Analyse d'Algorithmes et Génération Aléatoire :

Rapport sur l'algorithme glouton de [Li, Thai, Wang, Yi, Wan, Du]

Ensemble Dominant Connexe

Axel ZAVIER

3 Novembre 2019

Master 2 STL

SCIENCES SORBONNE UNIVERSITÉ

Sommaire

1 Problème et structures de données utilisées

Etant donné un graphe $G = (V, E)$, le problème Ensemble Dominant Connexe (CDS) consiste à calculer un plus petit sous-ensemble de sommets $D \subseteq V$ tel que pour tout sommet $v \in V$ du graphe est soit un élément de D, soit un voisin d'un élément de D (propriété de dominance), et tel que le sous-graphe $G[D]$ induit par les sommets de D est connexe. Les sommets de D sont appelés sommets dominants tandis que les sommets de $\{ V / D \}$ sont appelés sommets dominés.

La construction d'un CDS minimal est NP-difficile.

Le CDS est utilisé pour la mise en place de backbones virtuels et ainsi supporter et connecter de gros réseaux et trafics Internet notamment.

Dans ce problème, on manipule un graphe géométrique dans un plan 2D défini par un ensemble de points dans le plan appelés sommets, et un seuil sur la distance entre les points : il existe une arête entre deux sommets si et seulement si la distance euclidienne entre les deux sommets est inférieure à ce seuil.

Pour répondre à ce problème, un algorithme glouton 4/3 optimal a été conçu par des chercheurs. Il est effectué en 2 étapes et utilise un marquage de points que nous allons détailler plus tard.

Pour encoder cet algorithme, nous manipulons :

- ArrayList<Point> pour encoder un ensemble de Point
- HashMap<Point Color> pour effectuer le marquage où Color est une énumération
- ArrayList<ArrayList<Point>> pour encoder un ensemble de composants bleu noir

Un composant bleu noir est un ensemble de points induit par des points noirs et bleus.

Nous allons maintenant présenter l'algorithme glouton AlLi.

2 Analyse et présentation de l'algorithme glouton

Le principe de l'algorithme est d'abord de créer un ensemble stable maximal. Un ensemble stable maximal est un ensemble dominant tel que si on retire un point de cet ensemble, il n'est plus dominant. De plus, tous les points de cet ensemble sont reliés entre eux par **exactement deux sauts** (seul un point les sépare).

Pour obtenir cet ensemble, nous implémentons l'algorithme de Cardei, Cheng, Cheng, Du. Au départ, tous les points sont blancs. On détermine un point quelconque comme leader qu'on colore en noir, il va s'agir du premier dominant. On colore tous ses voisins en gris (*). Puis, s'il existe des points blancs voisins d'au moins un point gris, on les « active ». Parmi les points blancs « activés », on colore en noir celui de plus haut degré et on applique l'étape *. On effectue ces étapes jusqu'à ce qu'il n'y ait plus de points blancs. Dans notre implémentation, l'opération la plus couteuse qu'on effectue est de calculer k voisinages en O(n) où n = $|V|$. Cette étape de calcul d'ensemble stable maximal est donc de complexité O(n) * k où k est négligeable donc O(n).

Après avoir construit l'ensemble stable maximal, on le donne en entrée de l'algorithme glouton. L'objectif étant de créer un arbre couvrant, en reliant les points de l'ensemble stable maximal.

