Laamiri Achraf, El Achouchi Iliass

Rapport : Projet d'Algorithmique 2

Université Libre de Bruxelles
Science informatique
info-F-203

Année accadémique 2018-2019

1 Arbre

1.1 Implémentations

1.1.1 Entrée et Sortie

Tout ce qui est en rapport avec les sorties de ce projet se fera via les librairies `networkx` et `matplotlib`. Elles offrent un choix assez large quant à l'affichage et la gestion de graphes, ce qui facilite grandement la partie visualisation du projet.
Par contre les entrées pour cette partie du projet se feront via l'algorithme de génération aléatoire d'arbre `random_tree`.

1.1.2 Affichage

L'exécution de cette partie du projet ouvre une seule et unique fenêtre grâce à la méthode `show` de `matplotlib.pyplot`. Elle affiche sur la partie supérieure de la fenêtre l'arbre généré, et juste en dessous l'arbre réduit, conformément à l'énoncé du projet. Cette vue permet de comparer assez facilement l'arbre généré et l'arbre réduit.

Les nœuds de l'arbre sont nommés, ils sont enracinés en « r » et ont chacun une valeur comprise entre -5 et 5 afin de rester raisonnable dans la génération d'arbre aléatoire.

Enfin, d'un point de vue purement esthétique, un algorithme qui permet de "balancer" l'arbre a été implémenté, mais il n'est malheureusement pas de nous [1].

L'objet Tree est un objet récursif, ou chaque fils d'un arbre est lui-même un arbre. L'algorithme demandé, `max_subtree`, a été implémenté sous forme de méthode de la classe `Tree`. Il permet d'agir directement sur un objet `Tree`.

L'algorithme fonctionne comme cela : Pour chaque fils d'un nœud, il observe si la somme complète de l'arbre (calculée avec la méthode `get_subSum` détaillée plus bas) est inférieure ou égale à 0. Dans ce cas, il supprime le nœud et ses enfants s'il y en a. Le résultat sera donc une somme ou l'arbre sera positive ou rien du tout. Dans le cas ou il n'y a pas de sous-arbre positif de poids positif, il n'y a que l'arbre principal qui est affiché. Le paramètre G de la méthode permet simplement de gérer l'affichage de `networkx`, l'algorithme supprime le nœud du graphe `networkx` au même moment où il supprime le nœud qui fait partie de l'arbre.

La méthode `get_subSum` est très importante. En effet, elle permet de faire la condition de suppression d'un nœud dans la méthode `max_subtree`. Le parcours est similaire à `max_subtree`, on parcourt chaque fils d'un nœud et on ajoute à

1. https ://stackoverflow.com/questions/29586520/can-one-get-hierarchical-graphs-from-networkx-with-python-3

une certaine somme la valeur des nœuds enfants. On récupère ainsi la sous somme totale d'un sous-arbre.

