28/12/2014

Quentin Choullet

Céline de Roland

Johan Ravery

TP INFO 724

NP-Complétude de Min-Coupe-Circuit en utilisant le problème de Couverture des Arêtes

Table des matières

DESCRIPTION ET ENCODAGE DES PROBLÈMES

1) Description de P1

Le problème P1 est appelé Min-Coupe-Circuit (abrégé Min dans nos noms de fichiers) :

« Etant donné un graphe orienté G et un entier n, est-il possible d'éliminer tous les cycles de G en lui ôtant n arêtes »

2) Encodage de P1

Une instance de P1 s'écrit :

Digraph {

Liste des sommets séparés par des espaces (les noms des sommets doivent être des entiers consécutifs)

Liste des arêtes séparées par des sauts de ligne

(une arête s'écrit numéroSommetSource -> numéroSommetCible)

}

Entier {

UnNombreEntier (correspond au nombre n de la définition)

}

3) Description de P2

Le problème P2 est appelé Couverture des Arêtes (abrégé CA dans nos noms de fichiers) :

« Etant donné un graphe non orienté G et un entier n, est-il possible de colorier n sommets tels que l'ensemble des arêtes touchant les sommets coloriés soit égal à l'ensemble des arêtes du graphe G »

4) Encodage de P2

Une instance de P2 s'écrit :

Graph {

Liste des sommets séparés par des espaces (les noms des sommets doivent être des entiers consécutifs)

Liste des arêtes séparées par des sauts de ligne

(une arête s'écrit numéroSommetSource -- numéroSommetCible, contrairement au cas du graphe orienté, les sommets source et cible sont interchangeables)

}

Entier {

UnNombreEntier (correspond au nombre n de la définition)

}

5) Description et encodage du certificat

Un certificat pour le problème P2 est un ensemble de n arêtes à couper pour éliminer tous les cycles.

Un certificat s'écrit :

Digraph {

Liste des sommets séparés par des espaces (les noms des sommets doivent être des entiers consécutifs)

Liste des arêtes à couper séparées par des sauts de ligne

(une arête s'écrit numéroSommetSource -> numéroSommetCible)

}

Entier {

UnNombreEntier (correspond au nombre n de la définition)

}

VÉRIFICATEUR

On lit une instance de P1 (MinCoupeCircuit).

On lit le certificat

On élimine de l'instance de P1 toutes les arêtes du certificat

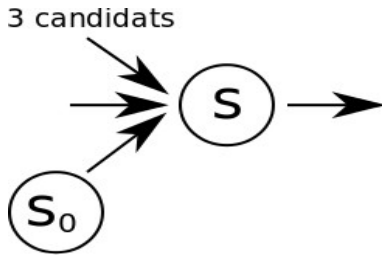
On vérifie en temps polynomial que le nouveau graphe orienté obtenu ne contient pas de cycle avec l'algorithme possedeUnCycle (dans la classe DiGraph).

Principe sur lequel s'appuie l'algorithme possedeUnCycle :

On crée une variable nbsommets, qui compte le nombre de sommets n'appartenant pas à un cycle.

Pour qu'un sommet s soit dans un cycle, il faut et il suffit que le cycle contienne une arête entrant dans s et une arête sortant de s :

Pour chaque sommet s, les arêtes entrantes sont des candidates à la formation d'un cycle passant par s :



Si le sommet s_0 n'est pas dans un cycle, alors on peut élimner l'arête $s_0 \rightarrow s$ des candidats.

Au départ, on stoque dans un tableau d le nombre de candidats (ie d'arêtes entrantes) de chaque sommet. On crée une pile « atraiter » contenant les sommets dont on est certains qu'ils ne sont pas dans un cycle.

Les sommets n'ayant aucune arête entrante ne sont pas dans un cycle, donc on incrémente nbsommet dès qu'on rencontre un sommet n'ayant aucune arête et on empile ce sommet dans « atraiter ».

Pour chaque sommet à traiter, on s'intéresse à ses voisins et on élimine l'arête des candidats (on peut éliminer dans l'exemple le candidat $s_0 \rightarrow s$. Si on a éliminé tous les candidats du sommet s, on empile s dans « atraiter » car on est alors certain que s n'est pas dans un cycle.