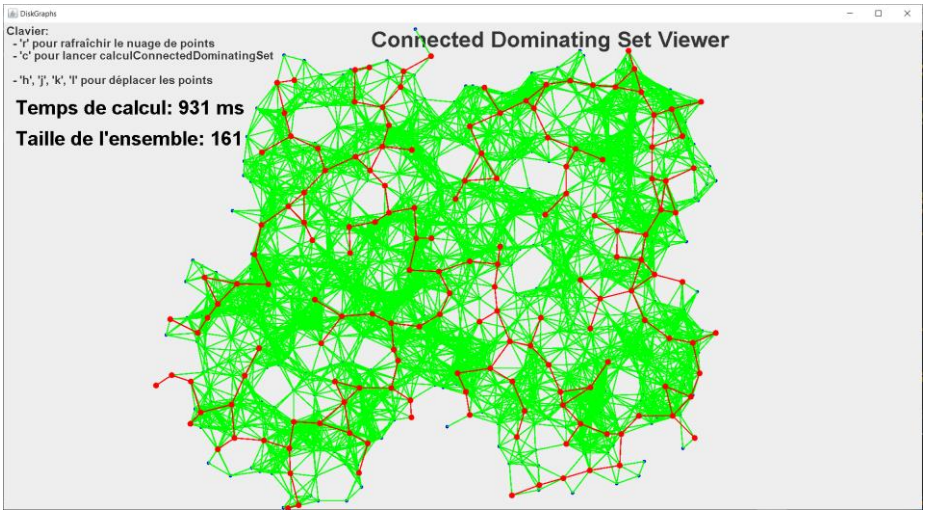
La technique proposée consiste en un marquage de tous les points de l'ensemble stable maximal en noir et de colorer le reste des points en gris. Au départ, chaque point noir est un composant bleu noir.

Puis, pour chaque point gris, on vérifie s'il est voisin d'au moins i=5 composants bleu noir. Pour ce faire, on vérifie si le point gris est voisin de 5 points noirs issus de 5 composants différents. Si c'est le cas, on change sa couleur en bleu, on fusionne tous ses composants voisins en un seul composant et on ajoute le point bleu dans cette fusion pour obtenir un composant englobant. On réitère ces opérations pour i=4,3,2 composants voisins. A la fin de cette recherche, on a coloré en bleu les points connectant les points de l'ensemble stable maximal.

Le résultat du CDS est l'ensemble de points induit par les points noirs et bleus.

Dans notre implémentation, la complexité du calcul du CDS peut être résumé à O(n) + 4* (O(n) * (O(nb composant total * nb points noirs)*2 + O(nb composants voisins du point courant)). Le premier O(n) correspond au parcours des entrées de la table dont la taille est n = $|V|$. En simplifiant, on obtient C = O(n) + 4 * (2*O(n)*m) = O(n)*m où m = nombre de composant total à chaque boucle + nombre de points noirs. Il est donc important de minimiser le nombre de points noirs.

En appliquant cet algorithme sur la première instance du projet, on obtient :



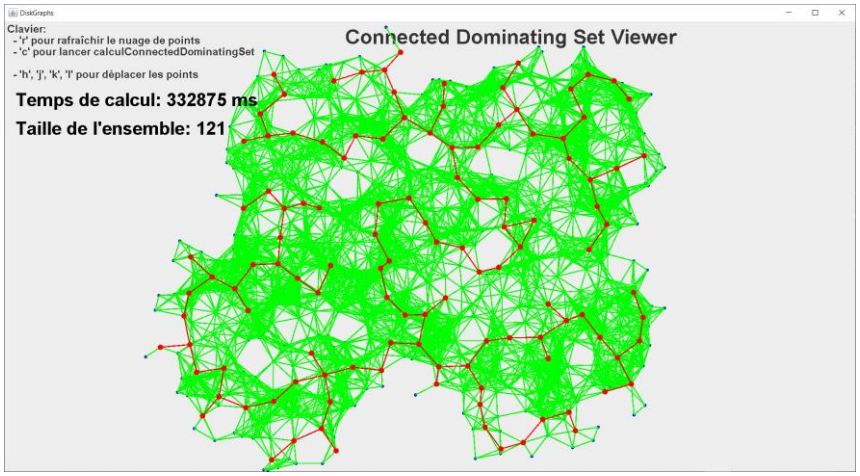
Cette approche étant 4/3 optimale, il est donc possible de l'améliorer.

3 Améliorations possibles

Il existe plusieurs heuristiques d'améliorations. Les plus connues sont sans doute l'heuristique naïve de suppression de points tant que l'ensemble reste connexe et dominant ainsi que la recherche locale. Dans notre implémentation, nous effectuons des recherches locales (deux points par un point) pseudo-aléatoires et entre chaque recherche locale on effectue une permutation pseudo-aléatoire entre un point du CDS et un point en dehors du CDS, toujours en vérifiant les propriétés de dominance et connexité.

Une recherche locale moins naïve pourrait être d'identifier les possibles cycles et de remplacer ces cycles par un ou plusieurs points tel que le nombre de remplaçants soit inférieur au nombre de points constituant le cycle.

