

Rapport sur l'algorithme de calcul d'Ensemble
dominant connexe minimal présenté dans l'article
*On greedy construction of connected
dominating sets in wireless networks* de Li,
Thai, Wang, Yi, Wan, Du, Tong, Bin, Zao, Wun,
Zhou, 2005 (1).

Darius Mercadier

Jordi Bertran de Balanda

Contents

1	Introduction	2
2	Présentation de l'algorithme	2
2.1	Calcul du MIS	2
2.1.1	Implémentation de l'algorithme de calcul du MIS	3
2.1.2	Complexité du calcul du MIS	4
2.2	Calcul du S-MIS : algorithme de Li et al.	5
2.2.1	Principe de l'algorithme	5
2.2.2	Analyse de la complexité	6
3	Performances	6
3.1	Méthode d'évaluation des performances	6
3.2	Taille du graphe géométrique	7
3.2.1	Densité et forme du graphe	7
3.3	Connexité du graphe géométrique	8
4	Références	9

1 Introduction

Un ensemble dominant d'un graphe $G = (S, A)$ est un sous-ensemble D de S tel que pour toute arête $uv \in A$, $u \in D$ ou $v \in D$. Le problème consistant à trouver un ensemble dominant connexe de taille minimal (MCDS) est NP-Difficile. Dans ce rapport, nous étudierons l'algorithme *S-MIS* présenté dans *On greedy construction of connected dominating sets in wireless networks* de Li, Thai, Wang, Yi, Wan, Du, Tong, Bin, Zao, Wun, Zhou, 2005, qui propose un Schema d'approximation en temps polynomial (PTAS) donnant une $(4.8 + \ln(5))$ -approximation de la solution optimale. Nous commencerons par présenter l'algorithme, puis présenterons les résultats expérimentaux obtenus, que nous comparerons avec d'autres algorithmes de résolution de ce problème.

Plus précisément, nous étudierons cette algorithme dans le contexte des graphes géométriques. Ceux-ci sont composés d'un ensemble de sommets S et d'arêtes A telles que $uv \in A$ si et seulement si $distance(u, v) \leq k$, k étant un seuil fixe. Nous présenterons également les générateurs aléatoires utilisés pour générer les graphes de tests.

2 Présentation de l'algorithme

L'algorithme *S-MIS* consiste en deux étapes : le calcul d'un *Maximum Independent Set* (MIS), puis le calcul du MCDS.

Le papier présentant l'algorithme *S-MIS* ne présente pas d'algorithme permettant le calcul du MIS, mais suggère deux approches de calcul (2,3). Nous avons donc implémenté l'algorithme de Wan, Alzoubi et Frieder (3).

2.1 Calcul du MIS

Un ensemble indépendant dans un graphe $G = (S, A)$ est un sous-ensemble D de S tel que pour tout $u \in D$ et $v \in D$, $uv \notin A$. Le problème de calculer un ensemble indépendant maximum est NP-difficile. C'est donc sur une α -approximation que nous avons implémenté.

Le MIS nécessaire au calcul du MCDS avec l'algorithme *S-MIS* doit de plus satisfaire une condition supplémentaire : pour tout $u \in D$, il doit exister $w \in S$ tel qu'il existe $v \in D$, $v \neq u$ tel que $uw \in A$ et $vw \in A$. Moins formellement, cela signifie qu'entre deux points appartenant au MIS, il doit y avoir un et un seul point n'appartenant pas au MIS.

Les figures 1 et 2 montrent toutes les deux des MIS : tous les sommets sont soit dans le MIS soit ont un voisin dans le MIS, et aucun sommet du MIS n'a de voisin dans le MIS. Cependant, dans le second, les deux sommets du MIS sont séparés d'une distance de deux sommets tandis qu'il ne sont séparés que d'un

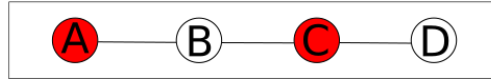


Figure 1: Un MIS valide comme base de l'algorithme S-MIS

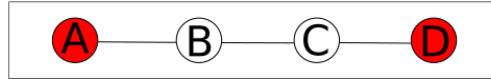


Figure 2: Un MIS invalide comme base de l'algorithme S-MIS

sommet dans le premier. Par conséquent, seule la figure 1 représente un MIS valide comme base de l'algorithme S-MIS.

