Rapport de stage au LaCIM

Adrien Pavao

Été 2017

Contents

1 Introduction

Ce stage à Montréal m'a été proposé par Alexandre Blondin-Massé, dont le contact m'a été transmis par Nicolas Thiery. Je tiens à les remercier chaleureusement. Je suis parti avec Thomas Foltête, un camarade de classe. L'objectif pour moi était de mieux comprendre le travail des chercheurs en combinatoire et de consolider mon niveau en mathématiques et en informatique.

J'ai écrit ce rapport dans le but de garder une trace de mon stage au Laboratoire de Combinatoire et d'Informatique Mathématiques (LaCIM) à l'UQAM. J'y résume ce sur quoi j'ai travaillé et ce que j'ai appris. J'ai par moment utilisé ce document au cours du stage pour prendre des notes et ainsi me souvenir des nouvelles notions apprises. J'y raconte également mon expérience dans ce laboratoire à Montréal.

A mon arrivée au Lacim, j'ai rencontré Alexandre Blondin-Massé, Elise Vandomme et Francesco Dolce. Nous avons discuté des différents problèmes qui pourraient faire l'objet du stage. Je devais donc commencer par me renseigner sur chaque problème avant de choisir celui sur lequel j'allais me fixer davantage. Une partie du travail des chercheurs consistent à déterminer sur quel sujet placer leurs efforts, à se documenter ou encore à participer à des conférences.

La majorité du code produit est en SageMath. Les différents fichiers que nous allons voir au cours de ce rapport sont dans le repository github suivant : *github.com/didayolo/fully-leafed-subtrees*.

2 Recherche documentaire

Voici quelques explications sur les problèmes qui ont retenus notre attention. Vous trouverez en bas de page des liens vers des documents relatifs aux problèmes. Dans le cas contraire, les documents concernés se situent sans doute dans le dossier *documents* du repository Github.

2.1 Conjecture 4/3

La conjecture 4/3 est une conjecture proposée concernant les sous-mots antipalindromiques sur l'alphabet binaire. [1] [2]

- **Antipalindrome :** Un mot a_1, ..., a_l sur l'alphabet binaire -, + est antipalindromique si $a_i = -a_l + 1 - i$ pour chaque $i \in [1, l]$.
- **La conjecture :** Soit un mot circulaire w sur un alphabet binaire -, +, avec n '-' et n '+'. W a un sous-mot linéaire de taille au moins $\frac{4}{3}n$ qui est anti-palindromique.

[1] http://ojs.statsbiblioteket.dk/index.php/brics/article/download/20073/17699
[2] https://cermics.enpc.fr/ meuniefr/OpenProblems.html

2.2 Codes couvrants

Pour expliquer le problème avec un exemple concret, on cherche à couvrir des surfaces pour des téléphones à l'aide d'antennes. On étudie ce problème sur des grilles discrètes. Ils ont déjà été étudiés dans les grilles carrées [3] [4] [5] , hexagonales et triangulaires [6] , mais pas encore dans la grille du roi.

- **Grille du roi :** Il s'agit de la grille dans laquelle sont liées les cases adjacentes accessibles avec un mouvement similaire à celui du roi dans un jeu d'échecs. Chaque case à 8 voisins : 4 adjacents et 4 en diagonale.

2.3 Sous-arbre induit le plus feuillu

Ce problème consiste à chercher le sous-arbre induit ayant le plus de feuilles dans un graphe. Une autre façon de voir le problème est de chercher le nombre maximal de feuilles réalisées par un sous-arbre induit à n sommets dans un graphe donné. C'est ce problème qui a le plus retenu mon attention, c'est pourquoi je l'exposerais plus en détail dans la section suivante.

3 Sous-arbre induit le plus feuillu

Penchons-nous plus en détail sur ce problème et sur des pistes de résolution.

