duquel il existe une arrête entre deux points de V.

En utilisant à nouveau la définition de la partie précédente, on note l'ensemble dominant connexe minimal pour un certain graphe MCDS. On définit également un ensemble dominant indépendant maximal que l'on note MIS qui suit la même définition que le MCDS sauf que $\forall p \in MIS$ on a $p \notin n(MIS \setminus \{p\})$ avec la condition supplémentaire que si l'on ajoute un point à cet ensemble, celui-ci n'est alors plus indépendant.

2.2 L'algorithme de Li et al.

Le problème du MCDS étant NP-complet toute solution que nous trouvons sera une approximation de la solution exacte si la solution exacte est non triviale. On décrit et analyse ici l'algorithme présenté dans l'article "Greedy construction of connected dominating sets in wireless networks". L'algorithme appelé S-MIS est constitué de deux étapes grandes étapes qui utilisent chacune un algorithme différent.

La première étape consiste à trouver un MIS qui respecte le *lemme 2* de l'article original. Ce lemme indique que *toute paire de sous-ensembles complémentaires du MIS a une distance d'exactement deux "hops"*. Une fois le MIS obtenu on peut alors trouver une 3-approximation d'un arbre de Steiner, cet arbre contiendra alors les points qui constituent un CDS sur le graphe. L'existence d'une telle approximation a été démontrée dans un article de 2001 [3]. Pour la génération du MIS qui respecte le *lemme 2* nous avons programmé à la fois un algorithme naïf et le premier des deux algorithmes cités dans l'article qui viennent d'autres publications [4] [5] et qui permettent de produire un tel ensemble.

La deuxième partie de l'algorithme permet donc de produire une liste de points de Steiner qui lient les points du MIS trouvés précédemment. On obtient donc un CDS en joignant les points du MIS aux points Steiner de la deuxième partie de l'algorithme.

2.2.1 Phase 1 : MIS

Définition : Le MIS d'un graphe connexe G est un sous-ensemble de G tel que :

$$\forall u \in G, u \in MIS \vee (\exists v \in MIS \wedge |uv| = 1)$$

La première étape est de générer un MIS d'un graphe , l'article [1] nous propose un algorithme glouton en temps linéaire. Le sous-ensemble obtenu possède alors une propriété intéressante.
Définition : La distance entre deux graphes est définie par la longueur d'un plus court chemin, en la taille du nombre d'arêtes, entre ces deux graphes.
Lemme : Chaque paire d'ensembles complémentaires du MIS, est séparé par une distance égale à 2.
Nous verrons que ce lemme est essentiel pour la phase 2 de l'algorithme. Nous appelons par la suite MIS-L2 tout MIS respectant ce lemme.
Remarque : Tout sous-ensemble de G qui est un MIS de G, est aussi un ensemble dominant (DS) de G.

Algorithme 1 : MIS-L2

Données : Un graphe connexe géométrique G de points blancs inactifs
Résultat : Un MIS-L2 de G

```
1  focus ← un point de G;
2  focus.statut ← actif ;
3  tant que il reste un actif faire
4  |  focus ← le point actif qui a le plus de voisins blancs ;
5  |  focus.statut ← inactif ;
6  |  focus.couleur ← noir ;
7  |  voisins(focus).couleur ← gris ;
8  |  voisins_blancs(voisins(focus)).statut ← actif ;
9  fin
10 retourner noirs ;
```

