Algorithme Li et al. : ensemble dominant connexe glouton

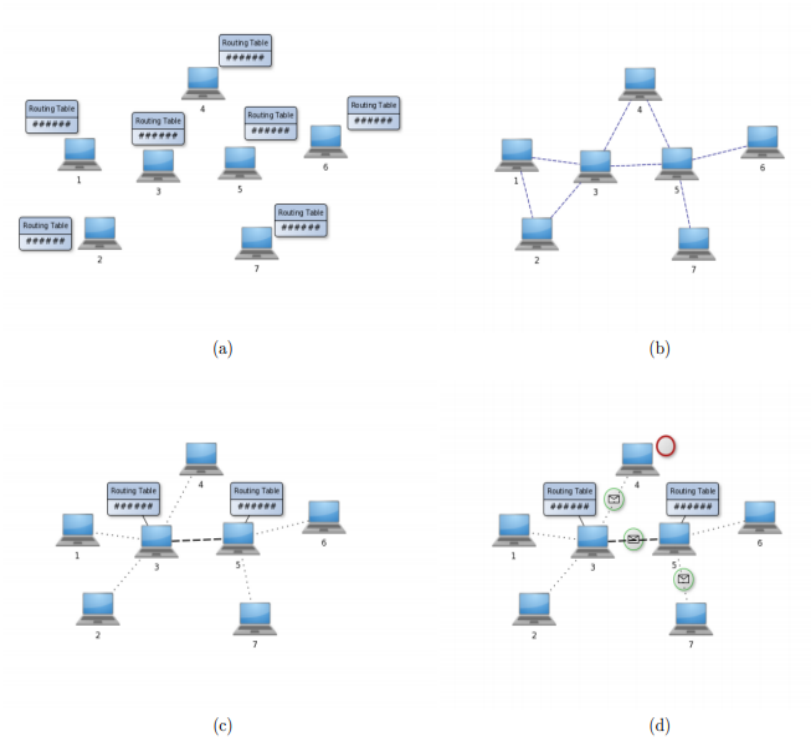
Matthieu Eyraud

3 novembre 2019

1 Introduction

La recherche d'un ensemble dominant dans un graphe a de nombreuses application. L'avantage d'un tel ensemble est la possibilité de donner une information à tous les sommets du graphe en un minimum de temps. Il suffit en effet, grâce à la connexité de l'ensemble dominants, de la passer à tous les sommets de l'ensemble, puis en le parcours d'une arête de faire circuler l'information à l'ensemble du réseau (en utilisant la propriété d'ensemble dominant).

La connexité de cet ensemble a également un avantage pratique dans le routage des réseaux sans fils. Comme il n'y a pas d'infrastructure physique, chaque noeud du réseau doit stocker une table de routage pour pouvoir communiquer avec les autres noeuds. La maintenance et le stockage de ces informations engendrent un coût de communication supplémentaire. En utilisant l'ensemble dominant connexe, on choisit un sous ensemble de noeuds à travers lequel se font toutes les communications sur le réseau (cf. figure ci-dessous [5]).



Néanmoins, le calcul d'un ensemble dominant connexe est un problème NP-complet, il faut donc se contenter d'heuristiques et d'algorithmes approchant la solution optimale.

L'algorithme que nous allons ici étudier est celui de Li et al qui utilise une approche gloutonne pour calculer une bonne approximation de la solution.
On va ensuite proposer une implémentation se basant sur des heuristiques mais sans rapport d'approximation prouvée, pour tester l'efficacité de l'algorithme de Li et al

2 Construction de l'ensemble stable maximal

Le MIS est défini comme suit dans le papier :

Définition 2.0.1 *Soit $G = (V, E)$ un graphe, l'ensemble stable maximal est le plus grand sous-ensemble $S \subseteq V$ tel que le sous-graphe $G[S]$ induit par S ne contient pas d'arêtes.*

Lemme 2.0.1 *Dans tout graphe géométrique, la taille de ses ensembles stables maximaux est majorée par $3.8opt + 1.2$ où opt est la taille de l'ensemble connexe dominant minimum.*

Lemme 2.0.2 *Toute paire de sous-ensembles complémentaires du MIS a exactement une distance de deux sauts.*

Lemme 2.0.1 garantit que le ratio de la solution de cet algorithme par rapport à la solution optimale est bien $4.8 + ln5$.

