Graphes, recherche arborescente et complexité Contrôle continu n°1

Balboni-BOUILLY Ivano, Peltier Albert

1 Isomorphisme de sous-graphes

Problem 1

Exercice 1.a
(G_1, H_1) est une instance positive de ISO-SOUS-GRAPHES car le sous-graphe (1,2,3,4) de G_1 est isomorphe à H_1.
Exercice 1.b
(G_2, H_2) n'est pas une instance positive de ISO-SOUS-GRAPHES car il est impossible d'obtenir des sommets d'ordre 3 à partir du graphe G_2.

Problem 2

2.a
Voir la fonction **verifISG** dans exGraphe.py

2.b
verifISG est de complexité linéaire, plus exactement $O(m)$ avec m le nombre d'arêtes de H.

Problem 3

Exercice 3.
On peut facilement vérifier une solution de ISO-SOUS-GRAPHES à l'aide de verifISG en temps polynomial (linéaire exactement), cependant un algorithme qui a pour couple (G, H) determinant si on a une instance positive de ISO-SOUS-GRAPHES est en temps exponentiel. Un certificat est de taille k (nombre de sommets du sous-graphe) ce qui est de taille raisonnable.
Nos algorithmes résolvant ISO-SOUS-GRAPHES n'étant pas polynomiaux on peut donc classer le problème ISO-SOUS-GRAPHES dans la classe NP.

Problem 4

Exercice 4.a
Voir **forceBruteISG** dans exGraphe.py.

Exercice 4.b
il y a $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ arrangements ce qui correspond dans le pire cas au nombre d'appels de **verifISG**, on a donc une complexité en temps de $O(s\frac{n!}{k!(n-k)!})$ ainsi qu'une complexite en espace de $O(s\frac{n!}{k!(n-k)!})$ car tous les arrangements sont stockés.

Problem 5

Exercice 5.
Les tests avec les graphes sont concluents, (G_{ex}, H_{ex}), (G_1, H_1), (G_3, H_3) ainsi que les graphes engendres par **exemplesInstancesPositives** renvoient bien Vrai et G_2 H_2 rentournent bien Faux.
Les tests sont explicités dans le main de exGraph.py, les resultats sont stockes dans les fichiers **benchmark_small_bruteForce** ainsi que **benchmark_big_bruteForce**.
L'objectif d'avoir les 2 ficher separés est de mettre en avant la nature exponentielle de la fonction **forceBruteISG**. En effet on peut constater dans les test que malgré le fait qu'un seul appel génère **benchmark_big_bruteForce**. C'est bien ce fichier qui montre que le temps d'exécution et bien sur le nombre d'appels de fonction primitives le plus élevé.

Problem 6

Exercice 6.
Pour que ISO-SOUS-GRAPHES soit dans P il faudrait un algorithme qui puisse résoudre le problème avec une complexité polynomiale, or cet algorithme n'a pas encore ete trouvé.
Pour que ISO-SOUS-GRAPHES soit dans P il faudrait un algorithme qui puisse vérifier une solution en temps polynomial avec une solution de taille raisonnable, il se trouve que c'est le cas comme le montre plus haut l'algorithme verifISG.
Un problème dans NP est necéssairement inclus dans les problèmes EXP, par conséquent ISO-SOUS-GRAPHES est aussi dans EXP.

2 Un problème de logique

Problem 7

Exercice 7.
En transformant la formule a en formule codé :
$[(1,), (2,-1,-3), (2,3,4), (-2,-1,-3), (-1,3,-4), (-3,4)]$
et par l'utilisaton de Solver on obtient que la formule est satisfaisable par $[1,2,-3,-4]$
soit a = vrai, b = vrai, c = faux, d = faux

En transformant la formule b en formule codé :
$[(-1,-2), (3,-4), (1,2,3), (1,2,-3,4), (1,-2,3), (1,-3), (-1,-3,4)]$
et par l'utilisaton de Solver on obtient que la formule est satisfaisable par $[1,-2,-3,-4]$ et par $[1,-2,3,4]$
soit a = vrai, b = faux, c = faux, d = faux
et a = vrai, b = faux, c = vrai, d = vrai

3 Une réduction polynomiale du problème de graphe au problème de logique

Problem 8

Exercice 8.a
voir **reduction** et la classe reduction de isg2sat.py, les clauses a et b étaient déjà implémentées, les clauses b et d on été implémentées.
Exercice 8.b
clause a: on parcourt le nombre de sommets de H. $clause_a = k$
clause b:
clause c: on parcourt le nombre de sommets de G puis de H. $clause_c = k * n$
clause d: on cherche le nombre d'arêtes du complémentaire de G multiplié par le nombre d'arêtes de H x2 car la formule génère 2 "blocs".
$clause_d = 2s(\frac{n(n-1}{2} - r) = s(n^2 - n - 2r)$
reduction est de complexité polynomiale car il s'agit de l'addition de 4 fonctions de complexité polynomiale.

Problem 9

Exercice 9.
voir **solveurISG** de isg2sat.py ainsi que **solution** de exSolveur.py

Problem 10

Exercice 10.
voir **solveurISG** ainsi que **decode** de isg2sat.py

Problem 11

Exercice 11.
Les tests se trouvent en grande partie dans exGraphe.py pour pouvoir plus tard comparer les résultats avec la solution **forcebrute**
tout comme pour force brute et pour les mêmes raisons (se référer a l'exercice 5) les résultats sont séparés en 2 fichiers: **benchmark_small_isg2sat.txt** et **benchmark_big_isg2sat.txt**

Problem 12

Exercice 12.
En comparant les résultats des Question 5. et 11. on se rend compte que l'approche d'un solveur SAT est plus rapide, même si la resolution est aussi en temps exponentiel cette approche permet d'aller plus loin dans le taille des graphes. Pour un temps autour de 10-30 secondes on peut avoir des graphes de cardinal **12** pour **forcebrute** contre **32** pour **SAT**.
Cela est probablement dû au fait que le solveur SAT est plus optimisé que la fonction force brute, si on possède donc des algorithmes déja bien optimisés et qu'il est possible de traduire le problème actuel vers celui dont on possède une bonne solution, on a tout intérêt à le faire.