Groupe 1 Projet Le 20 novembre 2020

Durée: 45 jours

Optimisation 2

Pour un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$, un sommet de $\mathcal{G}$ est dit sommet de branchement (abrev. SB) s'il a un degré strictement supérieur à 2. Le problème NP-difficile MBVST consiste à trouver un arbre de recouvrement de $\mathcal{G}$ ayant un minimum de sommets de branchement. Ce problème a été largement étudié dans la littérature au court de ces quinze dernières années, tant sur le volet résolution exacte que sur celui de la résolution approchée. Il trouve son intérêt pratique principalement dans le routage Broadcast appliqué aux réseaux optiques. Étant la structure connexe permettant de couvrir les sommets en utilisant un minimum de liens, l'arbre est le plus utilisé pour ce type de routage. Dans les réseaux tout optique, les fonctions de commutation et de routage sont fournies par les brasseurs OXC. Certains OXC, onéreux, peuvent diviser une longueur d'onde entrante vers plusieurs ports de sortie grâce à un coupleur optique. Cette division gènere un affaiblissement du faiseau lumineux ainsi qu'une dégradation du signal (pertes). Un sommet de branchement dans l'arbre correspond a un noeud équipé de coupleur dans le réseau. Afin de réduire les coûts et de limiter les pertes, il convient de minimiser leur nombre.

Dans ce projet, nous allons nous intéressé à deux axes de résolution. Le premier est la résolution exacte que nous allons aborder avec la programmation linéaire en nombres entiers. La seconde est la résolution approchée ue nous allons traiter grâce à l'élaboration de deux heuristiques. Les performances de ces différents algorithmes seront évaluées sur des graphes aléatoires. En effet, les jeux de tests sur différentes tailles et densité de graphes aléatoires permettront d'obtenir le temps de résolution moyen de l'algorithme exact mais aussi d'avoir la valeur de la solution optimale. Cette dernière sera le point de comparaison de la qualité des solutions approchées données par les heuristiques.

Ce projet à pour vocation de remplacer le partiel. Chacun des algorithmes décrit ci-dessous est à coder en langage C. La résolution exacte du programme linéaire peut se faire en utilisant les solveurs Cplex ou bien GLPK. Un rapport de quelques pages est demandé. Celui-ci doit contenir les courbes des tests effectués sur les différents algorithme ainsi que l'interprétation des courbes et les conclusions que vous en tirées.
La date de rendu du rapport ainsi que le code est fixée au Lundi 4 janvier 2021. Une soutenance est programmée le mercredi 6 janvier. Celle-ci dure 20 minutes pour chaque binôme (15 minutes de présentation et 5 minutes de questions).

I Génération de graphes aléatoires

Nous nous intéressons dans cette partie à la génération de graphes aléatoires. Ces graphes sont nécessaires pour faire des jeux de tests représentatifs en l'absence de benchmarks. Ils nous permettrons donc d'évaluer, par la suite, la qualité des heuristiques ainsi que celle de l'algorithme exact. Les graphes aléatoires que nous allons générer doivent être connexes. L'algorithme de Roy-Warshall permet de tester la connexité d'un graphe.
L'algorithme de Roy-Warshall est le suivant :

Entrées: Un graphe orienté $G = (U, A)$
Sorties: La fermeture transitive de G
  Pour $w \in U$ **Faire**
    Pour $u \in U$ **Faire**
      Pour $v \in U$ **Faire**
        Si $(u, w) \in A$ et $(w, v) \in A$ **Alors** Ajouter (u, v) à A.

---

1) codez l'algorithme en langage C, de sorte qu'il prenne en entrée la matrice d'adjacence d'un graphe.

2) Testez l'algorithme sur le graphe décrit par la matrice suivante :

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

3) A partir de la fermeture transitive renvoyée par l'algorithme de Roy-Warshall, écrire une fonction qui renvoie si deux sommets donnés en argument sont dans la même composante connexe ou non.

4) Écrire un qui génère des graphes aléatoires non orientés connexes en utlilisant l'algorithme de de Roy-Warshall pou garantir la connexité.

5) Grâce à cet algorithme, générerez 10 graphes aléatoires de taille $|V| = \{20, 50, 100, 400, 600, 1000\}$ et dont la densité (nombre d'arêtes) respecte la formule : $\lfloor(|V| - 1) + 2 \times 1.5 \times \lceil\sqrt{|V|}\rceil\rfloor$.

II Résolution approchée (heuristique)

- Récrire l'algorithme ***Edge Weighting Strategy*** en C.