3.1 Lexique

Quelques définitions utiles :

- **Arbre :** Un arbre est un graphe non orienté dans lequel deux sommets quelconques sont reliés par un unique chemin (acyclique). Une feuille est un sommet de degré 1 dans un arbre.
- **Largeur arborescente :** La largeur d'arbre est une façon de savoir si, intuitivement un graphe est proche d'un arbre.
- **Sous-arbre induit :** Un sous-graphe induit est un sous-graphe défini par un sous ensemble de sommets.
  Formellement, H est un sous-graphe induit de G si, pour tout couple (x, y) de sommets de H, x est connecté à y dans H si et seulement si x est connecté à y dans G. Autre formulation de la condition : l'ensemble des arêtes de H correspond à l'ensemble des arêtes de G incidentes à deux sommets de H.
  Vulgairement, on choisit les sommets du sous-graphe et on met les arêtes

[3] http://www.sciencedirect.com/science/article/pii/S0012365X02007446
[4] http://www.discuss.wmie.uz.zgora.pl/php/discuss3.php?ip=&url=bwww_praca&nId=2454&nIdCzasopisma=402&nIdSesji=-1
[5] http://math.nsc.ru/ puzynina/3_pc3.ps
[6] http://link.springer.com/article/10.1134/S0037446606010101

telles qu'elles étaient dans le graphe d'origine, sauf celles à qui il manque un sommet bien-sûr.
Un sous-arbre induit est un sous-graphe induit qui est un arbre.

- **Spanning tree :** Un spanning tree d'un graphe à n sommets est un sous-ensemble de n-1 arrêtes qui forme un arbre.
- **Leaf number :** Nombre maximal de feuilles d'un spanning tree d'un graphe.
- **Pleinement feuillu :** Se dit d'un sous-arbre dont le nombre de feuilles est maximal.
- **Cycle space :** Dans un graphe non orienté, l'ensemble de ses sous-graphes eulériens. Autrement dit, l'ensemble de ces cycles.
- **Graphe eulérien :** Un graphe connexe est eulérien si et seulement si chacun de ses sommets est incident à un nombre pair d'arêtes. Cette propriété est équivalente au fait que l'on peut "parcourir" le graphe en partant d'un sommet quelconque et en empruntant exactement une fois chaque arête pour revenir au sommet de départ, il admet donc un cycle eulérien. Un tel graphe a alors la propriété qu'il correspond à un dessin qu'on peut tracer sans lever le crayon.
- **Cycle basis :** La base est un ensemble permettant de reconstituer le cycle space en combinant ses elements ensemble par différence symmétrique.
- **Différence symétrique :** Une opération tanani

.

3.2 Programmation linéaire

Pour commencer, j'ai étudié un sous-problème plus simple à prendre en main : La recherche du sous-arbre induit de taille k le plus feuillu dans un arbre donné.

Figure 1: Graphe de départ

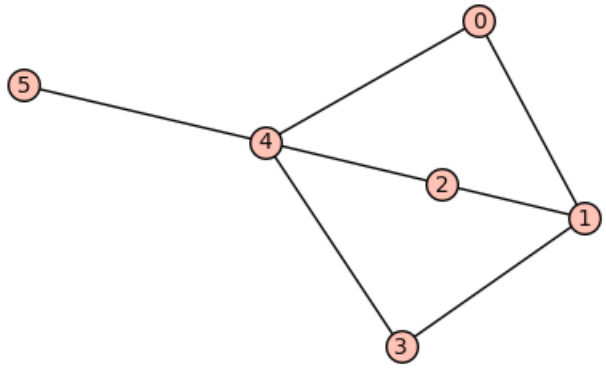
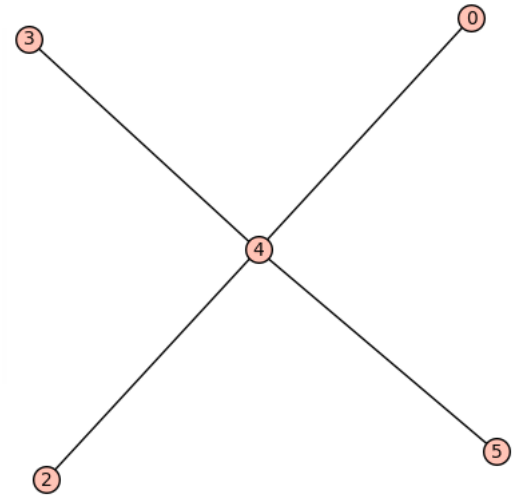