Si l'algorithme ne respecte pas la propriété du Lemme 2.0.2, alors on l'algorithme de construction de l'ensemble dominant connexe proposé ne renverra pas une solution valide, comme on le verra dans la deuxième partie.

Les auteurs citent le papier [1] pour construire un MIS vérifiant ces deux propriétés. Le fonctionnement de la version non distribuée de l'algorithme est assez simple et l'on va s'appuyer sur un exemple et détailler le déroulement sur une instance.

Le principe général est le suivant :
- `noir` : nœud dominant, appartient au MIS
- `gris` : nœud dominé, voisin d'un nœud dominant, n'appartient pas au MIS
- `blanc` : nœud encore non traité par l'algorithme
- `blanc actif` : état particulier d'un nœud potentiellement prêt à devenir dominant

Faisons tourner l'algorithme sur un exemple simple :

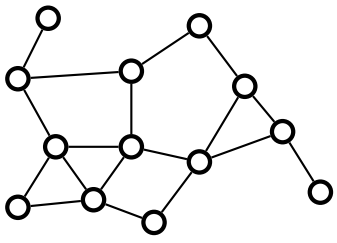


FIGURE 1 – Etat initial

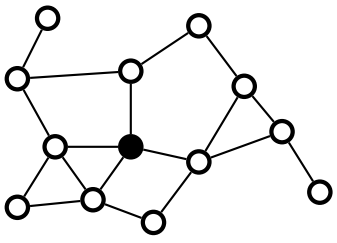


FIGURE 2 – Choix du leader

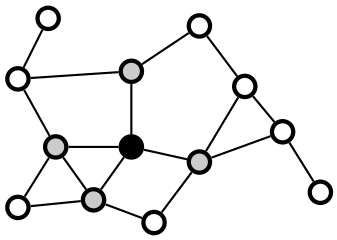


FIGURE 3 – Domination

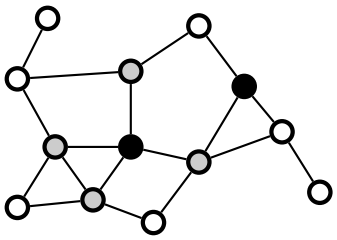


FIGURE 4 – Choix d'un nouveau leader

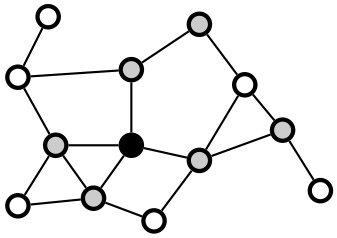


FIGURE 5 – Domination

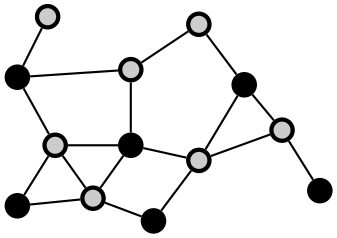


FIGURE 6 – Etat final

Un noeud leader est initialement choisi [2] (on va prendre comme heuristique le choix du noeud de plus haut degré, étant donné qu'il couvrira plus de sommets), que l'on va colorier en noir. Une fois que ce noeud est marqué, on parcourt la liste de ses voisins qui sont **dominés** par le noeud choisi, et donc coloriés en gris [3].
Tous les noeuds blancs voisins des noeuds gris deviennent alors des noeuds **actifs**. On choisit un nouveau leader parmi les noeuds blancs actifs. 4. Ce noeud devient leader, et on réitére le processus jusqu'à marquage de tous les noeuds (en gris ou en noir)
L'ensemble des noeuds noirs constitue le MIS final [6].