2.1.1 Implémentation de l'algorithme de calcul du MIS

L'algorithme utilisé pour calculer le MIS se base sur un système de couleur pour différencier les points non-visités (blancs) des points appartenant au MIS (noirs) et des points n'appartenant pas au MIS (bleus) : on part d'un point au hasard du graphe, que l'on marque noir (il est le premier point du MIS). On marque tous ses voisins bleus (il ne peuvent pas appartenir au MIS). Puis on ajoute les voisins des voisins qui sont encore blancs à la liste des points potentiellement dans le MIS. On retire le premier point de cette liste et on réitère le processus tant qu'il reste des points à examiner.

De manière plus pratique, le pseudo-code de cet algorithme est le suivant :

```

def MIS ( G = (V,E) ) :
    MIS = []
    for (p : V) :                                # Initializing the
        ↪ colors.
        p.color = White

    Stack = [V.pop]
    while (Stack.notEmpty) :
        current = Stack.pop
        if (current.color == Blue) :               # Already covered
            continue
        current.color = Black                      # Adding it to the
        ↪ MIS
        MIS.add(current)

        for (p : current.neighbors) :
            p.color == Blue                        # Marking the
            ↪ neighbors as covered
  
```

```

    for (p : current.neighbors) :
        for (q : p.neighbors) :
            if (q.color == White) :           # Adding the
                ↪ neighbors of the neighbors
                Stack.add(q)                   # to the
                ↪ potential points of the MIS.

return MIS

```

2.1.2 Complexité du calcul du MIS

On note $n=|S|$, et $m=|E|$.

Initialisation :

Initialiser les couleurs des sommets requière un unique parcours des sommets, en temps linéaire en la taille de S : $\mathcal{O}(n)$.

Pour des questions d'optimisation, on précalculera lors de l'initialisation une table d'association point-voisins qui a chaque point associera ses voisins. Cette opération est réalisable en $\mathcal{O}(n*m)$, et permet de réaliser l'opération **neighbors** en $\mathcal{O}(1)$.

Boucle principale :

Le pseudo-code précédemment donné est une version simplifié de l'algorithme réel pour des raisons de lisibilité, en particulier, dans une vrai implémentation un point n'est ajouté à la stack que si il n'y est pas déjà et si il est blanc. En tenant compte de cette condition, et en constatant qu'il y a autant d'itération de la boucle principale qu'il y a d'éléments qui sont ajoutés dans la stack lors de l'exécution de l'algorithme, et vu qu'un sommet blanc enlevé de la Stack est marqué noir, on en conclue que le nombre d'itération de cette boucle est borné par n .

On a expliqué précédemment que l'opération **neighbors** est réalisable en temps constant. Le nombre de voisins d'un points cependant est uniquement bornée par m . Par conséquent la boucle parcourant les voisins des voisins a une complexité en $\mathcal{O}(m*m)$.

La complexité de la boucle principale est donc $\mathcal{O}(n*m*m)$.

Il convient cependant de noter que dans les instances traitées, cette limite est une sur-approximation très large. En effet, les graphes étant géométriques, la distribution des sommets aléatoire et uniforme, et le seuil k très inférieur à la distance entre les extrêmes du domaine de définition des sommets, le nombre d'arrête par sommets sera très inférieur à m .

Par conséquent, une complexité plus réaliste serait de l'ordre de $\mathcal{O}(n*m)$. (Cela revient à supposer que le nombre d'arrêtes par sommets est de l'ordre de \sqrt{m} ; ce chiffre dépend en réalité de la quantité de sommets, de l'air de la surface dans

laquelle ils sont, et de l'uniformité de leur répartition. Dans nos test, le nombre d'arrêtes par sommets est en effet bien plus proche de \sqrt{m} que de m).

2.2 Calcul du S-MIS : algorithme de Li et al.

2.2.1 Principe de l'algorithme

L'algorithme se base sur un Lemme inhérent à la manière dont le MIS est construit : il faut au maximum ajouter un point pour connecter deux points. (cf les figures 1 et 2).

Informellement, à partir de ce Lemme, le principe de l'algorithme est de regrouper des clusters de sommets de manières greedy : on ajoute un à un les points qui permettent de regrouper le maximum de clusters.

L'algorithme utilise un système de couleurs pour déterminer les points appartenant au MCDS car ils appartiennent au MIS (black) ou car on les y a rajoutés (blue), et ceux dont le status est encore à déterminer (grey). Les "clusters" de sommets à reliés sont appelés *black-blue component* (car ils sont induits par des points bleus et noirs conjoints, en ignorant les connexions entre sommets bleus). Afin de les relier, on trouve les sommets gris qui sont voisins du plus de points noirs de différents clusters possible (cette affirmation est légèrement inexacte car *du plus* est en réalité *5 ou plus*, puis *4*, puis *3*, *2* et finalement *1*, car en effet, tester toutes les possibilités reviendrait à rajouter un facteur m dans l'algorithme alors qu'en tester *5* ne fait qu'ajouter une constante). C'est cette partie de l'algorithme qui est greedy : on ajoute les points qui sur le moment semble aider au maximum à rendre le MIS connexe.