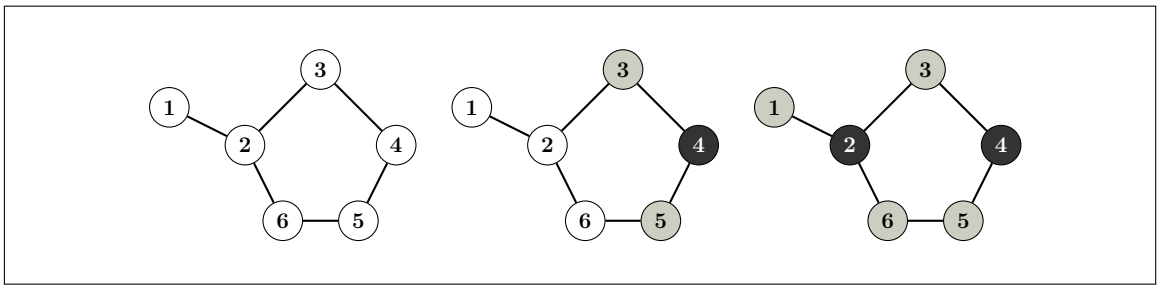

FIGURE 2 – Exemple de MIS-L2

Dans l'exemple, le point 4 est choisi et devient noir. Ses voisins 3 et 5 deviennent gris et les voisins de 3 et 5 deviennent actifs. Il faut choisir le prochain point entre 2 et 6 qui sont actifs ; 2 a le plus grand nombre de voisins blancs et devient donc noir. Ses voisins deviennent gris et il ne reste plus de points blancs donc l'algorithme se termine. L'ensemble obtenu est 2,4, deux ensembles complémentaires sont 2 et 4, qui sont bien séparés par une distance égale à 2 (Le chemin minimal est de deux arrêtes en passant par 3).

Complexité

Dans la boucle (l.3), l'algorithme parcourt tous les points en un temps $\mathcal{O}(n)$ tandis que la recherche des voisins (l.4, l.7, l.8) s'effectue également en $\mathcal{O}(n)$ si les nœuds ne possèdent pas l'information de leurs voisins (ce qui est le cas dans un graphe géométrique). La complexité en temps est donc en $\mathcal{O}(n^2)$, avec n le nombre de nœuds de G. Si le graphe est représenté par une liste d'adjacence, on pourra faire tomber cette complexité à $\mathcal{O}(n)$.

2.2.2 Phase 2 : ST-MSN

Grâce à la phase 1 nous obtenons donc un ensemble dominant indépendant ; il reste à rendre cet ensemble connexe. Pour ce faire, la deuxième phase propose d'utiliser un algorithme glouton d'approximation d'un arbre de Steiner avec un minimum de nœuds de Steiner (ST-MSN). L'ensemble obtenu à la phase 1 constitue l'ensemble des points dominants, et il faut trouver le minimum de nœuds de G (les nœuds de Steiner) capables de rendre cet ensemble connexe. L'ensemble des nœuds dominants

est un ensemble dominant indépendant de G, en ajoutant des nœuds de G rendant cet ensemble connexe, nous obtenons donc bien un ensemble dominant connexe (CDS).

Algorithme 2 : Approximation MCDS

Données : Un graphe G, un MIS-L2 de G
Résultat : Une approximation d'un MCDS de G

```
1 pour tout i de 5 à 2 faire
2 |   tant que il existe un nœud gris candidat qui a au moins i voisins de
  |   composantes connexes bleues-noires différentes faire
3 |   |   candidat.couleur ← bleu ;
4 |   fin
5 fin
6 retourner bleus ∪ noirs ;
```

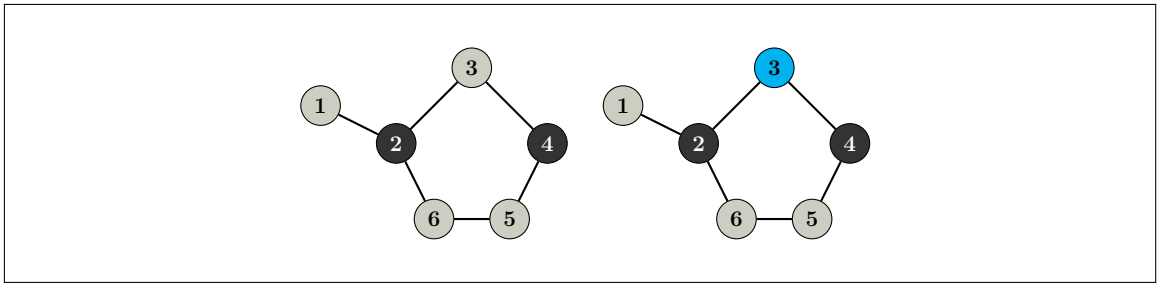

FIGURE 3 – Exemple de CDS

Au début, les points 2 et 4 sont dans des composantes connexes différents. Le premier candidat trouvé est le point 3 qui a deux composantes connexes bleues-noires différentes auxquelles appartiennent respectivement 2 et 4. Le point 3 devient donc bleu et 2,3,4 forment une composante connexe bleue-noire. Remarquons l'intérêt du lemme de la phase 1 : si un MIS ne respecte pas le lemme, l'algorithme présenté ne pourra le rendre connexe puisque chaque composante ne pourra pas être connectée avec une autre par un seul point.