Figure 2: Sous-arbre induit le plus feuillu



J'ai réalisé une série de programme linéaire dans Sage répondant aux problèmes suivants (les fichiers cités sont dans le dossier linear_programs) :

- Arbre le plus feuillu dans un arbre *(leafed_tree_in_tree.sage)*
- Arbre le plus feuillu dans un graphe *(leafed_tree.sage)*
- Forêt la plus feuillu dans un graphe *(leafed_forest.sage)*
- Plus grand arbre dans un graphe *(max_tree.sage)*

- Plus grande forêt dans un graphe *(max_forest.sage)*

Voici un exemple de définition de programme linéaire (arbre le plus feuillu dans un arbre) :

- **Variables :**

$$x_{i,j} = \begin{cases} 0, & \text{Absence d'arête entre les sommets i et j dans le sous-arbre} \\ 1, & \text{Présence d'arête entre les sommets i et j dans le sous-arbre} \end{cases}$$

$$y_i = \begin{cases} 0, & \text{Absence du sommet i dans le sous-arbre} \\ 1, & \text{Présence du sommet i dans le sous-arbre} \end{cases}$$

$$f_i = \begin{cases} 0, & \text{Le sommet i n'est pas une feuille} \\ 1, & \text{Le sommet i est une feuille} \end{cases}$$

- **Fonction objectif :**

  On souhaite maximiser la fonction suivante :

$$f() = \sum_{0}^{i} f_i$$

- **Contraintes :**

  On impose la taille du sous-arbre :

$$\sum_{0}^{i} y_i = k$$

  On vérifie que le nombre d'arêtes est égal au nombre de sommets - 1 :

$$\sum x_{i,j} = k - 1$$

  Présence d'une arête :

$$x_{i,j} <= \frac{y_i + y_j}{2}$$

  Une feuille est un sommet (mais pas forcément l'inverse) :

$$f_i <= y_i$$

  Une feuille est un sommet de degré 1 :

$$f_i <= 1 + \frac{1 - d}{m}$$

  avec m le degré maximal du graphe.

Les autres problèmes ont nécessité des contraintes plus lourdes, notamment pour garantir l'acyclicité du graphe. Pour cela il faut créer une contrainte pour chaque cycle du graphe de départ. La programmation linéaire n'est donc pas vraiment adapté à la résolution de ce problème. Il était cependant assez intéressant de faire ce travail de traduction du problème.

3.3 Pour aller plus loin

Pour aller plus loin, on pourrait chercher des algorithmes de complexité polynomiale qui répondent au problème pour une classe de graphe donnée. La question en suspens est la suivante : peut-on obtenir un algorithme polynomial pour les graphes de largeur arborescente fixée ?

4 Tree-in-the-box

4.1 Présentation

A partir du problème précédent, Elise m'a entrainé vers l'étude des hypercubes, à laquelle j'ai pris goût. Le problème est celui de la recherche du plus grand sous-arbre induit dans l'hypercube de dimension donnée. C'est de loin celui sur lequel j'ai le plus travaillé. La première question sur laquelle je me suis penché consistait à confirmer ou infirmer une construction proposée par Elise, permettant d'obtenir un arbre de taille maximale dans l'hypercube. Il s'est avéré que cette construction fonctionnait jusqu'au rang 6, ensuite il n'était pas possible d'aller plus loin. Cette question m'a demandé beaucoup de dessins, de brouillons, et m'a conduit à une meilleure compréhension et représentation mentale de ce graphe particulier.