Algorithm 1 *Edge weighting strategy*

Input: A connected graph $G = (V, E)$
Output: A spanning tree $T(V', E')$

Initialize $T(V', E')$ as $(V, \emptyset)$
for all $(u, v) \in E$ **do**
  $w(u, v) \leftarrow 1$
end for
$A \leftarrow E$
while $|E'| \neq n - 1$ **do**
  $L \leftarrow \{(u', v') \in A \mid w(u', v') \leq w(u, v), \forall \{u, v\} \in A\}$
  $\{u^*, v^*\} \leftarrow select(L)$ {Selection criterion to tie break}
  $A \leftarrow A \setminus \{\{u^*, v^*\}\}$
  if u^* and v^* are in different components of T **then**
    $T \leftarrow T \cup \{\{u^*, v^*\}\}$
    Update the weights of u^* and v^*
    $cover(\{u^*, v^*\}, G, T)$
  end if
end while

Notons par $\mathcal{C}_\mathcal{G}(v)$ le nombre de composantes connexes de $\mathcal{G} - v$ et par $d_\mathcal{G}(v)$ le degré de v dans $\mathcal{G}$. Nous commençons par diviser l'ensemble des sommets du graphe en trois types :

$Type(v) = 0$	$d_\mathcal{G}(v) = 1$ ou $(d_\mathcal{G}(v) = 2$ et $\mathcal{C}_\mathcal{G}(v) = 2)$	Non SB et toutes les arêtes incidentes sont dans $\mathcal{T}$
Type$(v) = 1$	$d_\mathcal{G}(v) = 2$ et $\mathcal{C}_\mathcal{G}(v) = 1$	Non SB et au moins une arête incidente est dans $\mathcal{T}$
Type$(v) = 2$	$d_\mathcal{G}(v) \geqslant 3$ et $\mathcal{C}_\mathcal{G}(v) \leqslant 2$	Peut être SB dans $\mathcal{T}$
Type$(v) = 3$	$d_\mathcal{G}(v) \geqslant 3$ et $\mathcal{C}_\mathcal{G}(v) \geqslant 3$	Sommet articulation, nécessairement SB dans $\mathcal{T}$

Nous notons par $Cut_\alpha(\mathcal{G}) = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E}'')$ une coupe de poids minimum de $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \alpha)$, tel que $\alpha : \mathcal{E} \to \mathbb{N}^+$ est une fonction qui attribue un poids à chaque arête de $\mathcal{G}$. L'ensemble $\mathcal{E}''$ est l'ensemble des arêtes de $\mathcal{G}$ qui ont une extrémité dans $\mathcal{V}_1$ et une extrémité dans $\mathcal{V}_2$. L'idée de cette heuristique est d'ajouter une fonction $\omega : \mathcal{E} \to \mathbb{N}^+$ qui attribue un poids à chaque arête $(u, v) \in \mathcal{E}$. Le poids $\omega(u,v)$ de l'arête (u,v) est utilisé comme indicateur de la probabilité que cette arête crée un SG si elle est sélectionnée pour être dans l'arbre (pénalité). L'algorithme commence par assigner le poids 1 à chaque arête du graphe $\mathcal{G}$. À chaque itération, une coupe $Cut_\alpha(\mathcal{G})$ de poids minimum est calculée. Le poids $\alpha(v)$ permet d'éviter qu'une arête figurant préalablement dans l'arbre ne soit dans $\mathcal{E}''$. Afin d'éviter les cycles, une seule arête $(u,v) \in \mathcal{E}''$ ayant un poids minimum est ajoutée à l'arbre $\mathcal{T}$. Si une extrémité de l'arête est de type 2, alors le poids de chaque arête qui lui est incidente est incrémenté de 1, ce qui a pour effet de diminuer leur probabilité d'être sélectionnée à l'itération suivante. Si un sommet passe du type 2 au type 3, alors l'ensemble des pénalités causées par l'ajout d'arêtes incidentes à ce sommet sont annulées. Si un sommet est de type 3 alors il est relié à tous ses voisins tant qu'aucun nouveau SG et aucun cycle ne sont crées dans l'arbre. L'algorithme s'arrête lorsque $|\mathcal{V}| - 1$ arêtes ont été sélectionnées.

Algorithm 1 Heuristique MBVST

Entrée : Un graphe connexe $\mathcal{G} = (\mathcal{V},\mathcal{E},\omega,\alpha)$
Sortie : Un arbre de recouvrement $T = (\mathcal{V},\mathcal{E}')$

Initialiser $T(\mathcal{V},\mathcal{E}')$ à $(\mathcal{V},\emptyset)$
for all $(u,v) \in \mathcal{E}$ **do**
  $\omega(u;v) \leftarrow 1$
  $\alpha(u;v) \leftarrow 1$
end for
for all sommets v de type 3 **do**
  Saturer(T,v)
end for
while $|\mathcal{E}'| \neq |\mathcal{V}| - 1$ **do**
  Calculer une coupe minimum $Cut_\alpha(\mathcal{G}) = (\mathcal{V}_1,\mathcal{V}_2,\mathcal{E}'')$
  $L \leftarrow$ Les arêtes de poids minimum de $\mathcal{E}''$
  if Le poids min est 3 **then**
    Departager(T,L)
  else
    Choisir $(u,v) \in \mathcal{E}''$ au hasard
  end if
  $\mathcal{E}' \leftarrow \mathcal{E}' \cup (u,v)$
  if Type$(u) = 2$ **then**
    Augmenter de 1 le poids ω de toutes les arêtes incidentes à u dans $\mathcal{G}$
    ChangerType(T,u)
  end if
  if Type$(v) = 2$ **then**
    Augmenter de 1 le poids ω de toutes les arêtes incidentes à v dans $\mathcal{G}$
    ChangerType(T,v)
  end if
  $\alpha(u,v) \leftarrow +\infty$
end while
return T