Complexité

La première boucle (l.1) a une complexité constante, la deuxième boucle parcourt les nœuds gris en trouvant leurs voisins (et donc les composantes auxquelles ils sont rattachés) parmi les bleus-noirs, soit $\prime(n \times m)$ avec n le nombre de nœuds gris, m le nombre de nœuds bleus et noirs. Si le graphe est représenté par une liste d'adjacence, on pourra faire tomber cette complexité à $\mathcal{O}(n)$.

3 Critiques et améliorations

Plusieurs remarques peuvent être adressées à cet algorithme, sur son efficacité mais aussi sur les explications données par les auteurs.

Il est important dans un premier temps de mettre en perspective les caractéristiques de l'algorithme S-MIS, dans l'article [1] il est indiqué que l'algorithme est censé produire un CDS dont la taille à une meilleure borne supérieure que les algorithmes précédents. C'est-à-dire que si l'on note n la taille de la solution optimale

(du MCDS), la taille du CDS obtenu avec S-MIS est bornée par :

$$6.40944n + 1.2$$

Cela signifie que pour une solution optimale de 5 on peut potentiellement obtenir un CDS de taille 33.2472, on voit donc que l'approximation de l'algorithme est assez importante, l'avantage de l'algorithme réside donc dans le fait que son temps d'exécution est court.

La deuxième phase de l'algorithme est censée rendre un ensemble de points qui constituent les points de Steiner (points bleus), c'est-à-dire les points qui lient entre eux les points de l'ensemble MIS rendu dans la première phase (points noirs) qu'on pourrait également obtenir en passant le MIS directement à l'algorithme de Steiner. Pour obtenir un CDS a partir de l'algorithme de l'article il faut donc faire l'union des points bleus et noirs mais cette solution contient beaucoup de points en trop que l'on peut éliminer avec un simple algorithme de filtrage que l'on présente par la suite.

4 Solution proposée

4.1 Autres algorithmes à partir des précédents

4.1.1 MIS-L2 avec nouvelle méthode de sélection

Algorithme 3 : MIS-L2 alternatif

Données : Un graphe connexe géométrique G de points blancs inactifs
Résultat : Un MIS-L2 de G

```
1 focus ← un point de G;
2 focus.statut ← actif ;
3 tant que il reste un actif faire
4 |   focus ← le point actif qui a le plus de voisins actifs ;
5 |   focus.statut ← inactif ;
6 |   focus.couleur ← noir ;
7 |   voisins(focus).couleur ← gris ;
8 |   voisins_blancs(voisins(focus)).statut ← actif ;
9 fin
```

Dans l'article [4], les auteurs sélectionnent à chaque fois l'actif qui a le plus de voisins blancs. Nous proposons de sélectionner l'actif qui a le plus de voisins actifs. Ainsi, la recherche du prochain actif est légèrement plus rapide. De plus, le résultat pourrait éventuellement produire un résultat différent, mais qui est toujours un MIS-L2.

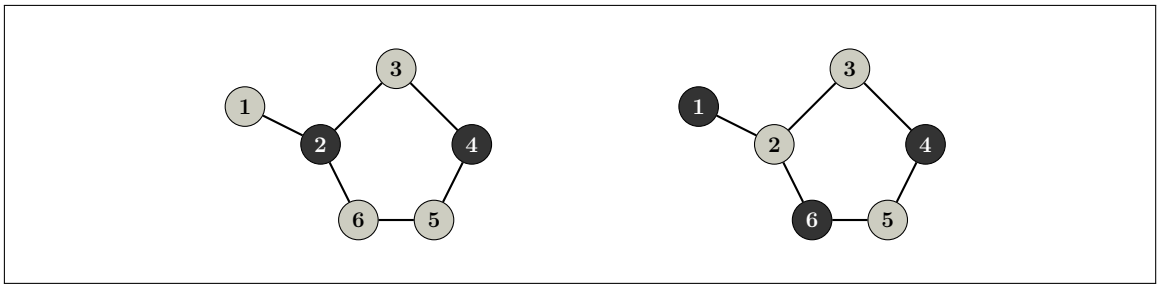

FIGURE 4 – Exemples de MIS-L2

Dans cet exemple, deux MIS-L2 différents, en sélectionnant le point 4 en premier.