En effectuant des LSP (Local Search with Permutation), on obtient :



4 Implémentation autre

Dans cette partie, on construit un CDS à partir du même ensemble stable maximal que pour l'algorithme glouton mais avec un reliage de points différents. On le relie par un arbre de Steiner calculé avec la méthode Kruskal.

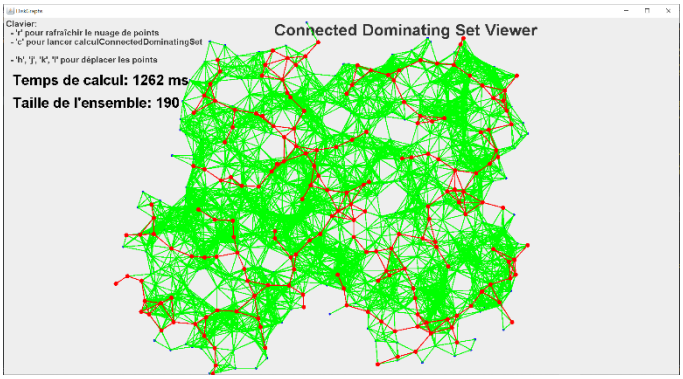
Etape 1 : on construit un graphe K induit par les sommets de l'ensemble stable maximal ainsi que par les arêtes correspondant aux plus courts chemins reliant ces sommets.

Etape 2 : on applique Kruskal au graphe K (on obtient un arbre T)

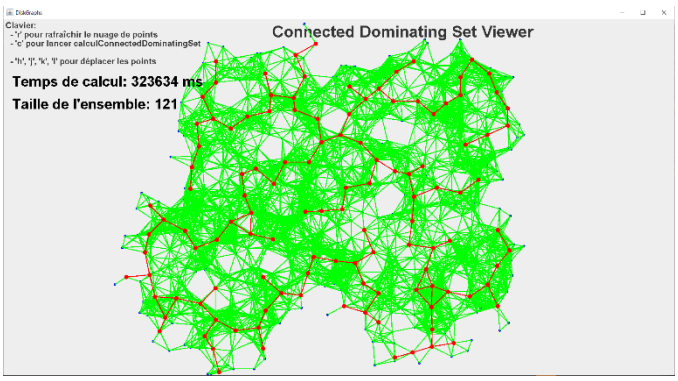
Etape 3 : on traduit T en chemins vers un graphe H

Etape 4 : on applique Kruskal sur H et on retourne le résultat

On obtient un nettement moins bon résultat que par l'algorithme glouton :

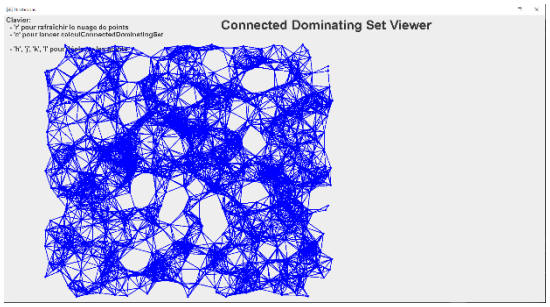


En effectuant le même nombre de LSP que précédemment, on tend vers des scores ressemblants :

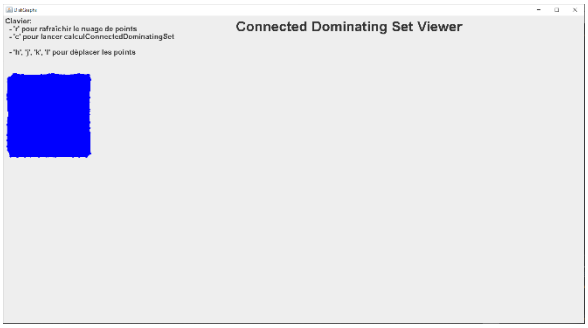


5 Testbeds

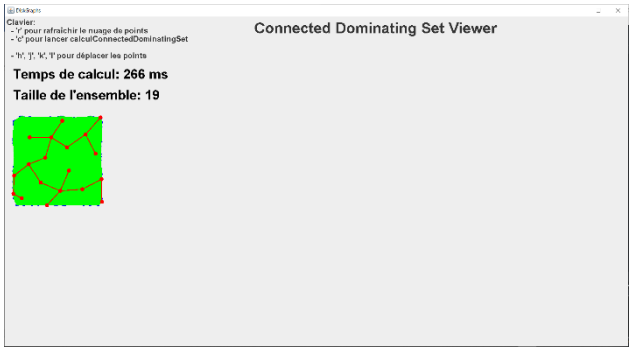
Pour la génération de testbeds on utilise deux approches. La première avec laquelle on a généré 20 instances qui est la moins amusante utilise le générateur pseudo aléatoire de Java avec des bornes de 100 à 900 :