Figure 3: Hypercube de dimension 4

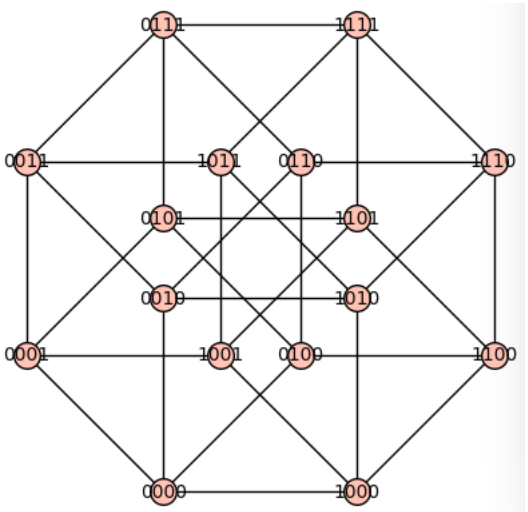
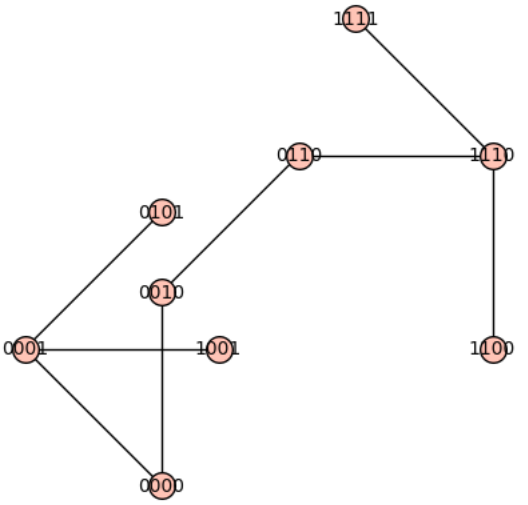


Figure 4: Un sous-arbre induit de taille maximale



J'ai décidé de reporter mes résultats et de réunir ceux déjà existants pour les différents problèmes sur l'hypercube dans un tableur partagé en ligne[7].

[7] https://docs.google.com/spreadsheets/d/1teoYyh9cheFuDEXUULDdZ0B53d8HfsvQHydPW6SpUhw/editgid=0

Voici une façon de définir l'**hypercube** avec de la théorie du langage : Dans un hypercube Q_n, chaque sommet porte une étiquette de longueur n sur un alphabet $A = 0, 1$, et deux sommets sont adjacents si leurs étiquettes ne diffèrent que d'un symbole.[8]

Cette définition est importante car elle permet beaucoup de simplification dans les algorithmes concernant cette famille de graphes.

- **Conjecture :** Le plus grand sous-arbre induit dans Q_n est de taille :

$$2^{n-1} + 1$$

Pour chacune des méthodes que je vais décrire ensuite, j'ai codé une version qui renvoie des arbres, et une version qui renvoie des forêts. Pour les explications nous nous placerons dans le contexte de la recherche d'arbre. Les fichiers concernés sont dans la racine du repository sur Github (*tree_box.sage, tree_box_patterns.sage, forest_box.sage....* Les fonctions à exécuter contiennent généralement "solve" dans leurs appellations et prennent en paramètre la dimension n pour laquelle on souhaite les solutions. La structure de données renvoyée est souvent un itérateur. On peut ainsi obtenir toutes les solutions si on le souhaite, ou bien n'en calculer qu'une seule pour aller plus vite.

4.2 Méthode brute

La solution naïve, facile à implémenter mais très inefficace en terme de complexité. L'algorithme est simple : On prend tous les sous-ensembles de sommets du graphe, on construit le sous-graphe associé à ces sommets. Si le sous-graphe est un arbre, on le renvoie. Avec cette méthode, on est assuré de renvoyer toutes les solutions du problème, cependant il est clair que cela demandera vite beaucoup trop de temps lorsque l'on augmentera le nombre de dimensions de l'hypercube.

4.3 Méthode backtracking

Une solution plus efficace, mais pas la meilleure. L'idée est simple, on ajoute les sommets un par un et à chaque itération on vérifie que le sous-graphe obtenu est un arbre. Si le sous-graphe est un arbre, on continue d'ajouter des sommets. Si ce n'est plus un arbre, on enlève le dernier sommet (on utilise une pile) et on en essaye un autre. On évite ainsi d'aller trop loin dans la recherche lorsque les sommets sélectionnés ne permettront pas d'obtenir un arbre. Ainsi, on renvoie toutes les solutions et on perd beaucoup moins de temps sur les sous-graphes qui ne sont pas des solutions.
Remarque : Si les nouveaux sommets sélectionnés sont toujours des voisins du précédent, alors l'algorithme résout le problème snake-in-the-box (voir *snake_box.sage*).