4.1.2 S-MIS avec filtre

Algorithme 4 : S-MIS filtré

Données : Un graphe G, un MIS-L2 de G
Résultat : Une approximation d'un MCDS de G

```
pour tout i de 5 à 2 faire
    tant que il existe un nœud gris candidat qui a au moins i voisins d'une
    composante connexe bleue-noire différente faire
        candidat.couleur ← bleu ;
    fin
fin
tant que il existe un point inutile noir ou bleu dont la suppression donne
toujours un sous-ensemble qui est un CDS de G faire
    inutile.couleur ← gris;
fin
retourner bleus ∪ noirs ;
```

Nous présentons ici une version du S-MIS avec un filtre supplémentaire pour se débarrasser des points en trop. En effet, quelque soit l'algorithme de MIS utilisé, le S-MIS retourne un ensemble qui peut-être facilement amélioré, puisque certains points peuvent être enlever de la solution, sans que celle-ci ne devienne invalide.
L'algorithme de filtrage que nous utilisons pour améliorer la solution de S-MIS est très simple mais donne de bonnes améliorations ce qui renforce le fait que l'algorithme de l'article n'est pas optimal. On parcourt chaque point de la solution proposée, on le supprime puis on vérifie que la solution est toujours valide. Si elle l'est on continue avec la nouvelle solution, sinon on rajoute le point supprimé puis on continue avec le prochain point.
Ce simple parcours de la solution en éliminant les points inutiles nous coûte $\mathcal{O}(n^2)$ c'est à dire autant que S-MIS, et élimine une partie non négligeable des points comme on le verra dans la présentation des résultats expérimentaux.

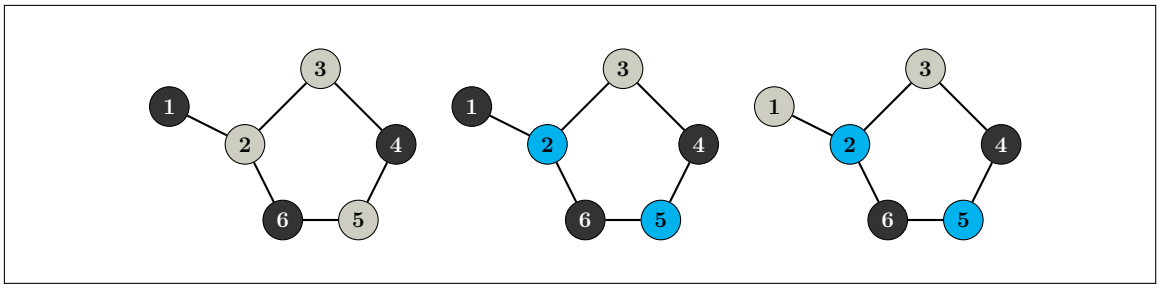

FIGURE 5 – Exemple de filtrage

Dans cet exemple, le deuxième graphe représente la solution apportée par l'algorithme du S-MIS en partant du MIS du premier graphe. Après le filtrage du troisième graphe, la solution est meilleure et valide.

4.1.3 CDS glouton à partir de l'algorithme du MIS-L2

Voici également un algorithme glouton qui est une modification de l'algorithme de la phase 1 pour produire directement un CDS sans passer par la phase 2.
Au lieu de sélectionner le point actif qui a le plus de voisins blancs, nous sélectionnons le point gris qui a le plus de voisins blancs. Ainsi, le nouveau point est directement connexe aux précédents et nous pouvons directement obtenir un CDS.

Algorithme 5 : CDS glouton naïf

Données : Un graphe connexe géométrique G de points blancs inactifs
Résultat : Un CDS de G

```
1  focus ← un point de G;
2  focus.statut ← gris ;
3  tant que il reste un blanc faire
4  |  focus ← le point gris qui a le plus de voisins actifs ;
5  |  focus.statut ← inactif ;
6  |  focus.couleur ← noir ;
7  |  voisins(focus).couleur ← gris ;
8  |  voisins_blancs(voisins(focus)).statut ← actif ;
9  fin
10 retourner noirs ;
```

