Aix*Marseille université
Initiative d'excellence
Année 2019-20

Diplôme Inter-Universitaire
Enseigner l'informatique au lycée

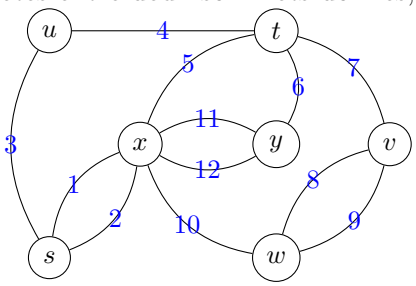
Bloc 5 : TP 8

Aléatoire en algorithmique

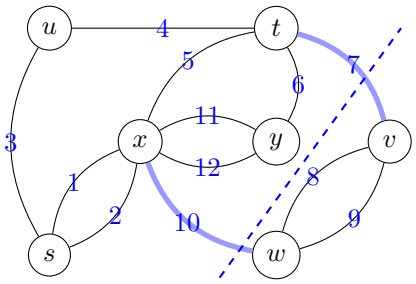
L'objectif de cette séance[1] est d'étudier un algorithme de Monte-Carlo résolvant le problème de la coupe minimale. Ce problème peut être résolu par un algorithme polynomial déterministe en cherchant un flot maximum (par exemple l'algorithme de Ford-Fulkerson). Ici, nous allons explorer une technique se basant sur un générateur aléatoire, introduite par Karger en 1993. On obtient ainsi un algorithme qui fournit un bon résultat avec une probabilité que l'on peut estimer.

1 Définitions

On considère un multigraphe non orienté $G = (S, A, t)$, où S est l'ensemble des sommets et A est l'ensemble des arêtes et la fonction t associe à chaque arête a l'ensemble $t(a) = \{x, y\}$ qui identifie ses deux extrémités, deux sommets distincts. C'est donc comme un graphe non orienté, sauf qu'il peut y avoir un nombre quelconque d'arêtes entre deux sommets donnés, d'où le préfixe "multi".

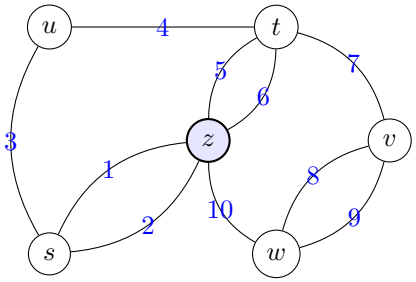


Une *coupe* de G est un ensemble $C \subset A$ d'arêtes de G tel que, si on enlève ces arêtes, le graphe partiel $G \setminus C = (S, A \setminus C)$ a 2 composantes connexes. Le problème de la coupe minimale est de trouver une coupe ayant un nombre minimum d'arêtes.



On définit la *contraction* d'un multigraphe $G = (S, A, t)$ par une arête a d'extrémités x et y comme étant le multigraphe $G/a = (S', M', t')$ dans lequel :

1. on remplace les sommets x et y par un nouveau sommet z, donc $S' = S \setminus \{x, y\} \cup \{z\}$;
2. on supprime les arêtes entre x et y, donc $A' = A \setminus \{a \mid t(a) = \{x, y\}\}$;
3. on relie à z les arêtes qui dans G étaient reliées à x ou à y mais pas aux deux.



1. Cette séance est adaptée d'un TD du cours d'algorithmique de Jean-Marc Vincent à l'Université Grenoble Alpes.

2 Génération d'une coupe aléatoire

On se donne l'algorithme suivant :

```
def coupe_aléatoire(G):
    for i in range(nb_sommets(G) - 2):
        a = choisir_arête(G)              # choix uniforme parmi les arêtes
        G = contraction(G, a)
    return G
```

Si on donne en entrée un multigraphe G, on obtient en sortie un graphe à 2 sommets, les arêtes entre ces sommets forment une coupe du multigraphe G.

1. Expliquer le fonctionnement de l'algorithme `coupe_aléatoire` sur l'exemple de la figure.
2. Montrer que cet algorithme produit bien une coupe.
3. À partir d'exemples, exprimer votre intuition que la coupe produite ne doit pas être trop mauvaise.

3 Propriétés de la coupe aléatoire

Pour analyser la complexité de cet algorithme, on estime la probabilité qu'il fournisse une coupe minimale. Le calcul exact étant difficile, on va minorer cette probabilité et montrer qu'elle est suffisante pour avoir un algorithme efficace.

Soit k la taille minimale d'une coupe de G. On choisit une coupe minimale C_m (il en existe forcément une mais il n'y a pas de raison qu'elle soit unique).

1. Donner un minorant du nombre d'arêtes du graphe G.
2. Calculer un minorant de la probabilité de ne pas tomber sur une arête de C_m au premier tirage aléatoire.
3. Donner un minorant du nombre d'arêtes du multi-graphe graphe G' après la première contraction.
4. Minorer la probabilité de ne pas tomber sur une des arêtes de C_m au second tirage.
5. Procéder par récurrence pour minorer la probabilité de ne pas tomber sur une des arêtes de C_m au i-ème tirage.
6. Calculer un minorant de la probabilité que `coupe_aléatoire` produise la coupe minimale C_m.

4 Conception d'un algorithme qui garantit une probabilité d'erreur

On ne produit pas forcément la meilleure coupe à tous les coups. Pour traiter ce problème, on va procéder de façon naïve : on répète le tirage un certain nombre de fois et on garde le meilleur résultat. Cela revient à produire un échantillon de taille suffisante N.

```
def répétition_coupe(G, N):
    Cmin = coupe_aléatoire(G)
    for i in range(N - 1):
        C = coupe_aléatoire(G)
        if taille(C) < taille(Cmin):
            Cmin = C
    return Cmin
```

1. Minorer la probabilité, en fonction de N, que `répétition_coupe(G,N)` fournisse le bon résultat.
2. Montrer que pour une taille d'échantillon de l'ordre de $N = n^2$, la probabilité d'obtenir une coupe minimale est minorée par $1 - 1/e$.
3. On se donne une probabilité α et un graphe de taille n. Donner la taille d'échantillon N telle que la probabilité d'avoir produit une coupe minimale soit supérieure à α. Donner la valeur de la taille de l'échantillon pour $\alpha = 0,99$ et $n = 100$.

5 Expérimentation en Python

Sur le cours Ametice, vous trouverez une implémentation Python de l'algorithme de Karger, ainsi que de l'algorithme non-randomisé utilisant un algorithme de recherche de flot maximum, provenant de `http://goatleaps.xyz/programming/kargers-algorithm.html`. Tester cet algorithme sur différents graphes (qu'on initialise à l'aide d'une matrice d'adjacence) que vous pourrez représenter graphiquement à l'aide de la méthode `draw` utilisant les bibliothèque `NetworkX` et `matplotlib`. En particulier, comparer le résultat de l'algorithme de Karger et l'exécution d'une seule coupe aléatoire (la méthode `karger_trial` qui demande ensuite à réinitialiser le graphe à l'aide de la méthode `restore_original`). Comparer également les temps de calcul de l'algorithme de Karger et de l'algorithme non-randomisé (qu'on peut lancer à l'aide de la méthode `mincut('edmonds_karp')`).