Complexité I
Rapport de projet

Alexandre Léonardi Ghislain Dugat Guélaud Lepetit
Abdelkader Benameur Loïc Vierin Amine Zayd

2 novembre 2015

Résumé

Rapport du projet de Complexité de Master 1. Ce rapport comporte une analyse des algorithmes utilisés pour résoudre les problèmes suivants :

- Vérification de la propriété « désert » d'un sous-graphe
- Vérification de la propriété « désert maximal » d'un sous-graphe
- Calcul d'un sous-graphe désert maximal quelconque à partir d'un graphe donné
- Calcul d'un sous-graphe désert maximum quelconque à partir d'un graphe donné
- Calcul d'un sous-graphe désert "'presque maximum"' à l'aide d'une heuristique laissée à notre discrétion

Chacun des algorithmes sera brièvement expliqué et accompagné de sa complexité et de son temps d'exécution.

Par ailleurs le tableau récapitulant la participation des différents membres du groupe, ainsi que la configuration de l'ordinateur utilisé pour mener les tests et enregistrer les temps d'exécution, seront présentés dans une première partie.

Table des matières

Algorithme	**Auteur du code**	**Auteur du rapport**	**Complexité**	**Validité**
is_desert	Guélaud	Alexandre	Guélaud	Guélaud
is_maximal	Ghislain	Alexandre	Ghislain	Ghislain
maximal	Loïc	Alexandre	Loïc	Loïc
maximum_exact	Alexandre	Alexandre	Alexandre	Alexandre
maximum_partial	Abdelkader	Alexandre	Abdelkader	Abdelkader

TABLE 1 – Récapitulatif des contributions au projet

1 Participations au projet & configuration de tests

Le tableau TAB 1 a été difficile à établir car la participation de chacun n'est pas toujours précisément mesurable, notamment quand il s'agit de réfléchir à plusieurs sur l'écriture d'un algorithme ; cela a notamment été le cas pour la fonction maximum incomplète qui, bien que probablement triviale, nous a demandé un certain temps de réflexion.

Comme on peut le voir, les personnes qui se sont occupées de la rédaction d'un algorithme se sont aussi chargées de sa vérification et de sa complexité, tandis que le présent rapport est pour sa part l'œuvre d'une seule personne.

Configuration de test :

- *OS* Windows 8.1 Professionnel 64 bits
- *CPU* Intel Core i7-7400MQ 2.40GHz
- *GPU* Nvidia GeForce GTX 760M
- *RAM* 8GB DDR3

2 Types de donnée utilisés

Les types présentés ci-après sont trouvables dans le fichier `types.h`.

2.1 Représentation des graphes

Les graphes sont représentés à l'aide de la méthode des matrices d'adjacence. Ce choix a été fait pour des raisons de simplicité de la représentation principalement.

En effet, même si elle s'avère plus consommatrice en mémoire qu'une représentation par liste chaînée, le gain en temps de codage n'est pas négligeable. Dans la mesure où l'ordinateur de test disposait d'une grande quantité de mémoire vive, ce défaut ne nous a pas paru rédhibitoire.

L'autre raison est la vérification en temps constant de l'existence d'un arc qui est utilisée à de nombreuses reprises dans le code.

2.2 Représentation des sous-graphes

Les sous-graphes sont représentés comme des tableaux de booléens dont la case d'indice i vaut 1 si le sommet i appartient au graphe.

Un alias nous permet de faciliter la représentation en donnant un nom explicite à ces tableaux et en fixant leur taille maximum comme étant le nombre de sommets que peut comporter un graphe au maximum.

Cette représentation a également l'inconvénient de la perte de mémoire, mais encore une fois la quantité de RAM disponible nous a mené à penser que ce ne serait pas un problème.

3 Alogirthmes de vérification

Cette partie regroupe les deux algorithmes de vérification prenant en paramètre un graphe et un sous-graphe, et vérifiant si le sous-graphe est, respectivement, désert et maximal.

Étant très similaires de fonctionnement, nous avons regroupé ces deux algorithmes dans la même partie. À noter qu'ils ne vérifient pas si le sous-graphe est bien un sous-graphe du graphe passé en paramètre : on suppose que les paramètres passés sont corrects.

3.1 Propriété « désert »

3.1.1 Principe

```
booléen is_desert(graphe g, sous-graphe s){
    booléen desert = vrai
    int n = taille de g
    matrice arc = ensemble des arcs de g
    pour i de 0 à n
        pour j de i+1 à n
            si (s[i]==1 et s[j]==1 et arc[i][j]==1) alors
                desert = faux
                sortir des boucles
            fsi
        fpour
    fpour
    retourner desert
}
```

Le principe de cet algorithme est de parcourir l'ensemble des arcs possibles à l'aide d'une double boucle et de vérifier, pour chaque arc :

1. si ses deux extrémités sont présentes dans le sous-graphe
2. s'il est présent dans le graphe

Si tel est le cas, cela veut dire que le sous-graphe n'est pas désert (deux de ses sommets sont reliés entre eux), on sort donc des boucles et on retourne faux.

Si rien ne nous arrête, alors on retourne vrai. En pratique pour éviter l'usage d'un `break`, les boucles sont des `while` qui vérifient à chaque itération la valeur du booléen.

3.1.2 Validité

3.1.3 Complexité

expliquer l'algorithme en français, l'écrire en pseudo-code, justifier sa validité (il n'est pas nécessaire de faire des preuves par invariant ou par induction), donner et justifier sa complexité.

le temps de calcul (pensez à préciser la configuration de l'ordinateur utilisé pour les tests), toutes données pertinentes comme, par exemple, la réponse oui/non au problème de décision considéré ou la taille de la clique maximale/maximum trouvée.

3.2 Propriété « maximal »