Algorithm 2 Departager

Entrée : La foret T, l'ensemble L
Sortie : Une arête (u,v)

if $(\exists(u,v) \in L /\ (d_T(v) = 1\ \&\ d_T(u) = 1))$ **then**
  Choisir (u, v) ;
else Choisir $(u,v) \in L$ au hasard
end if

Algorithm 3 Saturer

Entrée : La foret T, Le graphe $\mathcal{G}$, sommet v
Sortie : La foret T
for all $(u,w) \in (\mathcal{E} - \mathcal{E}')$ **do**
  if u et w sont dans différentes composantes connexes dans T & Type$(w) \neq 2$ **then**
    $\mathcal{E}' \leftarrow \mathcal{E}' \cup (u,w)$
    $\alpha(u,w) \leftarrow +\infty$
  end if
end for

Algorithm 4 ChangerType

Entrée : La foret T, sommet v
Sortie : La foret T
if $d_T(v) > 2$ **then**
  $Type(v) \leftarrow 3$
  Diminuer de 3 le poids ω de toutes les arêtes incidentes à v dans $\mathcal{G}$
  Saturer(T,v)
end if

L'algorithme ***Heuristique MBVST*** trouve un graphe sans cycle ayant $|\mathcal{V} - 1|$ arêtes (arbre) en $\mathcal{O}(|\mathcal{V}|^2|\mathcal{E}| + |\mathcal{V}|^3 log|\mathcal{V}|)$. Les coupes minimales sont trouvés grâce à l'algorithme de Stoer-Wagner.

1) codez l'algorithme ***Heuristique MBVST*** en langage C , en vous aidant de la librairie Boost pour trouver les coupes minimums (L'algorithme de Stoer-Wagner s'y trouve).

2) appliquez l'algorithme ***Heuristique MBVST*** sur les graphes aléatoires précédemment trouvés.

III Résolution exacte

Étant donné un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$, on note par $\gamma(v)$ le degré du sommet v dans $\mathcal{G}$. La variable x_{ij} pour chaque $(i,j) \in \mathcal{E}$ vaut 1 si l'arête (i,j) est dans la solution, elle vaut 0 sinon. La variable $y^k_{i,j}$ pour chaque $(i,j) \in E$ et $k \in \mathcal{V}$ vaut 1 si l'arête (i,j) est dans la solution et le sommet k est dans la composante connexe contenant j après la surpression de (i,j), elle vaut 0 si l'arête (i,j) n'est pas dans la solution ou bien si l'arête (i,j) est dans la solution mais le sommet k n'est pas dans la composante connexe contenant j.

$$\min \sum_{v \in \mathcal{V}} z_v$$

$$s.t. \begin{cases} \displaystyle\sum_{(i,j)\in\mathcal{E}} x_{i,j} = n - 1 & & (1a) \\ y^k_{ij} + y^k_{ji} = x_{ij} & \forall (i,j) \in E, k \in \mathcal{V} & (1b) \\ \displaystyle\sum_{k \in \mathcal{V}\setminus\{i,j\}} y^j_{ik} + x_{i,j} = 1 & \forall (i,j) \in E & (1c) \\ \displaystyle\sum_{(i,j)\in\gamma(i)} x_{ij} - |\gamma(i)| z_i \leqslant 2 & \forall i \in \mathcal{V} & (1d) \\ x_{ij}, y^k_{ij}, y^k_{ji} \in \{0,1\} & & (1e) \end{cases}$$

1) Expliquez chacune des contraintes (1a), (1b), (1c).

2) Déduisez que les contraintes (1b), (1c) empêchent l'existence de cycles dans la solution.

3) Déduisez qu'en ajoutant la contrainte (1a) aux contraintes (1b), (1c), on obtient un graphe partiel couvrant connexe et sans cycle (un arbre couvrant).

4) Expliquez pourquoi la contrainte (1d) ajoutée à la fonction objective permet de minimiser le nombre de sommets de branchement dans la solution retournée par le PLNE.

5) Quel est le nombre de variables et de contrainte du PLNE en fonction de la taille de l'instance ?

6) codez le PLNE en langage C et le résoudre en utilisant GLPK ou Cplex.

7) Trouver la solution optimale sur les graphes aléatoires précédemment trouvés.

8) Comparez, pour chaque taille de graphe, la valeur de la solution optimale et celle de la solution approchée trouvée avec l'heuristique ***Heuristique MBVST*** (utilisez des courbes pour expliciter la comparaison).