TP Optimisation – session 3 – Algorithmes d'approximation

3 novembre 2025

Objectif d'apprentissage : cette séance vise à concevoir des instances pathologiques pour des algorithmes d'approximation vus en cours. On combinera deux approches :

— la réflexion issue de l'analyse de l'algorithme et de ses limites ;
— l'exploration empirique basée sur de la génération aléatoire.

C'est en combinant ces deux approches que l'on maximise les chances de trouver la pire instance pathologique.

1 Couverture gloutonne

Le professeur Sourire propose l'heuristique suivante pour résoudre le problème de la couverture de sommets. On choisit de façon répétée un sommet de plus haut degré, et on supprime toutes ses arêtes incidentes. En cas d'égalité (pour simplifier la recherche d'un pire cas), on choisira le sommet d'indice maximal. On s'arrête lorsqu'il n'y a plus d'arêtes. On souhaite construire une instance dont le facteur d'approximation soit le plus large possible.

Pour cela, on commence par créer un graphe quelconque qui facilitera l'implémentation de l'heuristique en fournissant la structure sur laquelle on travaille :

```
import networkx as nx
G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4])
print(G.nodes())
G.add_edges_from([(1, 2), (2, 3), (1, 3), (3, 4)])
print(G.edges())
print(G[3])
```

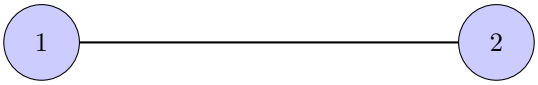
On va maintenant implémenter deux méthodes :

— `verifier_couverture` qui retourne vrai si une couverture donnée est valide pour un graphe donné ;
— `heuristique_Sourire` qui génère une couverture pour un graphe.

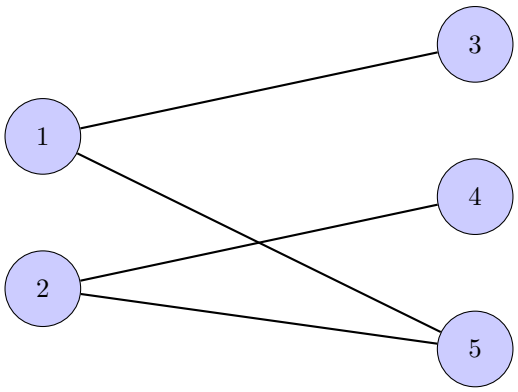
```
def verifier_couverture(G, C):
    #TODO

def heuristique_Sourire(G):
    #TODO

print(verifier_couverture(G, [1, 2, 4]))
print(heuristique_Sourire(G))
```

Vous pourrez utiliser la méthode `copy` pour copier le graphe afin de pour enlever des sommets sans modifier le graphe donné en argument.

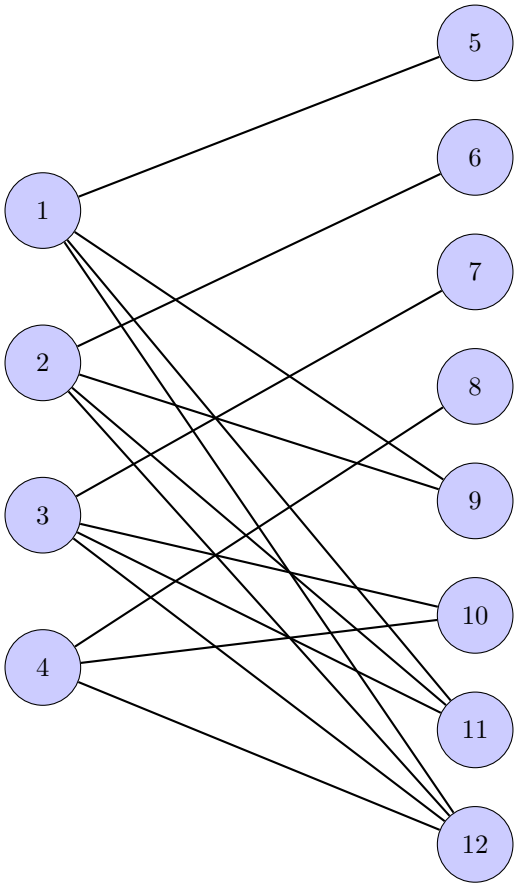
On va désormais construire une instance pour laquelle l'heuristique du professeur Sourire est arbitrairement mauvaise. L'instance aura une structure de graphe biparti avec n sommets à gauche. Ensuite, pour tout $1 \leq i \leq n$, on rajoute $k = \lfloor n/i \rfloor$ sommets à droite de degré i, tous connectés aux sommets de gauche qui ont le plus petit degré. Voici l'exemple pour $n = 1$:



Voici l'exemple pour $n = 2$:



Voici l'exemple pour $n = 4$:



De la même manière pour $n = 5$, on a 5 sommets à gauche et on rajoutera à droite 5 sommets de degré 1, 2 sommets de degré 2, 1 sommet de degré 3, 1 sommet de degré 4 puis 1 sommet de degré 5.

On peut vérifier que l'ensemble des sommets de gauche constitue une couverture et que l'heuristique retourne une couverture de taille supérieure (l'ensemble des sommets de droite) avec un ratio qui augmente avec n.

Pour quelle valeur de n obtient-on un facteur qui dépasse 2 ?

2 Couverture-Ensemble-Glouton

L'approche précédente consiste à concevoir spécifiquement une instance qui exploite une faille de l'heuristique. Une autre approche consiste à générer aléatoirement des instances pour étudier le comportement.

Nous allons maintenant générer des mots pour le problème de la couverture d'ensembles afin de déterminer s'il existe une instance dont le facteur d'approximation dépasse le 3.

Il faudra calculer l'optimal pour évaluer la qualité de ce que retourne l'algorithme d'approximation. Cette recherche ne sera donc utilisable que pour des petites instances (10 mots au maximum).

On va préparer la recherche en codant 3 parties : la construction d'une instance ; le calcul de la couverture optimale ; l'implémentation de l'algorithme d'approximation.

Pour la construction de l'instance, on générera n mots de 1 à k caractères. Le code suivant génère un mot aléatoire avec des caractères distincts :

```
import random, string

k = 10
''.join(set(random.choice(string.ascii_lowercase) for _ in range(k)))
```

Coder une fonction qui génère une instance paramétrée par n et k :

On peut désormais calculer une couverture optimale. On évaluera toutes les couvertures et on choisira la meilleure. Pour générer toutes les combinaisons, on peut s'appuyer sur la bibliothèque `itertools` :

```
import itertools
itertools.combinations(mots, 3)
```

Il ne reste plus qu'à implémenter Couverture-Ensemble-Glouton.

On peut commencer la recherche empirique. Il faut tâtonner en gérant les paramètres numériques suivant :

— le nombre de mots n ;
— la longueur des mots k ;
— le nombre de lettres distinctes (on ignorera ce paramètre).

Quelle est l'instance la plus petite qui dépasse $3/2$?