Le choix du noeud de plus haut degré initialement est en $O(n+m)$.
Le choix d'un leader se fait parmi tous les noeuds blancs actifs. Supposons qu'ils soient ordonnés dans un tas-max, le choix se fait en $O(1)$.
L'étape de domination consiste à colorier les voisins en gris Supposons qu'ils soient ordonnés dans un tas-max, une fois qu'un noeud est colorié en gris il doit faire savoir à ses voisins blancs qu'ils ont perdu un voisin blanc. Les noeuds blancs actifs sont remontés/descendus dans le tas avec les opérations de `PercolateUp` et `PercolateDown`, en une complexité de $O(log\ b)$, b étant le nombre de noeuds blancs actifs
Parcourir les voisins de voisins se fait en une complexité en $O(\delta^2)$, δ étant le degré maximal. Ce qui fait comme complexité $O(\delta^2 log\ n)$ en prenant en compte la mise à jour du tas.
Le nombre de noeuds blancs actifs admet comme majoration n, et donc la complexité finale est $O(n + m + n * \delta^2 log\ n)$. Cette borne de complexité est une approximation très supérieure à la complexité réelle. En pratique, plus la densité du graphe augmente (et donc δ^2 augmente), moins il y a de noeuds blancs actifs, et donc le nombre d'itérations va être réduit. Au contraire, si le graphe n'est pas dense alors $\delta^2 log\ b$ diminue et la complexité tend plutôt vers $O(n+m)$

3 Construction de l'ensemble dominant connexe

La construction de l'ensemble connexe dominant, à partir du MIS calculé précédemment est décrite dans le papier. Elle se rapporte à un problème de coloration de graphe.

```
compute a MIS satisfying lemmas 1 and 2.
color MIS nodes to black.
color other nodes to grey.
for i = 5; 4; 3; 2 do
  while there exists a grey node adjacent to at least i black nodes in
      different black-blue components do
    change its color from grey to blue;
return all blue nodes.
```

Chaque noeud du graphe a au plus 5 voisins non reliés entre eux, étant donné que l'on est dans un graphe géométrique. En effet, si le nombre de voisin est supérieur à 5, alors il y en aura au moins deux qui seront l'un de l'autre à une distance inférieure au seuil, et ce même si on choisit un espacement maximal entre chaque sommet voisin du noeud.

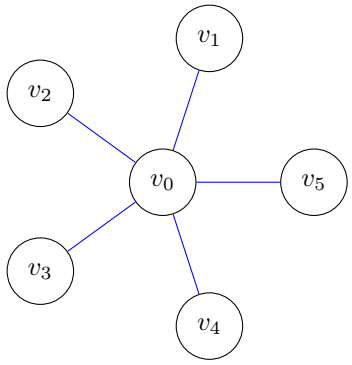


FIGURE 7 – Noeud ayant 5 voisins

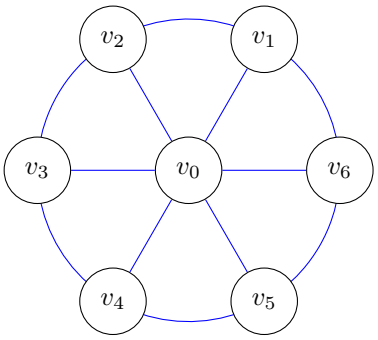


FIGURE 8 – Noeud ayant 6 voisins avec un espacement maximal

On peut donc en déduire naturellement que le nombre de noeuds noirs appartenant à différentes composantes bleues-noires voisin du noeud gris analysé est majoré par 5 (par construction du MIS, deux noeuds noirs ne peuvent être voisins).

L'algorithme se termine à partir du moment où chaque noeud est relié à au plus 1 composante bleue-noire. L'ensemble dominant connexe est alors l'union des noeuds bleus et noir.

FIGURE 9 – MIS sans la propriété des deux sauts

Comme on peut le voir dans la figure ci-dessus, l'algorithme de construction de l'ensemble dominant connexe va terminer sans colorier de noeuds en bleu,

alors que l'ensemble des noeuds noir n'est pas connexe.
D'où l'importance de respecter dans la construction du MIS le lemme donnant la propriété adéquate , à savoir que chaque noeud du MIS est à une distance d'au plus deux sauts un autre noeud du même ensemble.

FIGURE 10 – MIS 2-hops

FIGURE 11 – MIS 2-hops

Dans chacun des cas, l'un des noeuds gris est relié à deux composantes bleues-noires (constituées chacune d'un noeud noir), et l'autre noeud gris est relié à une composante bleue-noire.

