
Sorbonne Université

# Rapport de projet de MOGPL
# M1 Informatique

*Auteur :*
Herve Nguyen
28625990
Yuan Liu
21214576

*Groupe de TD :*
4

6 décembre 2023

# Résumé

Le projet cherche à améliorer l'algorithme de Bellman-Ford, une méthode pour calculer les plus courts chemins dans les graphes orientés pondérés. On voudrais améliorer le temps d'exécution de l'algorithme en introduisant une étape de prétraitement qui sélectionne un ordre pour les sommets, limitant alors les itérations de l'algorithme.

Le projet introduit le problème MVP (Minimum Violation Permutation) pour choisir un ordre total sur les sommets. La résolution de MVP peut être résolue avec une méthode gloutonne appelée GloutonFas, qui construit un ordre en se basant sur les degrés entrants et sortants.

On cherche alors à expérimenter avec l'algorithme GloutonFas pour avoir une idée de son comportement.

# Table des matières

# Chapitre 1

# Réponses aux questions du sujet

## 1.1 Préface

Le projet utilise trois fichiers, le fichier dictionnaireadjacenceorientepondere.py qui contient la classe permettant d'implémenter les graphes, le fichier notebook jupyter qui contient au code écris pour répondre aux questions et le fichier projet.py qui contient les algorithmes et autres codes lourds.

La fichier dictionnaireadjacenceorientepondere.py provient de l'UE de L3 "Algorithmique des Graphes" de l'Université Gustave Eiffel que un de notre membre du binôme a suivi l'année dernière (Hervé NGUYEN), néanmoins nous avons modifié par la suite cette classe assez lourdement afin d'optimiser les temps d'exécutions (notemment au niveau des degrés entrant et sortant), de ce fait l'API reste la même mais l'implémentation est différente. Nous précisons cela car il est possible qu'un autre élève qui provient aussi de la L3 Informatique de l'UGE utilise également cette classe pour ce projet.

Nous utilisons

## 1.2 Question 1

Nous avons codé Bellman-Ford à travers la fonction bellmanford(G, source). Celle-ci renvoie un tableau distances, un tableau de parents (qui peut-être interprété comme l'arborscence des plus court chemins) et le nombre d'itération nécéssaires.

Nous avons également écris une fonction obtenirChemin(source, destination, parent) pour les prochaines questions, qui permet de parcourir de manière inversée le tableau de parents pour déduire le plus court chemin entre la source et la destionation (si ce dernier est accessible à partir de la source).

## 1.3 Question 2

L'algorithme a été repris à la lettre avec GloutonFas(G).

Cependant pour le graphe ci-dessus contraitement à l'ordre obtenu dans le sujet s = [4, 6, 5, 8, 7, 1, 2, 3] nous obtenons un ordre légèrement différent s = [4, 6, 5, 7, 1, 8, 2, 3].

Un explication que l'on a trouvé est le fait que l'ordre de sommets tirés des listes des puits et des sources est différent. Nous obtenons alors un ordre tout autant valide vis-à-vis de l'algorithme GloutonFas mais pas nécésairement le même.

## 1.4 Question 3

Inspiré du graphe de base de l'exemple pour GloutonFas du sujet, nous avons tiré manuellement et aléatoirement des poids dans l'intervalle [-10,10] pour obtenir $G_1, G_2, G_3, H$.

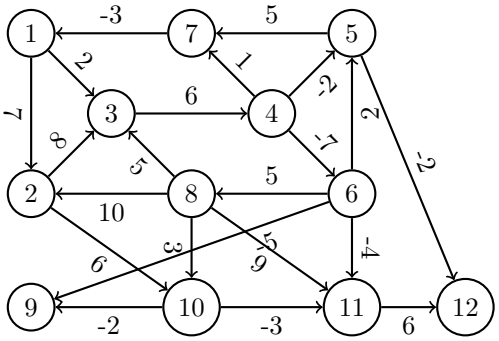


FIGURE 1.1 – Graphe $G_1$

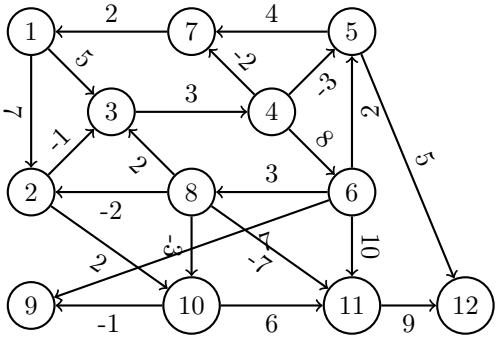


FIGURE 1.2 – Graphe $G_2$

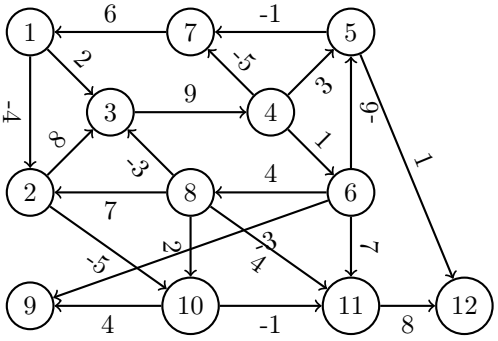


FIGURE 1.3 – Graphe $G_3$

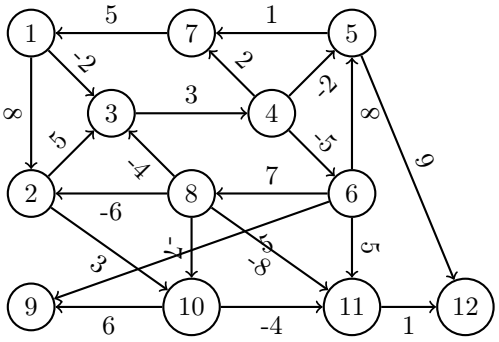


FIGURE 1.4 – Graphe $H$

## 1.5 Question 4

Pour obtenir l'union des arborescences des plus courts chemins de chaque graphe, nous utilisons obtenirPlusCourtChemins(source, parent, sommets) pour obtenir l'ensemble des plus courts chemins d'un graphe depuis le sommet "source" grâce au tableau parents (obtenu avec Bellman-Ford) et à la liste des sommets.

