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

Introduction

Un ensemble dominant d'un graphe $G = (S, A)$ est un sous-ensemble D de S tel que pour toute arrête $uv \in A$, $u \in D$ 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'algortihme *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 commenceront par présenter l'algorithme, puis présenteront les résultats expérimentaux obtenus, que nous compareront avec d'autres algorithmes de résolution de ce problème.

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

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).

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 apparetenant au MIS, il doit y avoir un et un seul point n'appartenant pas au MIS.

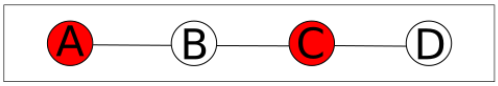

Figure 1: Un MIS valide comme base de l'algorithme S-MIS

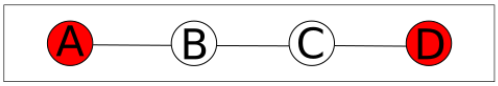

Figure 2: Un MIS invalide comme base de l'algorithme S-MIS

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

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
```

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 : `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 `O(n*m)`), et permet de réaliser l'opération `neihbors` en `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'execution de l'algorithme, et vu qu'un sommet blanc enlevé de la Stack est marqué noir, on en conclue que le nombre d'itéreation 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 `O(m*m)`.
La complexité de la boucle principale est donc `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 extrèmes 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 `O(n*m)`. (Celà 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- Calcul du S-MIS : algorithme de Li et al.

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 apartenant 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 finallement *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 a 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)
```

Analyse de la complexité

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)`.

Complexité temporelle

3- Performances

Méthode d'évalutation des performances

Pour chaque opération, nous répétons un calcul de S-MIS sur des graphes aléatoires 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 ($\leq$ 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.

Densité du graphe géométrique à taille de graphe égale

Nous appelons 'densité du graphe' la densité du nuage de points initial réduit aux noeuds des composantes connexes du graphe géométrique.

La densité du graphe peut être variée de deux façons différentes:

1. par une variation du seuil du graphe géométrique, qui a surtout pour effet de rendre le graphe plus faiblement connexe mais qui n'élimine pas forcément beaucoup de noeuds du graphe
2. par une augmentation de la densité réelle de noeuds, c'est-à-dire par une réduction de la plage des abscisses et ordonnées des points du graphe à taille du nuage de points fixe, ou par une augmentation de la taille du nuage de points à seuil et plage de valeurs fixes.

Nous choisissons de modifier la taille du nuage de points, et nous examinons la qualité de l'ensemble dominant connexe ainsi que le temps moyen pris pour le générer. De manière à ne pas varier la connexité du graphe, nous calculons à partir du seuil de départ un seuil approprié pour chaque valeur présentée ici.

```
![Runtime de S-MIS en fonction de la densite du graphe
   ↪ ](img/densityruntime.png)

![Qualite du S-MIS en fonction de la densite du graphe
   ↪ ](img/densityqual.png)
```

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, mais aussi le nombre de composantes indiquant les chemins connexes potentiels que l'algorithme impose de recalculer.

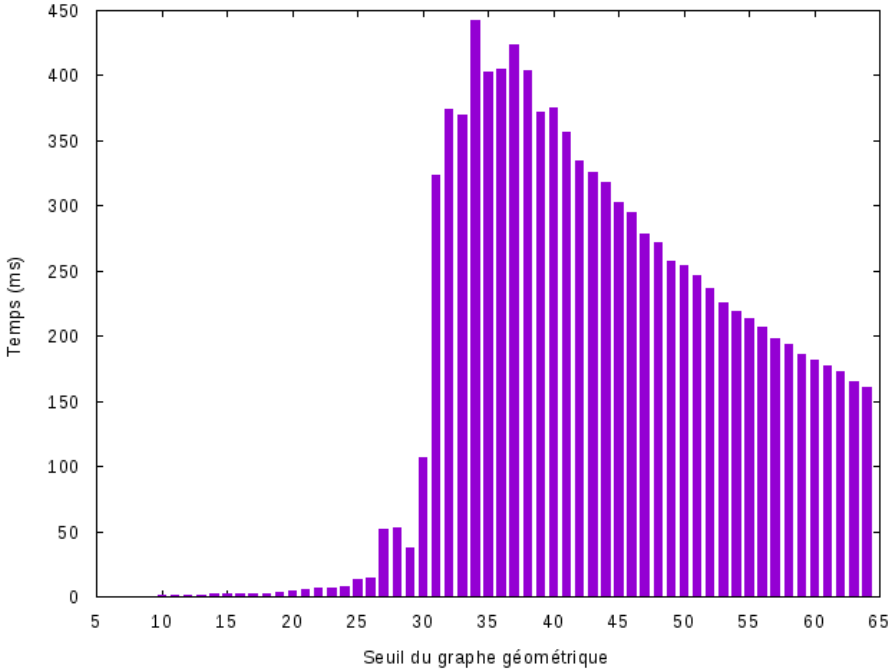

Figure 3: Runtime de S-MIS en fonction de la connexite du graphe

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.

Taille du graphe géométrique à densité égale

Nous varions ici les trois paramètres d'entrée pour évaluer le coût de l'augmentation de la taille du graphe

```
![Runtime de S-MIS en fonction de la connexite du
   ↪ graphe](img/sizeruntime.png)

![Qualite du S-MIS en fonction de la connexite du
   ↪ graphe](img/sizequal.png)
```

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.