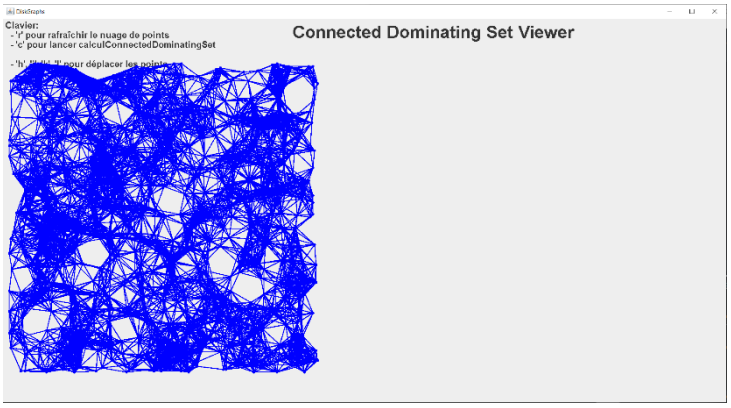
La deuxième utilise l'algorithme de génération du Mersenne Twister, dont le code se trouve dans la classe MersenneTwister. Seulement, les entiers générés sont très grands, on les divise donc par un facteur de 10 000 000. On obtient ainsi :



Avec CDS :



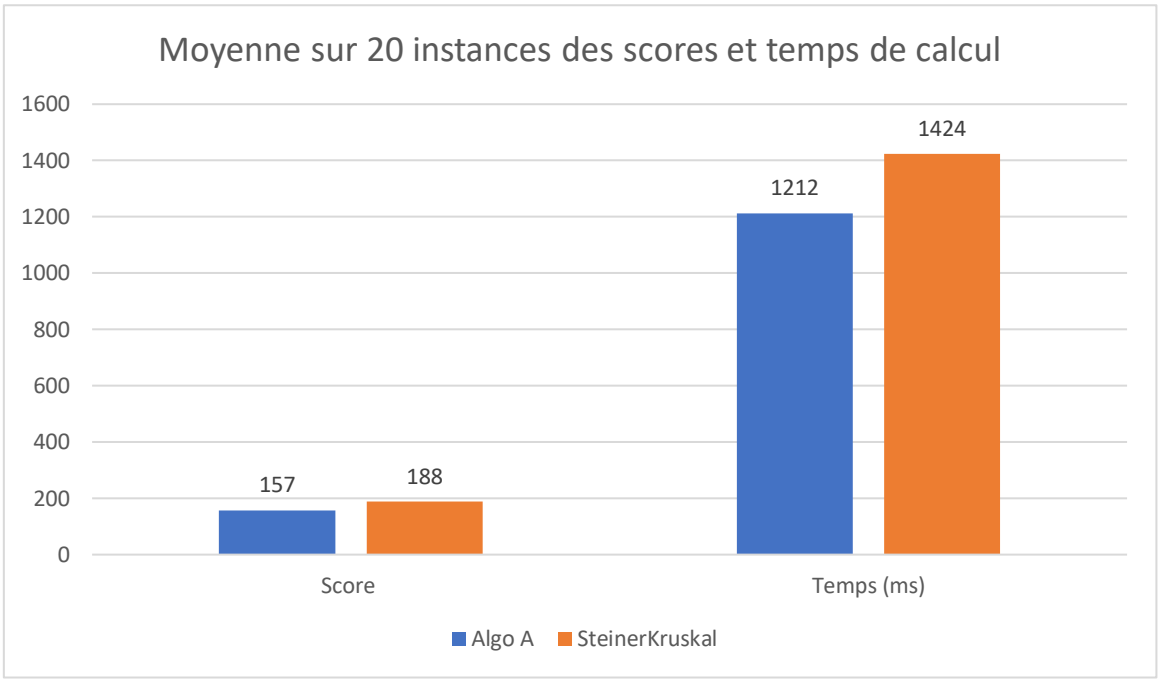
Pour remédier à ce problème on multiplie les entiers générés par 3 :

