Algorithmes du Monde Réel
Projet

Partie 2 - Approximations

Professeurs : M. Zeitoun, A. Muscholl, C. Gavoille

Master 2 Informatique

Ludovic Brochard, Fabien Kuntz, Benoît Védrenne

Table des matières

1 Organisation du projet

1.1 Partie 1

Le but de cette partie est de trouver un algorithme qui calcule la couverture par sommets minimale d'un arbre en temps linéaire.

Pour cela, nous allons montrer une propriété des couvertures par sommets sur les arbres, puis utiliser cette propriété pour proposer un algorithme linéaire de résolution du problème de minimisation d'une couverture par sommets sur un arbre.

1.2 Partie 2

Dans cette partie, on veut comparer deux algorithmes d'approximation pour la couverture par sommets :

- L'algorithme du projet qui calcule un arbre correspondant à une recherche en profondeur puis qui choisit comme couverture l'ensemble des sommets qui ne sont pas des feuilles.
- L'algorithme du cours qui utilise le couplage maximal.

Nous allons justifier que l'algorithme du projet est bien un algorithme 2-approché du problème couverture par sommets et nous allons comparer les temps d'exécution et taille de la solution des deux algorithmes.

1.3 Organisation du rapport

Nous allons donc vous présenter dans ce rapport les étapes de la première partie du sujet, puis de la deuxième pour continuer par parler rapidement de l'implémentation du projet et enfin effectuer un bilan rapide.

2 VC et arbres

2.1 Question 1

Soit F une forêt et uv une arête telle que le degré de v est 1.

Considérons un couverture U de F qui ne contient pas u. La couverture ne contenant pas u et uv étant une arête, alors U contient nécessairement v.

Or, le degré de v est 1, donc le fait que v soit dans U ne sert qu'à couvrir l'arête uv. On peut donc choisir u dans U à la place de v, la seule arête que v couvrait est alors aussi couverte par u et la taille de la couverture est la même.

On peut donc construire à partir d'une couverture minimale U de F qui ne contient pas u, la couverture U' de même taille qui contient u.

2.2 Question 2

2.2.1 Idée de l'algorithme

Afin de calculer une couverture de taille minimale sur un arbre T, nous utilisons un parcours en profondeur de l'arbre.
En utilisant la Question 1 (2.1 page 1), nous marquons les sommets qui sont parents d'une ou plusieurs feuilles car les feuilles sont de degré 1. Ensuite, lors de la partie "retour" de la récursion, nous allons marquer les sommets qui sont pères de sommets non marqués.

Pour qu'un sommet sache s'il doit être marqué, chacun de ses fils va lui dire *"Marque toi !"*$(true)$ ou *"Je suis déjà marqué donc tu n'as pas besoin de l'être vis-à-vis de moi."*[1]$(false)$.

Marquer un sommet revient à mettre la case qui lui correspond dans le tableau de booléens *cover* à $true$.

[1] Certains fils sont plus bavards que d'autres.

2.2.2 Algorithme

Algorithm 1: coverTree() : Boolean table

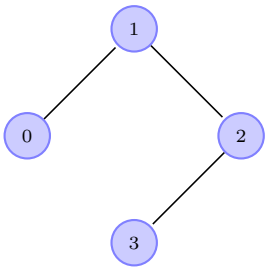
```
1 global var cover : Boolean table ;
2 begin
3 |   foreach v ∈ V do
4 |   |   cover[v] := false ;
5 |   end
6 |   coverTree_aux(root) ;
7 |
8 |   return cover ;
9 end
```