[8] https://fr.wikipedia.org/wiki/Hypercube_(graphe)

4.4 Méthode avec patterns

Cette méthode consiste à assembler des forêts ensembles pour obtenir un arbre. Il est possible de la décliner de plusieurs façons et de l'améliorer. C'est selon moi l'approche la plus intéressante parmis celles que je présente dans ce rapport. Dans la suite de l'explication, j'emploierai le mot "cube" pour désigner Q_3.

- On peut diviser Q_n (avec $n > 3$) en $2^(n-3)$ cubes. On va ensuite sélectionner des forêts dans chacun de ces cubes (de taille 4, sauf la première de taille 5 qu'on appellera la racine). L'objectif est de déterminer un assemblage de ses forêts formant un arbre. J'ai d'abord choisi ces valeurs en m'appuyant sur la conjecture selon laquelle le plus grand sous-arbre induit dans Q_n est de taille $2^(k-1)+1$.
- On sélectionne donc la moitie des sommets dans chaque cube sauf un qui en aura la moitié + 1 (la racine)
- On utilise ensuite la définition de l'hypercube avec les mots binaires pour réassembler les cubes ensembles. On leur ajoute un préfixe (tous les mots binaires selon k)
- Une fois ceci fait, on vérifie si le graphe obtenu est un arbre. Si c'est le cas, on le renvoi.
- On se rend donc compte que cet assemblage se fait de façon brute. Il permet d'optimiser grandement la recherche d'arbre, cependant cela reste une méthode brute. On peut donc faire l'équivalent avec **backtracking**, comme pour les algorithmes précédents.

  L'idée est la suivante : On sélectionne une racine, puis des patterns dans les cubes, un par un. Tant que le graphe obtenu est un arbre on continue, sinon on enlève le dernier pattern sélectionné et on en essaie un autre.

Note : Joindre un schéma pourrait apporter de la clarté à l'explication.

Pour la suite, une piste intéressante serait la généralisation de cette méthode, ou bien l'utilisation de brique de base de dimension plus grande que 3, etc.

5 SageMath

SageMath est un logiciel libre de mathématiques sous licence GPL. Il combine la puissance de nombreux programmes libres dans une interface commune basée sur le langage de programmation Python[9]. J'ai beaucoup utilisé le logiciel Sage au cours de ce stage. Chaque mercredi, une après-midi Sage était organisée. Un chercheur ou un étudiant y présentait une utilisation du logiciel puis chacun s'entraidait dans son travail avec Sage, afin de s'améliorer et d'approfondir son utilisation du logiciel.
J'ai préparé deux présentations :

[9] http://www.sagemath.org/fr/

- Une sur les graphes.
- Une sur la programmation linéaire.

Cela m'a permis de réviser ces notions, car pour expliquer il est nécessaire de maîtriser. Ces présentations m'ont également permis de pratiquer l'oral devant un public, la gestion du projecteur et du tableau pour rendre la présentation dynamique, les questions pour l'audience, l'utilisation de notebook à compléter durant la séance. C'était une expérience enrichissante et il n'est pas toujours aisé d'avoir l'occasion de s'exercer pour acquérir ces compétences.

6 Conclusion

En conclusion, ce stage dans un laboratoire de combinatoire m'a fait découvrir des aspects très intéressants et très motivants de la recherche à la frontière des mathématiques et de l'informatique, ou chaque discipline est l'outil de l'autre. Je suis très curieux des futures avancées dans les problèmes que j'ai étudié, principalement concernant les hypercubes. Je tiens à nouveau à remercier mon tuteur et l'équipe du LaCIM qui ont été très accueillants et toujours disponibles pour m'aider lorsque j'en avais besoin. J'espère avoir à nouveau l'occasion de travailler avec eux.