Rapport OPTI2 : MBVST

Thibaut Milhaud & Jawhar Zamouri

6 janvier 2021

1 Introduction

L'objectif de ce projet était de comparer deux méthodes de résolution du problème NP complet MBVST. Ce problème vise à minimiser le nombre de sommets de branchements dans un arbre couvrant. On trouve des application à ce problème dans le domaine du routage optique où ces sommets de branchements sont des routeurs coûteux et nuisibles à la qualité du signal.

2 Structure de graphe

Soit un graphe $G = (V, E)$, dans la suite, on note $n = |V|$ et $m = |E|$.

3 Génération de graphes aléatoires

L'objectif était ici de générer des graphes aléatoires connexes avec $m = n - 1 + 3\sqrt{n}$. Pour y parvenir nous avons utilisé l'algorithme suivant.

Algorithm 1 Algorithme de génération aléatoire de graphe

1: **function** RandGraph(n)
2: $r = n - 1 + 3\sqrt{n}$
3: $G \leftarrow$ emptyGraph(n) ▷ Un graphe avec n sommets mais pas d'arêtes
4: $O \leftarrow \{1, \dots, n-1\}$
5: $I \leftarrow \{n\}$
6: **while** $O \neq \emptyset$ **do** ▷ Construction d'une base connexe
7: $v \leftarrow$ random_pop(O)
8: $v_2 \leftarrow$ élément aléatoire de I
9: $E \leftarrow E \cup (v, v_2)$
10: $I \leftarrow I \cup v$
11: $r - -$
12: **end while**
13: **while** $r > 0$ **do** ▷ Ajout d'arêtes jusqu'à en avoir le bon nombre
14: i, j deux entiers tirés au hasard entre 0 et n, $i \neq j$
15: **if** $(i, j) \notin E$ **then**
16: $E \leftarrow E \cup (i, j)$
17: $r - -$
18: **end if**
19: **end while**
20: return G
21: **end function**

Connexité La première boucle (ligne 10) de l'algorithme3 garanti la connexité du graphe de retour.

Uniformité de l'aléatoire Pour la première boucle, on génère un arbre de taille $n+1$ en connectant un sommet à un arbre de taille n. Or tous les arbres de taille $n+1$ se construisent de cette façon.

Dans la seconde boucle, on commence avec un graphe a n sommets et m arêtes. On veut obtenir un graphe avec n sommets et $m+1$ arêtes. Si on interdit les doublons et les boucles, on remarque qu'il y a $n(n-1)-m$ façons de faire ce choix. Or la méthode qui consiste à tirer de façon uniforme 2 sommets $n(n-1)$ résultats possibles (et équiprobables), si l'on retire à ces résultats les m arêtes déjà présentes dans G on retombe sur $n(n-1)-m$.

Ainsi mis à part le choix initial du sommet n (ligne 5), les étapes de constructions sont uniformément aléatoires. On a donc la même probabilité d'obtenir n'importe lequel des graphes à n sommets et $n-1+3\sqrt{n}$ arêtes à renommage près.

4 Résolution approchée

Réalisation de l'algorithme 2 " Departager "

Entrée : La foret T , L'ensemble L (L est l'ensemble de tous les arrêtes de poids minimum de la coupe)

Sortie : Une arrête (u,v)
*Pour Faire ça , On a créer une structure E qui représente les arrêtes :

```
typedef struct E
{
uint u;
uint v;

}E_t;
```

*Une fonction : **void afficher_arrets(arret_p p)**

qui sert à afficher les arrêtes ce qui nous à aider dans les tests

*On a implémenter aussi deux fonctions **createForet(int n)** (Fonction qui set à créer des Tree à partir d'un entier n qui est le nombre de graphe de tree)et **createl(int n)** (Fonction qui sert à créer un ensemble qui contient n arrêtes)

*Enfin L'implémentions de **arret_p departager(graph_p* foret, arret_p* l)** qu résolu l'algorithme suivant :

if (il existe (u,v) $\in L|(d_T(u) = 1 \ \& \ d_T(v) = 1)$)

Then

choisir (u,v)

Else

choisir(u,v) $\in Lauhasard$

end if

5 Résolution exacte : programme linéaire

5.1 Programme linéaire

On travaille avec les variables binaire suivantes :

— $z_i, i \in V$, vaut 1 si i est dans la solution et est un sommet de branchement ;
— $x_{ij}, ij \in E$, indique si oui ou non l'arête ij est dans la solution ;
— $y^k_{ij}, ij \in E, k \in V$, vaut 1 si ij est dans la solution ET k est dans la même composante connexe que j après suppression de ij.

De plus, on note $d(i)$ le degré de i et $V(i)$ le voisinage de i.

$$\min \sum_{i=1}^{n} z_i$$

$$s.c. \begin{cases} \sum_{ij \in E} x_{ij} = n - 1 & \\ y^k_{ij} + y^k_{ji} = x_{ij} & \forall ij \in E, k \in V \\ \sum_{k \in V \setminus \{i,j\}, ik \in E} y^j_{ik} + x_{ij} = 1 & \forall ij \in E \\ \sum_{ij \in V(i)} x_{ji} - d(i) z_i \leq 2 & \forall i \in V \\ z_k, x_{ij}, y^k_{ij}, y^k_{ji} \in 0, 1 \forall k \in V, \forall ij \in E & \end{cases}$$

5.2 Réponses aux questions

La première contrainte impose un maximum de $n-1$ arêtes dans la solution. La deuxième indique que si ij est dans la solution, alors pour tout sommet k, lorsque l'on supprime ij, k se retrouve nécessairement avec i ou j mais pas avec les deux. La troisième impose que si ij est dans la solution, alors aucune arête ik est dans la solution avec k dans la même composante connexe que j. Enfin la dernière contrainte indique que si le sommet i est connecté à plus de deux autres sommets dans la solution alors, on doit le considérer comme un sommet de branchement ($z_i = 1$).

La contrainte (2) impose la connexité de la solution et la troisième empêche les cycles. En ajoutant cela à la première contrainte on obtient que la solution est un graphe connexe sans cycles et de $n-1$ sommets, c'est-à-dire un arbre couvrant de G. La dernière contrainte ajoutée à la fonction objectif permet de minimiser le nombre de sommets de branchement.

Cela fait un total de $m + n + 2nm$ variables et $1 + nm + m + n$ contraintes.

5.3 Implantation

Malheureusement l'implantation logicielle basée sur la librairie GLPK comporte encore quelques bugs : parfois le solveur ne trouve pas de solution alors que le graphe passé en argument est connexe...