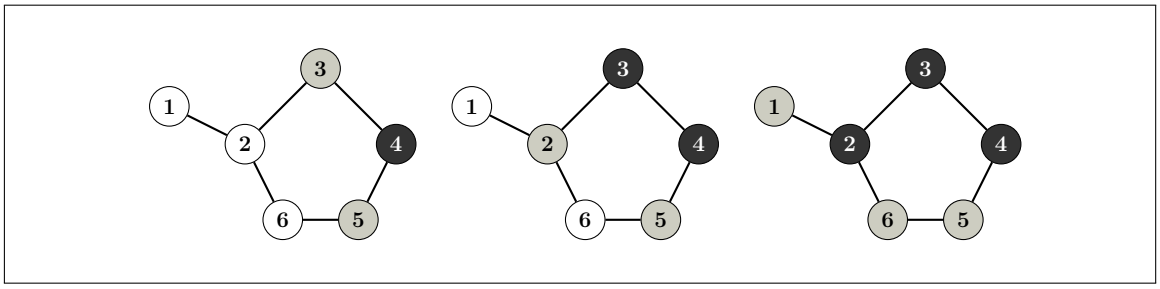

FIGURE 6 – Exemple du CDS glouton

Dans cet exemple, on trouve le déroulement de l'algorithme avec l'ensemble noir qui s'étend directement à travers le graphe.

4.2 Technique de recherche locale

On décrit ici la technique de local searching utilisée pour améliorer un peu plus la solution obtenue à partir du S-MIS mais en sacrifiant de l'efficacité. En effet, la complexité en temps du local searching est en $\mathcal{O}(n^4)$ alors que l'algorithme pour produire le MIS est en $\mathcal{O}(n^2)$ [4] [6] et l'algorithme de la phase 2 est en également en $\mathcal{O}(n^2)$ [1].

Voici le pseudo code de l'algorithme de recherche locale :

Algorithme 6 : Local search

```
   Données : Un graphe G, Un CDS C
   Résultat : Un CDS plus petit ou égal à C
 1 pour tout (a, b) de point du CDS ∧ a ≠ b faire
 2 |   pour tout p de (G \ CDS) faire
 3 |   |   CDS.remove(a, b);
 4 |   |   CDS.add(p);
 5 |   |   si Dominant(G, CDS) ∧ Connexe(G, CDS) alors
 6 |   |   |   On recommence la première boucle à zéro
 7 |   |   |   avec le nouveau CDS contenant un point en moins
 8 |   |   sinon
 9 |   |   |   CDS.add(a, b);
10 |   |   |   CDS.remove(p);
11 |   |   fin
12 |   fin
13 fin
14 retourner CDS;
```

On voit ici qu'on parcourt tous les points du CDS puis une partie des points du graphe G, puis pour chaque triplet de points on remplace les deux points du CDS par un point de G. On vérifie alors que le nouvel ensemble est bien un CDS, cela requiert deux autres parcours du graphe, un pour vérifier que le nouvel ensemble est connexe et l'autre pour vérifier qu'il est toujours dominant.

On voit donc que pour un coût important en temps de calcul on peut améliorer un peu plus le résultat obtenu après le filtre décrit précédemment. On peut également faire une recherche locale avec n'importe quel CDS mais l'algorithme prendra plus de temps pour optimiser un mauvais CDS.

5 Résultats expérimentaux

5.1 Description des tests

Génération de graphe : Nous avons développé pour réaliser nos tests un générateur de graphes connexes (Random Connected Geometric Graph Generator - RC3G) accessible sur le site Github à partir du lien dans la bibliographie [7]

Machine utilisée : Lors de nos tests, nous avons utilisé une seule et même machine, afin de ne pas fausser les résultats. La configuration est Ubuntu16.04 avec un Intel Core I54200M 2.5GHz×4 et 8GiB de RAM, enfin les tests ont été lancé depuis l'IDE Eclipse Neon.
Les résultats de ces tests peuvent être retrouvé sur le github du projet [8].

Les tests de tous les algorithmes se sont passés de la façon suivante. Nous avons généré 1000 graphes aléatoires, avec pour tous les 100 graphes des tailles variant de 100 points à 1000 points. Chaque algorithme est donc testé sur un total de 1000 graphes de tailles allant de 100 à 1000 points. Bien entendu, chaque algorithme a été testé avec les mêmes 1000 graphes générés.