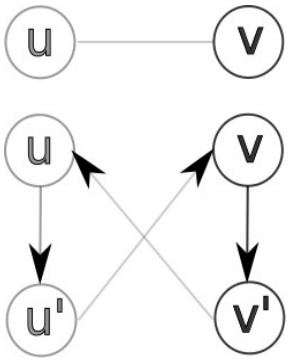
A la fin, si le nombre de sommets qui ne sont pas dans un cycle est égal au nombre de sommets du graphe, alors il n'y a pas de cycle. Dans le cas contraire, c'est qu'il y a au moins un cycle.

La page suivante décrit de façon précise l'algorithme et montre ainsi que sa complexité est en O(nbsommet x nbaretes). Si N est la taille de l'instance, on a nbsommet < N et nbaretes < N donc la complexité de l'algorithme est en $O(N^2)$.

```
algo possedeUnCycle

    d : tableau de nombre entiers, de taille nbNoeudsDuGraphe , initialisé à 0
    atraiter : pile de nœuds (vide au départ)
    nbsommets : nombre entier initialisé à 0

    pour chaque nœud n du graphe    ==========> Complexité : nbSommets
       d[n] ← nombre d'arêtes entrantes
       si aucune arête entrante alors
          atraiter.empiler(n)
          incrémenter nbsommets
    fin pour

    tant qu'il reste des nœuds à traiter  ==========> Complexité < nbSommets x nbAretes
       n ← atraiter.depiler();
       pour toutes les arêtes a sortantes de n
          decrémenter d[a.destination()]
          si d[a.destination] = 0 alors
             atraiter.empiler(a.destination());
             incrémenter nbsommets
          fin si
       fin pour
    fin tant que


    si nbsommets = nbNoeudsDuGraph alors G ne possède pas de cycle
    sinon G possède un cycle;
    fin si

 fin algo possedeUnCycle
```

RÉDUCTEUR

Le principe de la réduction est le suivant : On transforme le graphe non orienté G en un graphe orienté G', de façon à ce que colorier un sommet dans G revienne au même que couper un cycle dans G'. On réalise cela en transformant chaque sommet en paire de sommet, et chaque arête en cycle comme sur le schéma ci-dessous :



Colorier u dans G revient à couper u → u' dans G'.

SOLVER

Le principe de base du solveur est le suivant :

- On cherche l'ensemble de tous les cycles du graphe
- On donne un poids à chaque arête, correspondant au nombre de cycles passant par cette arête
- On fait un algorithme de type backtrack en commençant par supprimer les arêtes de poids le plus fort

1) Algorithme de recherche de l'ensemble des cycles du graphe

On fait un parcours en profondeur, et dès qu'on revient sur un sommet déjà visité dans la même branche, on obtient un cycle.

```
algo trouverTousLesCycles() {

    tableau vu initialisé à faux (nœuds déjà explorés dans le parcours en profondeur)
    tableau explore initialisé à faux (nœuds explorés dans une branche donnée)
    cycles : liste des cycles trouvés;

    tant qu'il reste des sommets non vus
        vu[premierNonVu] = true;
        trouverTousLesCyclesRec(premierNonVu, nouveau cycle, explore);
    fin tant que

fin algo

algo trouverTousLesCyclesRec(Noeud n, boolean[] vu,  Cycle cycle, boolean[] explore)

    vu[n] ← true, explore[n] ← true

    pour toutes les aretes a sortant de n
        on crée un nouveau cycle cycleCourant égal à cycle + a
        si explore[a.destination() ] = faux
            trouverTousLesCyclesRec(a.destination(),vu,cycleCourant,explore);
        sinon
            du cycle courant on extrait le véritable cycle ( exemple : de 1 – 3 – 5 – 4 – 2 – 5, on
extrait  5 – 4 – 2 – 5 )
            cycles.ajouter(cycleCourant);
        fin si
    fin pour

    explore[n] = faux;

fin
```

2) Algorithme de résolution

INSTANCES PROPOSÉES

Nous avons créé des instances vraies et fausses de CA en utilisant les scripts proposés dans l'énoncé. Nous avons adapté l'encodage des instances obtenues à nos impératifs (sommets indicés consécutivement).

Nous avons ensuite utilisé notre réducteur pour créer des instances de Min-Coupe-Circuit correspondantes (de taille double).

Puis nous avons trouvé intuitivement les sommets à colorier pour les instances de CA vraies, nous en avons déduit les arêtes à couper dans les instances de Min correspondantes, ce qui nous a permis d'écrire des certificats corrects.

Des représentations graphiques des instances et certificats sont disponibles dans le dossier img du dossier fourni. Les certificats correspondent aux sommets coloriés / arêtes barrées des représentations des instances vraies.