Jules Bobo

Projet Pace 2024

Introduction :

Le but est de minimiser les croisement d'arêtes dans un graph bipartie $G = ((A \cup B, E)$ composé de deux partitions A et B. Les sommets A sont fixés et sont reliés à des sommets B. Le programme a pour but de trouver une permutation des sommets B minimisant les croisement entre les arêtes du graph avec une contrainte de limite de temps de 5min.

Ce programme fonctionne de la manière suivante :

1. Lecture du graph
2. Médiane
3. Barycentre
4. Moyenne Médiane Barycentre
5. Séparation de graph en Composante Connexe CC
6. Médiane sur les CC
7. Barycentre sur CC
8. Moyenne Médiane Barycentre sur CC
9. Recherche local
10. Envoie de la réponse

Si la limite de temps est atteinte, on arrête toutes les fonctions en exécution et on envoie la réponse, le programme s'arrête.

Explication du Programme :

1. **Lecture du graph:**

- On lit le graph dans l'entré standard et on le stock dans deux dictionnaire, de la forme : Sommet A : [liste des sommets B connecter à A]
  Sommet B : [liste des sommets A connecter à B]
  Complexité : O(m)

2. **Méthode de la médiane :**

- On associe à chaque sommet de B la médiane des sommets A auxquels il est connecté. On trie la liste de sommets selon cette nouvelle séquence.
  Complexité O(nlog n)

3. **Méthode du Barycentre**

- On associe à chaque sommet de B la moyenne des sommets A auxquels il est

connecté. On trie la liste de sommets B selon cette nouvelle séquence.
Complexité O(nlogn)

4. **Moyenne Médiane barycentre**

- On utilise les séquences Médiane et barycentre et on associe chaque sommet B à sa moyenne entre de médiane et barycentre. On trie la liste de B selon ce nouvel ordre.
  Complexité O(nlogn)

5. **Séparation en Composante connexe.**

- On sépare le graph en composante connexe avec le DFS en itératif. On récupère alors k composantes connexes stockées dans k graph A:[sommet B] avec k séquence de sommets b associés. Complexité O(n+m)

6. **Application de médiane et barycentre sur les CC**

- On applique les fonctions médianes et barycentres précédentes sur les CC récupérées précédemment et on concatène les séquence de sommets B récupérées.
  Complexité O(k*n*log n)
  On effectue ensuite la moyenne entre médiane et barycentre de nos nouvelles séquence B
  Complexité O(n*logn)

7. **Recherche Local**

- S'il nous reste du temps, on parcourt notre meilleure séquence et on échange les paires d'éléments et on teste si le nombre de croisement est réduit, si ce n'est pas le cas on passe à la paire suivante sinon on remplace notre meilleure séquence.

8. **Sélection de la meilleur séquence de sommets B**

- Après chaque heuristique trouvant une séquence d'élément B on compte les croisements pour garder la meilleure séquence.
  On compte les croisements en O(n logn) avec une méthode semblable à la fonction merge-sort. On sépare notre séquence en logn partions dans lequel on compte les croisements, on rassemble 2 à 2 ces partitions et on ajoute le nombre de croisement entre ces partitions. Pour cela je me suis inspiré d'un post sur un forum en adaptant grandement le code. (voir les sources la fin)

Autre Heuristic :

Dans les heuristic sur le CC on concatène les séquence de gauche à droite trouver or les CC sont probablement imbriquées les unes les autres, il faudrait donc une meilleur méthode.
J'ai donc créé une méthode qui triait les CC selon la moyenne des sommets A dans chaque CC et qui concaténait les séquence B de médiane et barycentre selon ce nouvel ordre mais le résultat était souvent moins B qu'avant j'ai donc désactivé cette fonction.

Avant d'implémenter médiane et barycentre j'avais créé une méthode qui s'occupait des graph particulier. Elle détectait les graphs complets en comparant le nombre d'arêtes et de sommets. Elle créait aussi une séquence donnant 0 croisement si elle existait. En effet, elle triait les sommets de B de façon à créer 0 croisement, ainsi si le cas 0 croisements était possible elle le trouvait en O(n^2). Cependant la fonction barycentre traite ces cas la rendant inutile.

Difficulté rencontré :

J'avais initialement implémenté de DFS en récursif mais la limite de récursion en python posait des problèmes sur les très grand graph car la limite peut être augmentée jusqu'à la valeur limite des int au maximum. J'ai donc dû la transformer en itératif.

Création d'un Solver

La limite de commit sur optil.io étant assez contraignante et ne voulant pas commit mon code sur le github pour le tester j'ai créé un solver testant mon programme sur les 100 instance publiques et le testant avec la librairie comptant les croisement sur optil.io pour être sûr de mon résultat. Il a la possibilité de sauvegarder mes résultats pour les comparer avec mes autres versions de mon programme. J'ai aussi commit mon solver sur le github.

Source :

Les instructions sur le moodle du projets
Les documents des gagnants de la Pace 2024 mais à moindre échelle car j'ai très peu compris leur contenu.

pour le comptage des croisements
https://www.cp.eng.chula.ac.th/~prabhas//teaching/algo/algo2008/count-inv.htm