L'algorithme va naturellement choisir celui qui relie les composantes bleues-noires entre elles, et le colorier en bleu. Il restera ensuite un unique noeud gris relié à une unique composante bleue-noire, ce qui veut dire que l'on a obtenu un ensemble dominant connexe.

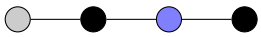


FIGURE 12 – EDC obtenu

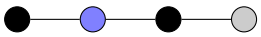


FIGURE 13 – EDC obtenu

On comprend donc que l'objectif de l'algorithme est de parvenir à la connexité de l'ensemble dominant en reliant les composantes bleues noires incrémentalement, et en procédant de manière gloutonne c'est-à-dire en cherchant le noeud maximisant le nombre de composantes qui vont être reliées si on le colorie en bleu.

La complexité principale de l'algorithme est donc dans le calcul des composantes bleues noires. Si l'on peut obtenir les différentes composantes bleues noires et connaître l'appartenance de chaque noeud bleu ou noir à sa composante en temps constant, alors il suffit de parcourir la liste des noeuds gris 4 fois.

Le calcul des composantes bleues noires ne doit se faire qu'à initialisation. Une fois que ces composantes sont calculées, il suffira de fusionner les composantes concernées lorsqu'un noeud est colorié en bleu (auquel cas tous ses voisins noirs appartenant à différentes composantes bleues noires seront reliés).

Rappelons qu'un noeud gris ne peut pas avoir uniquement des voisins bleus (sinon, l'ensemble stable ne serait pas maximal. Une fois qu'il sera colorié en bleu il sera donc nécessairement intégré à une composante pré-existante.

Le calcul initial des composantes se résume à faire un parcours du graphe pour identifier les noeuds appartenant au MIS, les colorier en noir et les rattacher chacun à une composante les contenant eux seul. Il est donc en $O(m)$, m étant le nombre de points dans le MIS (en supposant que le graphe soit représenté sous forme de liste d'adjacence de noeuds coloriés, chaque noeud ayant un indice et le MIS étant un ensemble d'indice).
Pour avoir une fusion des composantes bleues noires en temps constant, on va utiliser une structure de données possédant les opérations $Union$ et $Find$.

La racine de chaque sous-ensemble est appelé représentant de la composante. Pour trouver à quel composante appartient un noeud, on va remonter le chemin des parents jusqu'à la racine. Deux noeuds pointant vers la même racine appartiendront à la même composante.

Quand on a besoin d'identifier la composante à laquelle appartient un noeud `n`, on est obligé de remonter jusqu'à la racine. On en profite alors pour faire pointer `n.parent` vers cette racine. Cela évitera alors à la prochaine recherche de devoir reconstruire la chemin jusqu'à la racine. Cette optimisation est appelée la compression de chemin. Cela profite non seulement au noeud n mais également à tous ceux qui l'ont comme ancêtre (comme parent, ou bien parent de parent etc.). La complexité amortie pour `Find` est constante avec cette optimisation.

`Find` implémente la recherche de la racine avec la *path compression* :

```
function Find(x)
  if x.parent != x
    x.parent := Find(x.parent)
  return x.parent
```

La deuxième optimisation est lorsque l'on fusionne des composantes `c1` et `c2`, on identifie celle qui a le rang le plus petit. Elle devient le représentant des deux composantes. Si les deux composantes ont le même rang, cela signifie qu'elles ont la même profondeur et que inévitablement le rang de l'une d'entre elle va être augmenté.
Cette union par rang permet d'avoir une bonne complexité pour l'opération `Find`, en $O(log\ n)$, n étant le nombre de noeuds dans la structure.