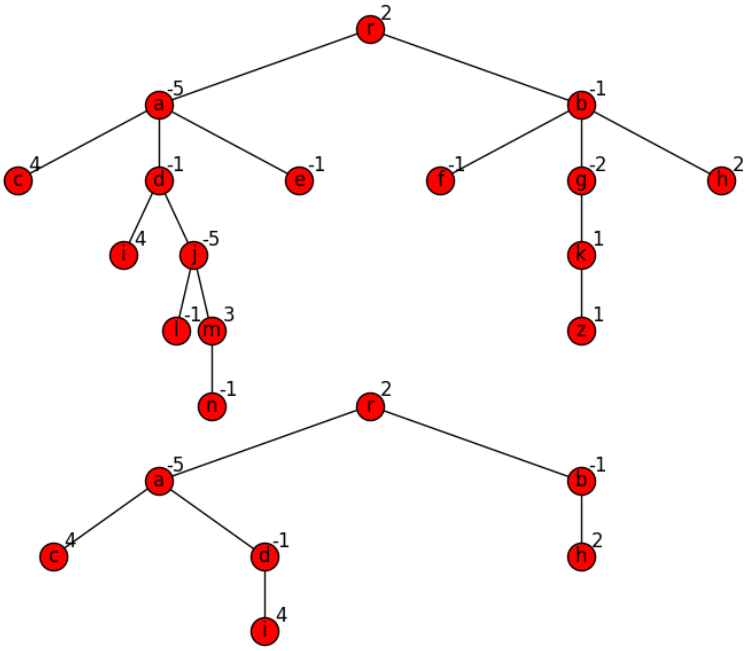


FIGURE 1 – Arbre de l'énoncé et son homologue maximisé

1.1.3 Fonctions et méthodes

1.2 Complexité

2 Hypergraphe

2.1 Implémentations

2.1.1 Entrée et Sortie

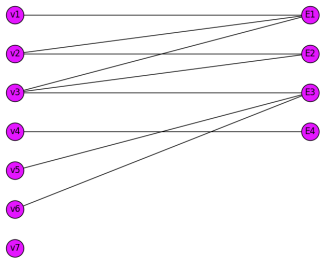
En ce qui concerne l'entrée, nous avons opté pour une matrice d'incidence car il était plus simple de schématiser les arrêtes par une valeur booléenne. De plus, il était facile, via la librairie `Numpy` et sa méthode , de générer des matrices aléatoires, ainsi que pour l'affichage qui sera décrit ce dessous.

2.1.2 Affichage

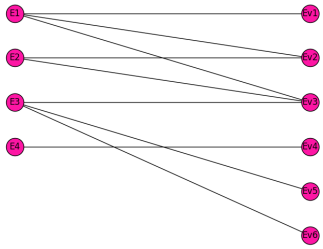
Nous avons décidé d'afficher le graphe d'incidence, via la matrice d'incidence, le graphe dual, via une simple transposée de la matrice d'incidence et enfin le

graphe primal du graphe dual. De cette façon, on pouvais plus facilement afficher toutes les informations du graphe et de les comparer.

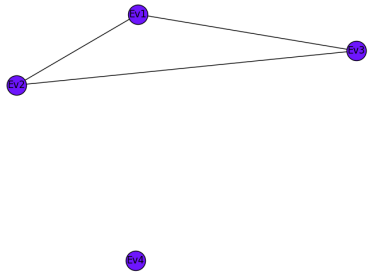
Nous avons aussi décidé d'afficher si oui ou non l'hypergraphe est un hypertree. La matrice d'incidence sera aussi montrée.



(a) Graphe d'incidence.



(b) Graphe dual.



(c) Graphe primal.

FIGURE 2 – Affichage des différents graphes.

2.1.3 Fonctions et méthodes

En ce qui concerne le fonctionnements des méthodes et fonctions de notre implémentation, nous avons essayé de respecter un maximum la mécanique d'encapsulation, via l'utilisation de plusieurs accesseurs et mutateurs dans notre classe `Hypergraph`.

Un objet de cette classe prend comme paramètre une matrice d'incidence, et via cette-dernière, va créer sa transposer. Avec ces 2 matrices, on va pouvoir créer les graphes d'incidence, dual et primal.

Pour pouvoir vérifier si un graphe est un hypertree ou non, nous devons d'abords vérifier si il est cordal. C'est ce que fait exactement la méthode `is_chordal`

de networkx. Elle prend en paramètre un graphe et retourne un booléen si oui ou non le graphe est cordal.

Si cette condition est vérifié, on va analyser chaque clique maximale donnée par la méthode networkx `find_cliques` dans la méthode `checkClique`. Cette dernière va avant tout créer un dictionnaire des hyperarrêtes avec leurs sommets respectifs, pour ensuite comparer chaque clique maximale avec les sommets des hyperarrêtes. Si toutes les cliques sont présentes dans les hyperarrêtes, alors la méthode renvoie `True`, et `False` si non.

2.2 Complexité

Table des matières