Algorithm 2: coverTree_aux(Vertex v) : Boolean

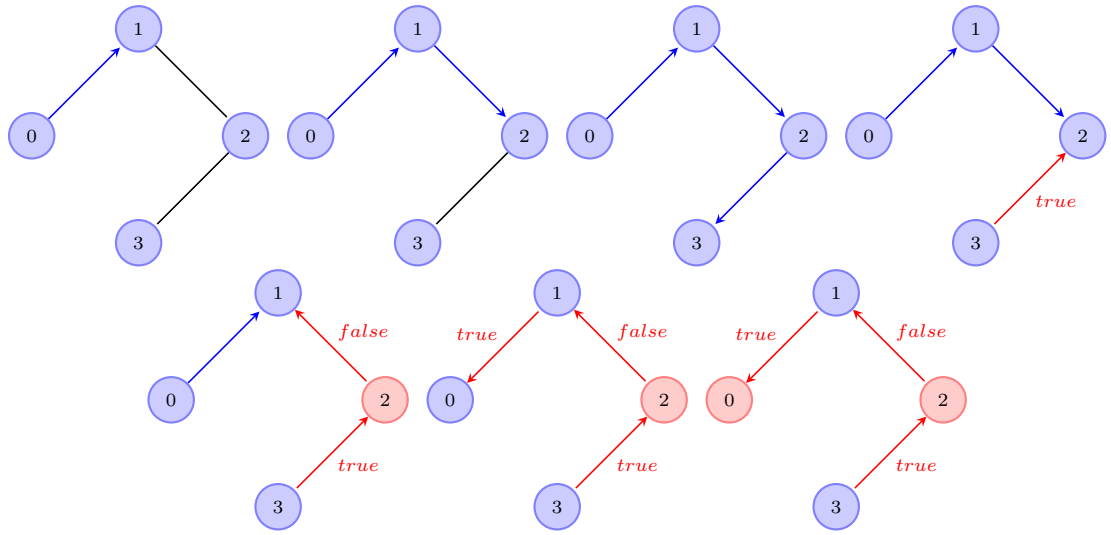
```
1  local var markFather := true : Boolean ;
2  begin
3  |   if v is a leaf then
4  |   |   return true ;
5  |   end
6  |   foreach (v, w) ∈ E do
7  |   |   if coverTree_aux(w) then
8  |   |   |   cover[node] := true ;
9  |   |   |   markFather := false ;
10 |   |   end
11 |   end
12 |
13 |   return markFather ;
14 end
```

2.2.3 Exemple

On nous donne un arbre en entrée sur lequel on souhaite trouver une couverture par sommets.



On applique alors notre algorithme.



L'algorithme nous retourne alors le tableau de booléens $[1, 0, 1, 0]$ qui correspond à la couverture $\{0, 2\}$.

2.2.4 Complexité

L'algorithme n'utilise qu'un parcours en profondeur et des calculs constants (le calcul du nombre de voisins d'un sommet est aussi constant car nous avons stocké le degré des nœuds dans la structure du graphe). La complexité est donc celle du parcours en profondeur, c'est-à-dire linéaire.

3 VC et graphes

3.1 Question 1

L'algorithme du projet renvoie tous les sommets non feuilles de l'arbre T obtenu à partir d'un parcours en profondeur du graphe d'origine G. Pour montrer que l'on obtient bien une couverture de G, il faut montrer qu'il n'y a pas d'arêtes entre les feuilles de l'arbre (les feuilles sont les seuls sommets qui ne sont pas dans la couverture).

Soient u et v deux feuilles de l'arbre T. Il ne peut pas exister une arête entre u et v dans le graphe d'après la définition du parcours en profondeur. En effet, soit u est un descendant de v dans le parcours et alors v n'est pas une feuille de T, soit v est un descendant de u dans le parcours et donc u n'est pas une feuille de T [2].

Il n'y a donc aucune arête possible entre les feuilles de l'arbre dans le graphe G, on en déduit ainsi que l'algorithme du projet retourne bien une couverture.

3.2 Question 2

Le but ici est de faire la comparaison entre l'algorithme du projet vu dans la question précédente (3.1 page 3) et l'algorithme vu en cours basé sur un couplage maximum.

Pour comparer ces deux algorithmes, nous allons mesurer leur temps d'exécution ainsi que la taille de la couverture trouvée. Le résultat de ces mesures se trouve dans le tableau 1 page 4. Toutes les comparaisons sont faites sur trois graphes[3] : *ex32-12.gin, testTree1000.gin, testTree100.gin.*

Le calcul des temps d'exécution se fait à l'aide des fichiers *timing.cc* et *timing.h* que nous avons récupérés des projets *Padico*[4] et *Adage*[5] de l'*INRIA.*

On peut constater sur le tableau 1 page 4 que le temps d'exécution de l'algorithme du couplage maximum est bien plus rapide que l'algorithme du projet. Pour le graphe de taille 1000, le temps d'exécution est 70 fois plus rapide pour l'algorithme de couplage.