On présente deux types de courbes différentes, le premier type est le temps de calcul moyen des solutions en fonction de la taille du graphe en entrée, c'est-à-dire que la courbe doit refléter la complexité en temps de chaque algorithme.
Le deuxième type de courbe présente la taille de la solution en fonction au carré de la taille du graphe d'entrée, une plus petite valeur signifie donc un résultat d'une meilleure précision.

5.2 Résultats

On compare ici (fig. 7) la méthode de sélection des points pour le MIS-L2 c'est-à-dire celle de l'algorithme 1 avec celle de l'algorithme 3, on passe ensuite chacune des solutions à l'algorithme 2 pour obtenir un CDS. On compare également les résultats après le filtre décrit précédemment dans l'algorithme 4.

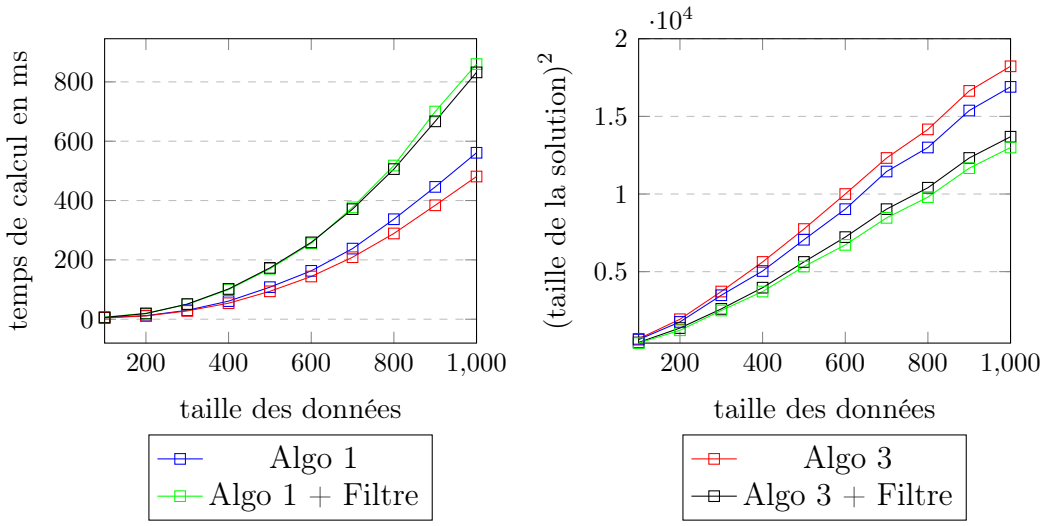

FIGURE 7 – Comparaison de la performance des algorithmes 1 et 3 avec et sans filtre

On peut à présent appliquer l'algorithme 6 de local search au résultat de l'algorithme 1 avec filtre et de l'algorithme 5, que l'on appelle glouton, et comparer les données (fig. 8). Dans le graphe comparant les temps d'exécution, on utilise une échelle logarithmique pour les ordonnées.