Le pseudo-code est le suivant :

```
# Params : MIS, the MIS obtained with the previously
    ↪ described method,
#           UDG, the graph from which the CDS should be
    ↪ returned.

def AlgorithmA (MIS, UDG) :
    # Initialization
    for (p ∈ MIS) :
        p.color = black
    for (p ∈ UDG and p ∉ MIS) :
        p.color = grey

    # Main loop
    for (i in 5,4,3,2) :
        if (∃ p ∈ UDG | p.color = grey and p is adjacent
            ↪ to at least i black nodes in different black
            ↪ -blue compoments ) :
```

```

    # Then we choose p to connect those different
    ↪ black-blue components :
    p.color = blue

return { p ∈ UDG | p.color = blue }

# The S-SMIS is then: MIS ∪ AlgorithmA(MIS,UDG)

```

2.2.2 Analyse de la complexité

2.2.2.1 Complexité spatiale

Premier point, toutes les opérations de l'algorithme se font *en place*, c'est à dire sans créer de nouvelles structures, mais uniquement en travaillant sur les couleurs des sommets. Par conséquent, l'espace mémoire nécessaire à cet algorithme est uniquement l'espace utilisé pour stocker le graphe, linéaire en le nombre de sommets.

Quelques structures de données supplémentaires peuvent cependant s'avérer utiles, par exemple pour stocker les *black-blue components*, et pour accéder aux voisins de chaque noeuds, mais l'espace occupé restera cependant en $O(n)$.

2.2.2.2 Complexité temporelle

3 Performances

3.1 Méthode d'évaluation des performances

Les données obtenues dans cette section sont des moyennes du calcul de S-MIS sur des graphes générés aléatoirement avec abscisses et ordonnées des points du graphe choisies uniformément. Nous prenons 500 échantillons par mesure pour pallier aux éventuels cas pathologiques de graphes, qui arrivent surtout avec des tailles de graphes faibles (≤ 250 points).

Ceci a pour conséquence de créer un graphe bien moins 'centré' que ceux rendus disponibles par **supportGUI**. En conséquence l'évaluation est moins adaptée à des réseaux qui seraient centralisés autour d'un pôle.

Nous distinguons 3 paramètres sur les points en entrée qui peuvent influencer sur la performance des algorithmes de calcul d'un ensemble dominant connexe.

1. La densité du nuage de points par rapport à leur fourchette d'abscisse et d'ordonnées
2. La valeur du seuil associé au graphe géométrique
3. La taille du nuage de points à traiter, à densité égale et seuil correspondant

Afin de ne pas causer de changement involontaire de l'un des paramètres lors de l'analyse d'un autre, nous modifions les paramètres annexes.

3.2 Taille du graphe géométrique

3.2.1 Densité et forme du graphe

On cherche ici à examiner l'impact de la seule augmentation de la taille du graphe sur les performances de l'algorithme. De manière à préserver le nombre moyen de voisins d'un noeud, on cherche à contrôler la 'densité' du graphe en variant les plages de valeurs d'abscisses et d'ordonnées disponible pour respecter:

$$d = \frac{n}{r^2}$$

avec d la densité du nuage de points, n sa taille et r la taille du carré 2D à l'intérieur duquel les points sont générés aléatoirement.