`Union` fait l'union par rang :

```
function Union(x, y)
  xRacine := Find(x)
  yRacine := Find(y)

  // x et y sont dans la meme composante
  if xRacine == yRacine
     return

  // x et y ne sont pas dans la meme composante, on les fusionne
  if xRacine.rang < yRacine.rang
    xRacine.parent := yRacine
  else if xRacine.rang > yRacine.rang
    yRacine.parent := xRacine
  else
    //Choix arbitraire de la racine
    yRacine.parent := xRacine
    xRacine.rank  := xRacine.rang + 1
```

4 Heuristique

4.1 Présentation de l'algorithme

On se base sur une nouvelle heuristique, proposée par Rai, M et Garg, N et Verma, S et Tapaswi, S [4].

```
function MCDS(G(V, E))
   u := MAX-DEGREE(V)
   F := F ∪ {u}
   Q = ∅

   for ∀ x ∈ N{U}
       INSERT(Q,x)

   while |Q| ≠ ∅
       u := REMOVE(Q)
       if G(V - {u}) est connexe
           V := V - {u}
       else
           F := F ∪ {u}
           empiler les voisins de U dans Q non inseres dans Q

   return F
```

L'heursitique démarre avec un noeud racine comme ensemble dominant connexe initial. On choisit le noeud de plus haut degrés, arbitrairement si il y en a plusieurs.
A chaque étape, un noeud est sélectionné dans la file de priorité ;

- Soit il est inclus dans la solution finale
- Soit il est retiré du graphe

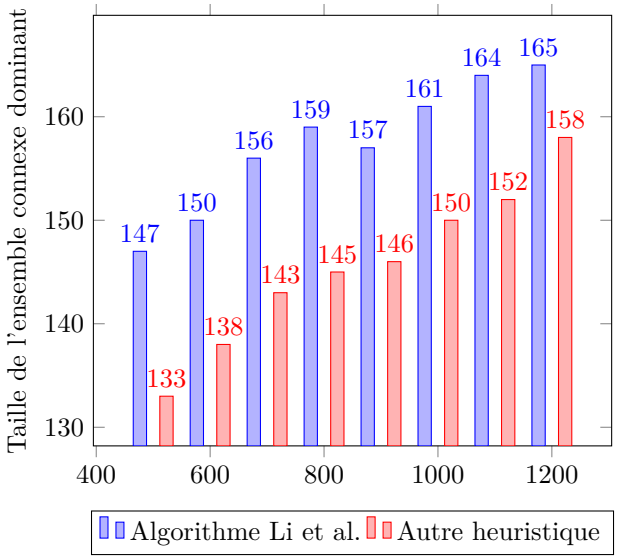
On considére deux types de noeuds : les noeuds fixés et non-fixés. Le retrait des noeuds fixés déconnecterait le graphe et on n'aurait donc pas de solution au problème de l'ensemble dominant **connexe**, ils feront donc partie de cet ensemble.
Les noeuds non fixés peuvent être enlevés si le sous-graphe induit par leur retrait est toujours connexe. A chaque étape de l'algorithme, au moins un noeud du graphe est fixé, ou enlevé, jusqu'à ce que l'algorithme se termine avec la construction d'un CDS.

4.2 Analyse de complexité

Chaque noeud du graphe est empilé une fois. Quand un noeud est dépilé, on vérifie la connexité du graphe $G(V-u)$. Le calcul de connexité a une complexité en $O(V+E)$.
Quand un noeud est effacé du graphe (si le graphe reste connecté), il faut mettre à jour ses voisins (le degré de chaque voisin est décrémenté d'un), ce qui est fait au maximum en $(N-1$ étapes (un noeud a au plus N-1 voisins). Si le graphe n'est pas connecté, alors le noeud est ajouté aux noeuds fixes et donc à la solution et tous ses voisins non insérés sont empilés dans Q, ce qui se fait en une complexité en $O(N-1)$ également.
La complexité de l'algo est donc $O(N(N+E+N-1))$ donc $O(N^2)$.

5 Résultats

On génère des points aléatoirement dans une fenêtre de coordonnées circulaire, puis on retourne la plus grande composante connexe. Pour éviter la dispersion des points, on considérera des graphes de plusieurs centaines de points à minima.



Ce graphe présente la moyenne du nombre des points de l'ensemble dominant connexe pour une dizaine de graphe pour chaque barre, dans un intervalle de 100 autour de la valeur x de la barre (des graphes de 500 à 600 points pour la première barre etc.). On peut voir que l'algorithme de Li et al. retourne des ensembles dominants connexes d'une taille légèrement supérieure en particulier si la taille du graphe est réduite. Cet écart se réduit en augmentant la densité du graphe (qui est dans notre cas corrélée à la taille du graphe, puisque les points sont générés dans une fenêtre donnée).