[2] Démonstration par infusion sur les feuilles de T
[3] Graphes fournis dans le répertoire jeuxRapport
[4] http://gforge.inria.fr/projects/padico
[5] http://gforge.inria.fr/projects/adage

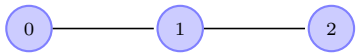
	ex32-12.gin		testTree100.gin		testTree1000.gin	
	algoP	algoC	algoP	algoC	algoP	algoC
Temps d'execution(en μs)	58.5808	1.12014	400.344	5.82175	3911.25	56.8837
Taille de la couverture	8	14	38	58	365	576
Taille couverture optimale	7		31		325	

FIG. 1 – Tableau comparatif des exécutions des plusieurs exemples (algoP : Algorithme du projet et algoC : Algorithme par couplage maximum).

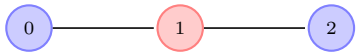
Néanmoins, si l'on compare la taille des couvertures retournées, on constate que l'algorithme du projet est bien meilleur. On voit que sur un graphe à 1000 sommets, ce dernier trouve une converture à 365 sommets contre 576 pour l'autre. De plus, les tests étant réalisés sur différents types de graphes, et notamment *ex32-12.gin, testTree1000.gin, testTree100.gin* qui sont des arbres, que plus le graphe comportera de cycles et aura beaucoup de feuilles (des graphes assez développés quand même car un graphe étoile n'irai pas), plus la différence entre les deux algorithmes se fera sentir.

3.3 Question 3

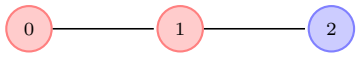
Considérons le graphe segment de trois sommets suivant :



La couverture minimale de ce graphe est $\{1\}$ de taille 1.



L'algorithme du projet, lui, va retourner les sommets non feuilles de ce graphe qui est un arbre, c'est-à-dire par exemple $\{0, 1\}$ qui sera de taille deux fois plus grande.



3.4 Question 4

Montrer que G possède un couplage C de taille $|S|/2$.

Inachevé : Nous avons voulu donc montrer qu'il n'existe aucun couplage maximal pour G dont la taille soit strictement plus petite que $|S|/2$. C'est-à-dire que pour tout couplage C tel que $|C| < |S|/2$, on peut créer un couplage C' qui est C où on a ajouté une arête de G. C'est donc que C n'était pas maximal.

Nous n'avons malheureusement pas réussi à démontrer cette propriété, ni par cette dernière méthode, ni avec d'autres idées que nous avons eues (démonstration par induction sur la taille de S ou C, démonstration par calcul du nombre d'arêtes minimum de C, ...). Nous allons donc admettre cette propriété pour montrer la 2-approximation.

C est un couplage du graphe, donc on sait que toute couverture doit être de taille au moins $|C|$ (il est obligatoire de choisir au moins l'un des deux sommets de chacune des arêtes du couplage). On a donc :

$$|OPT| \geq |C| \iff |OPT| \geq |S|/2 \iff 2 * |OPT| \geq |S|$$

Nous avons donc $|S| \leq 2 * |OPT|$ et on sait que l'algorithme atteint le cas double (voir question précédente 3.3 page 4), on peut donc en déduire que l'algorithme du projet est 2-approché.

4 Implémentation

4.1 Réutilisation de code

Nous avons réutilisé la classe que nous avions implémentée dans le premier projet pour représenter les graphes ainsi que la partie *"parser"* de la classe principale.

Il nous restait donc à implémenter la classe d'arbre.

4.2 Implémentation de la classe d'arbre

Nous avons donc créé la classe *Tree.cpp* qui dépend de *Graph.cpp* où un arbre est créé à partir d'un parcours en profondeur sur le graphe passé en paramètre.

4.3 Algorithmes

L'algorithme de la partie 1 a été implémenté dans *Tree.cpp* dans la méthode *coverTree*.

L'algorithme du projet (partie 2) a été implémenté dans *Graph.cpp* dans la méthode *coverProject*.

L'algorithme du cours (partie 2) a été implémenté dans *Graph.cpp* dans la méthode *coverCourses*.

4.4 Exécution du programme

Le programme est exécuté à partir de la commande *./cover*, une fonction d'usage est appelée pour donner plus d'informations sur l'exécution.

5 Bilan

Nous avons remarqué l'intérêt que peuvent avoir les algorithmes d'approximation lorsque le temps d'exécution est important. En effet, nous avons observé que l'algorithme du cours, même moins précis en général, est beaucoup plus rapide et avec la même certitude de 2-approximation que l'algorithme du projet.

Il est donc évident que sur des problèmes de très grandes tailles, lorsqu'il est impossible, au niveau temps, de calculer une solution exacte, on peut se tourner vers les algorithmes d'approximation qui sont d'une bien meilleure efficacité et qui donnent des solutions proches d'une solution optimale.