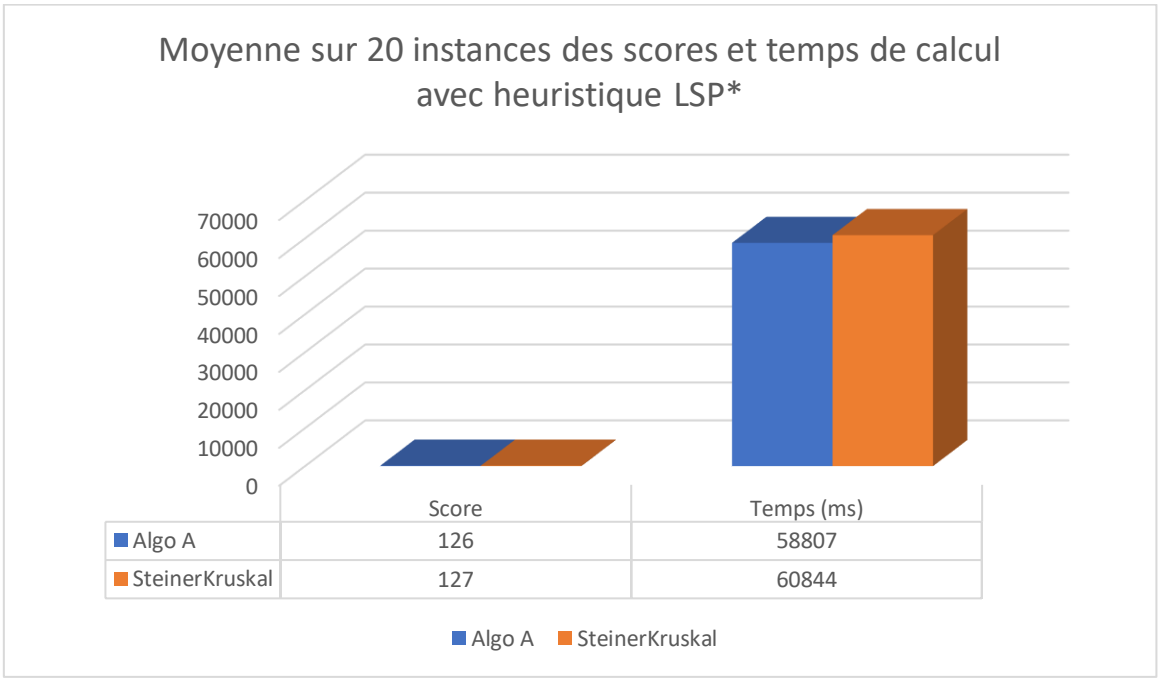


Notre étude expérimentale ci-après a utilisé des testbeds générés avec la première approche pour la simple raison que j'ai implémenté le Mersenne Twister après avoir effectué l'étude expérimentale sur les 20 instances déjà produites avec le générateur de Java…

Nous allons maintenant pouvoir passer à la fameuse et tant attendue étude expérimentale comparative entre les deux approches précédentes.

6 Etude expérimentale





LSP : Local Search Permutation

7 Conclusion

Au vu de notre étude expérimentale, il est clair que l'algorithme glouton proposé par nos chercheurs propose de meilleurs scores et temps de calculs que pour le SteinerKruskal. Un autre avantage est qu'il a été plus simple à implémenter et nécessite moins d'étapes. On remarque que le coloriage de points est une pratique courante et extrêmement dévastatrice pour obtenir de bons résultats.

Malgré l'efficacité de l'approche gloutonne, on a vu qu'il était possible d'améliorer les résultats en sacrifiant du temps de calcul. De plus, peu importe l'algorithme utilisé, en utilisant des heuristiques d'améliorations comme le LSP, on tendra vers les mêmes résultats pour un temps de calcul augmenté, se rapprochant ainsi de l'optimal.