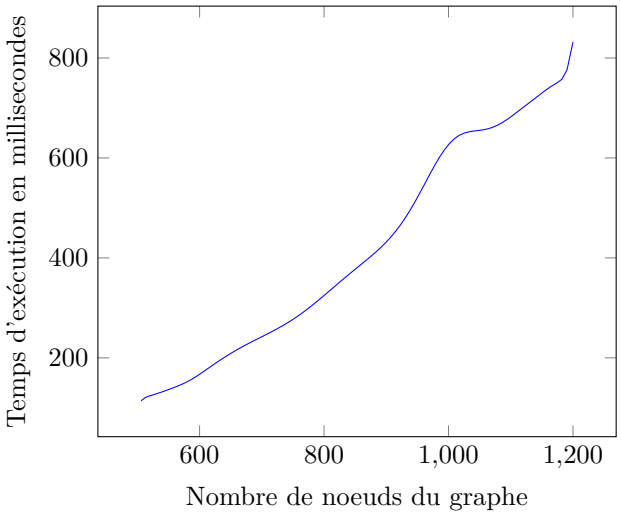


FIGURE 14 – Temps d'exéuction pour Li et al.

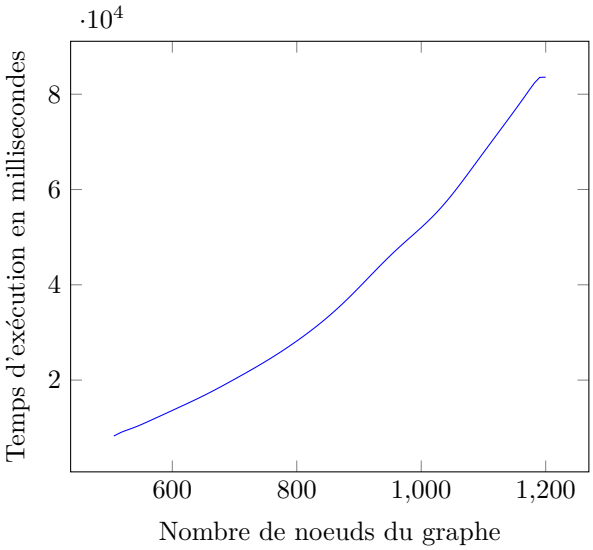


FIGURE 15 – Temps d'exéuction pour l'heuristique

Le $O(n^2)$ se fait ressentir sur le temps d'exécution du calcul de l'ensemble dominant connnexe en utilisant le deuxième algorithme implémenté (cf. échelle des ordonnées sur le deuxième graphe).
Il y a également probablement une constante multiplicative en plus de la complexité quadratique puisque le langage de programmation utilisé n'a pas été le même (Java pour Li et al. Python pour l'autre heuristique, qui offre des facilités de programmation pour la théorie des graphes à travers le package networkx).

Références

[1] Mihaela Cardei, Maggie Xiaoyan Cheng, Xiuzhen Cheng, and Ding-Zhu Du. Connected domination in multihop ad hoc wireless networks. In *JCIS*, pages 251–255, 2002.

[2] Bo Gao, Yuhang Yang, and Huiye Ma. A new distributed approximation algorithm for constructing minimum connected dominating set in wireless ad hoc networks. *International Journal of Communication Systems*, 18(8) :743–762, 2005.

[3] 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. *Wireless Communications and Mobile Computing*, 5(8) :927–932, 2005.

[4] M Rai, N Garg, S Verma, and S Tapaswi. A new heuristic approach for minimum connected dominating set in adhoc wireless networks. In *2009 IEEE International Advance Computing Conference*, pages 284–289. IEEE, 2009.

[5] Sofiane Soualah. Algorithmes heuristiques et exacts pour le probleme de l'ensemble dominant connexe minimum, 2014.

[6] Wikipedia. Graphe de disques — wikipedia, the free encyclopedia. `https://fr.wikipedia.org/wiki/Graphe_de_disques`. [consulté le 15-Oct-2017].

[7] Wikipedia. Maximal independent set — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Maximal_independent_set`. [consulté le 15-Oct-2017].