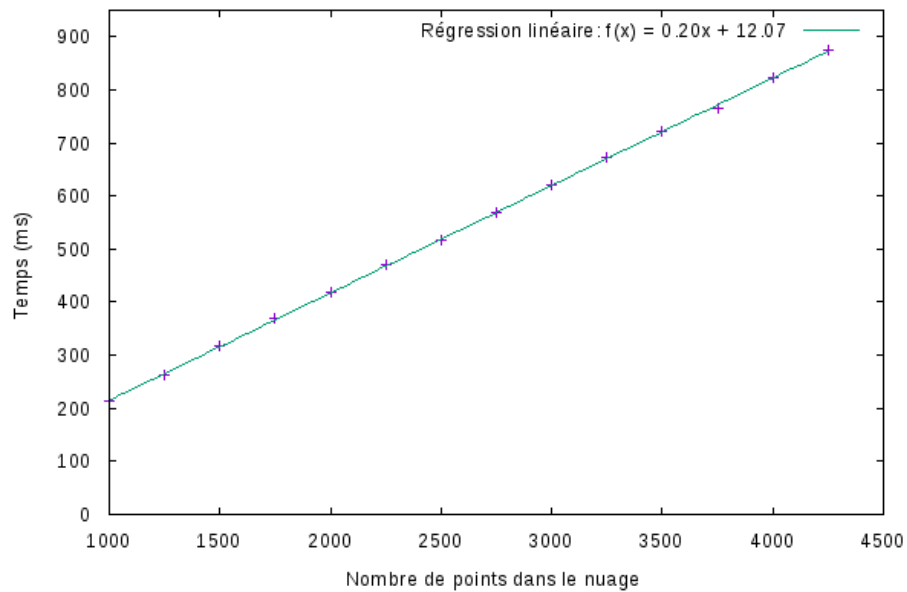


Figure 3: Runtime de S-MIS en fonction de la taille du graphe

Nous constatons ici que, à densité et connexité égales, l'algorithme est linéaire en la taille du nuage de points en entrée.

3.3 Connexité du graphe géométrique

Nous évaluons ici l'impact de la connexité du graphe sur les performances de l'algorithme S-MIS. À tailles de nuage de point égales, nous comparons les impacts de modifications de seuil du graphe géométrique. Nous nous attendons à ce que la performance en temps de l'algorithme varie en fonction directe de ce seuil, puisqu'intuitivement réduire la connexité du graphe réduit non seulement le nombre de noeuds à traiter en rendant certains noeuds orphelins, mais aussi le nombre de composantes indiquant les chemins connexes potentiels que l'algorithme impose de recalculer.

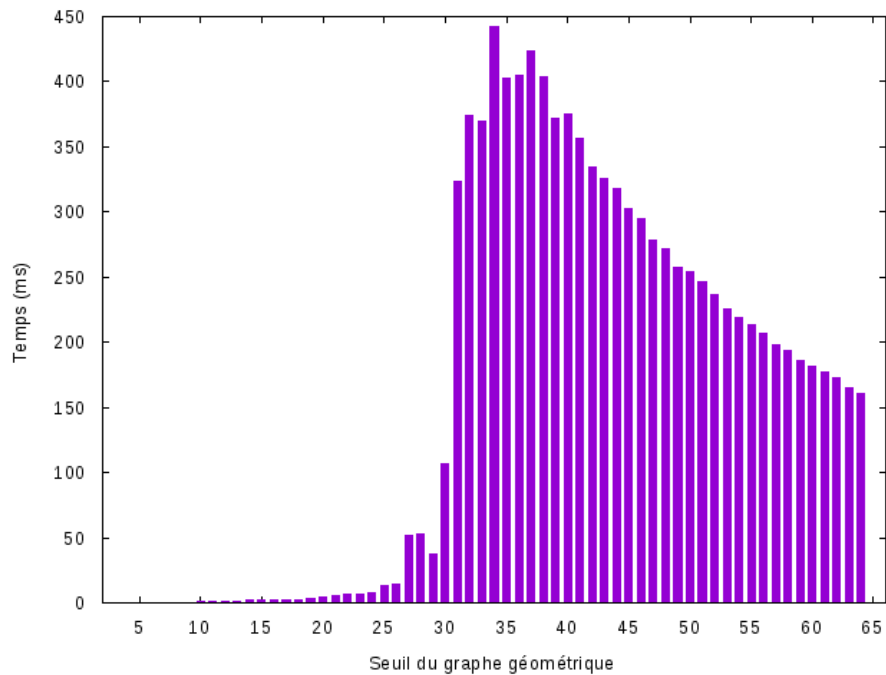


Figure 4: Runtime de S-MIS en fonction de la connexité du graphe, avec une taille de nuage fixe de 800×800

On remarque que le comportement de l'algorithme n'est pas monotone. En effet, un graphe très fortement connexe élimine très vite une forte quantité de noeuds des itérations suivantes de l'algorithme, là où un graphe moins connexe conserve une quantité importante de noeuds pour les itérations suivantes.

4 Références

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, 2005.
2. Cadei M, Cheng MX, Cheng X, Du D-Z. Connected domination in ad hoc wireless networks. In Proceedings of the 6th International Conference on Computer Science and Informatics (CS&I'2002), Durham, NC, USA, March, 2002.
3. Wan P-J, Alzoubi KM, Frieder O. Distributed construction of connected dominating set in wireless ad hoc networks. In Proceedings of IEEE Infocom 2002, New York, NY, USA, June 2002.