CAMPORA Marion
GREAUX Thomas

Rapport miniprojet 2

Nous avons commencé par créer une structure nous permettant de pouvoir traiter les 3 différents algorithmes. Pour cela, nous avons une classe Graph qui nous permet gérer toutes les actions relatives au graphe, comme l'ajout ou la suppression d'arêtes et le choix d'une arête aléatoire.
Nous avons aussi créer une classe nous permettant de générer aléatoirement des exemples, en prenant en compte le fait que les noeux doivent bien être reliées entre eux, ainsi qu'une classe permettant de lire les fichiers d'exemples pour en tirer le graphe correspondant.
Pour finir, nous avons implémenté les algorithmes. Le premier algorithme (classique()), prend en paramètre un graphe dont les poids sur les arêtes sont tous de 1. Ensuite, tant que l'on a plus de 2 sommets, on choisit au hasard deux sommets que l'on contractent. Le deuxième algorithme (karger_stein()) et le troisième (karger_stein_persalized()), prennent le même graphe d'entrée que le premier. Tant que l'on a plus d'un certain nombre de sommets (resp. $n*2^{-0.5}$ et n/1.1), on choisit au hasard deux noeuds du graphe et on les contractent. On effectue alors un certain nombre de fois (resp. 2 et 3) l'algorithme classique() sur les sommets qu'il reste.

Le premier algorithme correspond à l'algorithme de Karger simple. Nous arrivons à une complexité de O(n4) ou O(n log3 n). Cependant, la probabilité d'obtenir la coupe minimum est de 2/[n(n-1)].
Le deuxième algorithme correspond à l'algorithme de Karger-Stein. L'implémentation de cet algorithme engendre une complexité de $O(n^2 \log 3\ n)$, il est donc bien plus complexe que le premier. Cependant, la probabilité d'obtenir la coupe minimum devient beaucoup plus importante : 4/[n(n-1)].
Pour le troisième algorithme nous avons choisi 1.1 pour la valeur de a et 3 pour la valeur de b. Etant donné que $1.1 < \sqrt{2}$, nous avons plus de chance que pour l'algorithme de base (soit plus de 50%) de rester dans la coupe minimale à la fin des premières itérations. Afin d'améliorer notre chance de trouver la coupe minimale, nous faisons trois appels récursifs au lieu de 2. Cependant, ces implémentations ont un coup. En effet, nous faisons plus d'appels récursifs et le nombre d'itérations à l'intérieur de celui-ci est plus important. Ainsi, nous obtenons une complexité encore plus importante. Cet implémentation nous permet d'obtenir une meilleur taux de réussite contre une complexité plus élevée.

Pour conclure, l'algorithme le plus adapté pour un gain de temps et de complexité est l'algorithme classique, tandis que le mieux adapté pour trouver la coupe minimale est l'algorithme personnalisé. Entre ces trois algorithmes, c'est l'algorithme de karger-stein qui possède le meilleur rapport complexité / taux de réussite.