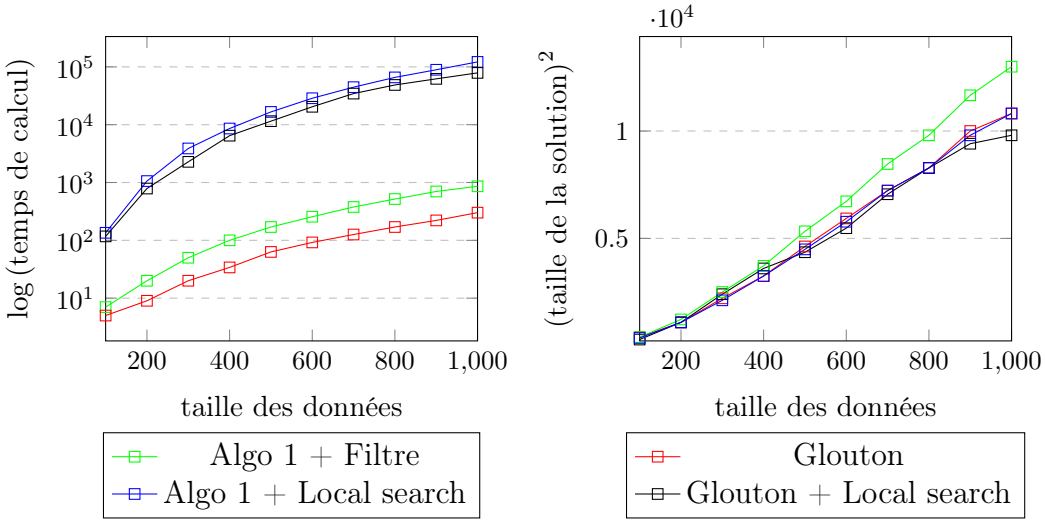

FIGURE 8 – Étude de l'effet de local search et performance du glouton

On compare à présent la solution précédente avec l'algorithme 5 qui est une modification de l'algorithme 1 qui produit directement un CDS. On peut également comparer les performances du glouton avec un filtre et la technique de local search.

6 Discussion

Voici un tableau comparatif qui résume les résultats présentés dans la partie précédente.

algorithmes	**vitesse**	**approximation**
Algo 1	++	+
Algo 1 & filtre	++	++
Algo 1 & local search	+	++
Algo 3	++	+
Algo 3 & filtre	++	++
glouton	+++	++
glouton & local search	+	+++

Grâce à ce tableau comparatif on peut voir clairement les forces et les faiblesses de chaque algorithme. De façon générale on peut appliquer la recherche locale à n'importe quel algorithme pour obtenir de meilleurs résultats mais cela se fait au prix du temps de calcul. Ce tableau permet également de réaliser que le meilleur choix si l'on souhaite trouver un CDS pour un graphe géométrique est l'algorithme glouton car il offre le meilleur rapport précision/temps de calcul.

7 Conclusion

Nous avons donc analysé l'algorithme proposé par Li, Thai, Wang, Yi, Wan, et Du. intitulé "Greedy construction of connected dominating sets in wireless networks", qui est en fait une combinaison de deux algorithmes qui permettent d'obtenir un CDS. On a montré que cet algorithme est rapide mais aussi inefficace car il donne une approximation assez importante de la solution optimale. Nous avons alors testé trois autres approches pour obtenir un CDS que nous avons comparé à l'algorithme S-MIS dans une série de tests.

Grâce à ces résultats expérimentaux et la discussion qui les suit nous avons donc conclu que nous avons à notre disposition une multitude d'algorithmes différents. En fonction de nos besoins en rapidité ou précision du résultat on peut choisir l'algorithme qui accommodera au mieux les contraintes qui sont imposées. Mais plus généralement l'algorithme glouton (numéro 5) donne le meilleur ratio entre le temps de calcul et la précision du résultat et on peut ajouter une recherche locale pour avoir un résultat très précis mais au prix d'un temps d'exécution élevé.

On peut donc conclure que comme pour la majorité des problèmes NP-complets, il existe des algorithmes plus performant que d'autres mais généralement il y a un compromis entre une solution proche de la solution optimale et un temps de calcul rapide.

Références

[1] Y. Li, M. T. Thai, F. Wang, C. W. Yi, P. J. Wan, and D. Z. Du, "On greedy construction of connected dominating sets in wireless networks," *Wireless Communications and Mobile Computing*, vol. 5, no. 8, pp. 927–932, 2005.

[2] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," *Discrete Mathematics*, vol. 86, no. 1, pp. 165–177, 1990.

[3] D. Chen, D.-Z. Du, X.-D. Hu, G.-H. Lin, L. Wang, and G. Xue, "Approximations for Steiner trees with minimum number of Steiner points," *Theoretical Computer Science*, vol. 262, pp. 83–99, 2001.

[4] M. Cardei, X. Cheng, X. Cheng, and D.-Z. Du, "Connected Domination in Multihop Ad Hoc Wireless Networks," no. November 2015, pp. 1–5, 2002.

[5] P.-J. Wan, K. M. Alzoubi, and O. Frieder, "Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks," *Mobile Networks and Applications*, vol. 9, pp. 141–149, apr 2004.

[6] Peng-Jun Wan, K. Alzoubi, and O. Frieder, "Distributed construction of connected dominating set in wireless ad hoc networks," in *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, pp. 1597–1604, IEEE.

[7] A. D. Tanguy Retail, "Random Connected Geometric Graph Generator," 2016 `https://github.com/tanguinoche/RC3G`.

[8] A. D. Tanguy Retail, "Code et tests du projet d'AGAA," 2016 `https://github.com/archonSTB/ProjetAAGA`.