INFO-F-203 - Rapport

Projet 1

Yahya Bakkali
Matricule : 445166

Maxime Hauwaert
Matricule : 461714

Date : Novembre 2018

Table des matières

1 Introduction générale

Ce projet a pour but de mettre en pratique des concepts sur les graphes vus au cours d'algorithmique 2 pour une meilleure compréhension et maîtrise de ceux-ci.

2 Sous-arbre de poids maximum

2.1 Introduction

Dans ce problème nous manipulons des arbres constitués de nœuds ayant un poids. Le problème consiste à transformer un arbre $T = (V, E)$ en arbre $T' = (V', E')$ de façon à maximiser la fonction

$$w(V') = \sum_{v \in V'} w(v)$$

2.2 Choix d'implémentation

Pour stocker l'arbre nous avons décidé d'utiliser une classe nommée "Tree". Chaque nœud a un nom, un poids et une liste de ses enfants. Nous avons décidé de ne pas modifier l'arbre de départ mais de créer une liste qui contiendra le nom de tous les nœuds qui forme le sous-arbre de poids maximum.

2.3 Algorithme

Algorithme 1 maxContribution

```
Require: liste nœuds_à_désactiver
1: poids_total = nœud.poids
2: for chaque enfant du nœud do
3:    if enfant.maxContribution()<= 0 then
4:       Ajouter enfant à nœuds_à_désactiver
5:    else
6:       Ajouter enfant.maxContribution() à poids_total
7:    end if
8: end for
9: return poids_total
```

La complexité de cet algorithme est de $O(n)$ car il parcourt chaque nœud de l'arbre, $O(n)$ et à chaque nœud on additionne deux

valeurs en $O(1)$, on compare deux valeurs en $O(1)$ et on ajoute à chaque fois[1] le nom du noeud dans une liste donc $O(1)$. Donc la complexité finale est de $O(n)$.

2.4 Arbres aléatoires

Cette génération aléatoire d'arbres a une très bonne distribution. Tous les arbres sont possibles. Il y a de 1 à n nœuds qui composeront l'arbre, 'n' étant 15 dans ce projet. Chaque nœud choisira tout simplement de qui il veut être l'enfant parmi les nœuds déja placés.

3 Les hypergraphes et hypertrees

3.1 Introduction

Dans cette partie nous manipulons des hyper-graphes et des hyper-graphes duals. Les outils utiles pour travailler sur les hyper-graphes sont les graphes d'incidence et les graphes primals. On manipule plusieurs concepts comme les graphes acycliques, les hyper-arbres, les cliques maximales, la chordalité et la couverture exacte.

3.2 Choix d'implémentation

Pour stocker l'hyper-graphe nous avons décidé d'utiliser une classe nommée "Hypergraph". Chaque objet de cette classe a un ensemble de ses sommets, un dictionnaire d'hyper-arêtes, une matrice d'incidence, un dictionnaire de sommets liés et un graphe primal. Nous avons aussi décidé d'utiliser une classe "PrimalGraph" pour une meilleure visibilité de nos différents algorithmes. Chaque objet de cette classe a un ensemble de ses sommets, un dictionnaire de ses hyper-arêtes, le nombre de sommets qu'il a et une liste de ses arêtes.

[1] Au pire des cas

3.3 Algorithmes

Algorithme 2 find_cliques

Require: R : {nœuds d'une clique maximale}, P : {nœuds possibles dans une clique maximale}, X : {nœuds exclus}

```
1:  if P et X sont vides then
2:    if la clique R est de taille >= 2 then
3:      Ajouter R a la liste des cliques
4:    end if
5:  else
6:    pivot = élement aléatoire de l'ensemble P ∪ X
7:    for chaque sommet S dans l'ensemble P \ {sommets liés au pivot} do
8:      newP = P ∩ {sommets liés à S}
9:      newR = R ∪ {S}
10:     newX = X ∩ {sommets liés à S}
11:     find_cliques(newP,newR,newX)
12:     P = P \ {S}
13:     X = X ∪ {S}
14:   end for
15: end if
```

[1] Tout graphe à n sommets a au maximum $3^{n/3}$ cliques maximales, et le temps d'exécution le plus défavorable de l'algorithme de Bron-Kerbosch (avec une stratégie pivot qui minimise le nombre d'appels récursifs effectués à chaque étape) est $O(3^{n/3})$, correspondant à cette limite.

Algorithme 3 is_chordal

```
1: unnumbered = ensemble des sommets du graphe
2: s = sommet choisi aléatoirement dans unnumbered
3: unnumbered = unnumbered \ {s}
4: numbered = {s}
5: while unnumbered ! = {∅} do
6:    Vertex = le sommet de unumbered qui a le plus de connections aux sommets de numbered
7:    unnumbered = unnumbered - Vertex
8:    numbered = numbered + Vertex
9:    clique_wanna_be = {sommets liés à Vertex} ∩ numbered
10:   subGraph = Un sous-graphe induit des sommets appartenant à clique_wanna_be
11:   if le subGraph n'est pas complet then
12:      return False
13:   end if
14: end while
15: return True
```

[2] Au début, nous créons l'ensemble des sommets du graphe, "unnumbered", $O(N)$, choisissons un sommet arbitraire d'unnumbered $O(N)$, enlevons ce sommet d'unnumbered $O(1)$, créons l'ensemble des sommets, "numbered", contenant ce sommet $O(1)$. Ensuite, tant qu'unnumbered n'est pas vide $O(N)$ nous chercherons le sommet "Vertex" dans unnumbered qui a le plus de connexions aux sommets de numbered $O(S*N*W) = O((N/2)^2*N) = O(N^3/4)$ parce que la cardinalité de l'ensemble $\{S+W\}$ est toujours égale à N, après nous supprimons ce sommet d'unnumbered en l'ajoutant à numbered $O(1)$ puis nous créons un ensemble qui contiendra l'intersection entre numbered et l'ensemble des sommets liés au sommet "Vertex" $O(\min(W, A))$ et le sous-graphe induit a partir de cet ensemble $O(N * \min(W, A) * \min(V, A))$. Ce qui fait en final une complexite de $O(N * \max(N^3/4, \min(W, A), N * \min(W, A) * \min(N, A)))$

A : nombre de sommets liés au sommet "Vertex"
N : nombre de sommets du graphe
S : nombre de sommets dans "unnumbered"
W : nombre de sommets dans "numbered"

Algorithme 4 Algorithm_X

Require: Matrice

Faire une copie de la matrice et exécuter l'algorithme sur cette matrice
Choisir la colonne C contenant un minimum de 1
L = L'ensemble des lignes tel que $Matrice_{l,c} = 1$, $\forall l \in L$
for chaque ligne l de L **do**
  columnslist = []
  rowslist = []
  Ajouter la ligne à la solution partielle
  for chaque colonne j de la matrice **do**
    if $Matrice_{l,j} = 1$ **then**
      for chaque ligne i de la matrice **do**
        if $Matrice_{i,j} = 1$ **then**
          Ajouter la ligne i à rowlist
        end if
      end for
      Ajouter la colonne j de la matrice à columnslist
    end if
  end for
  Supprimer les lignes et les colonnes de la matrice présentes dans rowslist et columnslist
  if la matrice n'est pas vide **then**
    if toutes les colonnes de la matrice ont au moins un 1 **then**
      Répéter cet algorithme de façon récursive sur la matrice réduite
    end if
  else
    Ajouter la solution à l'ensemble des solutions
  end if
  Supprimer la ligne de la solution partielle
  Réutiliser la matrice de départ
end for

[3] Au début on fait une copie de la matrice sur laquelle on va appliquer l'algorithme $O(M*N)$, après on cherche les colonnes avec un minimum de 1 $O(M*N)$ ensuite on trouve les lignes $Matrice_{L,C} = 1$ $O(N)$. Après pour chaque ligne des lignes trouvées précédemment on trouve les lignes et les colonnes à supprimer $O(M*N)$. On les supprime de la matrice $O(M*N)$. Si la matrice réduite n'est pas vide et que la colonne avec un minimum de 1 n'est pas 0, on répète cet algorithme sur cette matrice réduite de façon récursive ce qui donne une complexité non déterministe. Ça veut dire que l'algorihtme peut présenter des comportements différents sur des passages différents d'une matrice à l'autre réduite. Sinon la matrice est vide, on trouve

une couverture exacte parmi les autres.

References

[1] Bron–Kerbosch algorithm
`https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm`

[2] networkx.algorithms.chordal
`https://networkx.github.io/documentation/stable/_modules/networkx/algorithms/chordal.html`

[3] Knuth's Algorithm X
`https://en.wikipedia.org/wiki/Knuth%27s_Algorithm_X`
NP (complexity)
`https://en.wikipedia.org/wiki/NP_(complexity)`

3.4 Hypergraphes aléatoires

On crée d'abord l'ensemble de n sommets[2], n pouvant aller de 1 à 15, ensuite on détermine le nombre m d'hyper-arêtes[3], m pouvant aller de 0 à $\min(2^n - 1, 15)$.

4 Interface graphique

Pour la première partie du projet, on affiche à l'écran tous les noeuds de l'arbre de départ. Les noeuds faisant partie du sous-arbre de poids maximum seront colorés en rouge et les autres seront colorés en gris.

Pour la deuxième partie, on affiche à l'écran l'hypergraphe dual sous deux formes, son graphe primal et son graphe d'incidence, et on affiche si oui ou non c'est un hyper-arbre. (+ terminal)

[2] 15 étant une limite imposée
[3] 15 étant une limite arbitraire

5 Librairies utilisées

5.1 Numpy

C'est une librairie très utile dans ce projet pour l'utilisation d'opérations mathématiques telles que les fonctions sinus/cosinus, etc ainsi que dans la manipulation de l'aléatoire.

5.2 Matplotlib

C'est une librairie assez utile dans ce projet pour l'affichage d'objets mathématiques en 2D tels que des cercles, des lignes, etc.

5.3 Copy

C'est une librairie contenant la fonction "deepcopy" permettant de copier l'intégralité d'un objet sans qu'il n'y ait de liens entre l'ancien et le nouvel objet.

6 Conclusion

En plus de la simple mise en pratique de certains concepts sur les graphes, ce projet nous a permis de développer nos compétences de travail en groupe.