T est alors obtenu en créant un graphe qui contient tous les arcs contenus dans les chemins les plus court des graphes $G_1, G_2, G_3$.

## 1.6 Question 5

En appliquant GloutonFas à T, on obtient donc [1, 2, 3, 4, 6, 5, 7, 8, 10, 9, 11, 12].

## 1.7 Question 6

En appliquant Bellman-Ford à H avec l'ordre grâce à bellmanford_ordre(G, source, ordre), l'algorithme termine en 2 itérations.
Pour comparer, sans ordres particulier l'algorithme termine en 7 itérations.

## 1.8 Question 7 et 8

Avec un ordre tiré aléatoirement, on remarque que le nombre d'itération a tendence à être supérieur à l'ordre obtenu via GloutonFas. Sur un tirage nous avons obtenu 6 itérations.

Avec comparaisonOrdre(G, source, ordre, n), à partir d'un graphe G, un ordre donné et un nombre n d'ordre aléatoire à générer, nous pouvons regarder le nombre de fois où un ordre aléatoire finit l'algorithme avec un nombre d'itération qui est soit inférieur, égal, supérieur à l'ordre fourni. La fonction affiche la moyenne des itérations des ordres aléatoires, le nombre d'itération que l'ordre fourni a pris, et le nombre d'ordres aléatoires qui ont eu un nombre d'itération soit inférieur, égal ou supérieur à ce dernier

Pour une exécution particulière avec 10000 ordres générés sur le graphe H, nous avons obtenu :

— 5.0111 en moyenne pour les ordres aléatoires
— 2 pour l'ordre obtenu avec GloutonFas
— 9998 ordres qui ont pris plus d'itération que l'ordre généré par GloutonFas
— 2 ordres qui ont autant itération que l'ordre généré par GloutonFas
— aucun ordre ayqnt moins d'itération que l'ordre généré par GloutonFas

## 1.9 Question 9

Pour traiter cette question, on représente un graphe "modèle" / "template" avec une liste de sommet et d'arcs non pondérés.
Avec ce modèle on peut donc créer autant d'instances avec des fonctions de poids tirés aléatoirement.

On cherchera à construire des modèles ayant 20 sommets.

Par curiosité, en utilisant les mêmes test que ci-dessus, on peut remarquer que assez souvent pour un modèle donné, l'ordre obtenu via l'algorithme permet d'obtenir le meilleur nombre d'itération parmis 10000 autres ordres aléatoires.

Il y aussi des modèles générés où l'on peut remarquer qu'un grand nombre de d'ordre est de même qualité (la meilleure) que celui calculé. Certains ce ces modèles sont en réalité des modèles triviaux obtenus aléatoirement où on a 1 seule itération car les noeuds accessibles à partir de la source sont seulement les successeurs de la source.
Par contre, il est plutôt rare de trouver des ordres aléatoire meilleurs que celle calculés mais peuvent apparaître.

## 1.10 Question 10

Pour tenter de trouver si il y a une certaine corrélation, nous utilisons le "modèle" de graphe de la question 3 et aussi quelques autres modèles générés aléatoirement.

On veut alors calculer la moyenne des itérations (avec n échantillons donc n ordres calculés, pour un nombre de instances fourni à l'algorithme GloutonFas, 100 pour le modèle prédéfini et 10 pour ceux qui sont générés par soucis de temps d'exécution).

On va faire varier le nombre de graphe fourni à GloutonFas de 3 à 15.

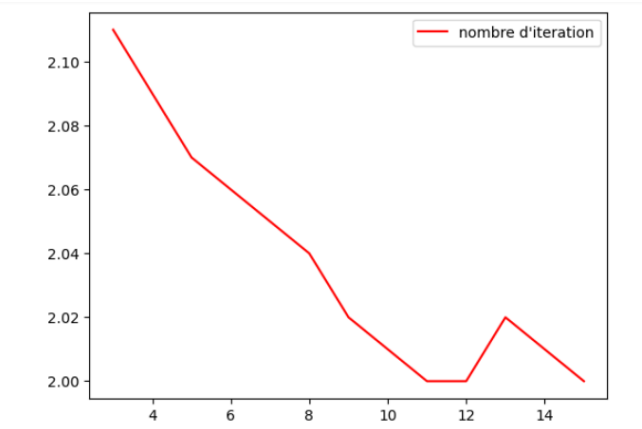


FIGURE 1.5 – Moyenne des itération selon les nombre de graphes fourni à GloutonFas pour le modèle de la question 3

Les l'allure des courbes pour les modèles générés aléatoirement varient grandement d'une génération à une autre, certaines peuvent être ascendante ou bien descendantes

Certaines peuvent même être plates comme dans la fig 1.15, ce qui pourrait être expliqué par la simplicité des chemins depuis la source (le sommet 0) donc un ordre optimal peut-être alors déduis facilement avec un nombre relativement bas d'instances fournis à GloutonFas.

Cependant nous ne pouvons pas déduire grand chose étant donné la volatilité des résultats parmis les nodèles générés.

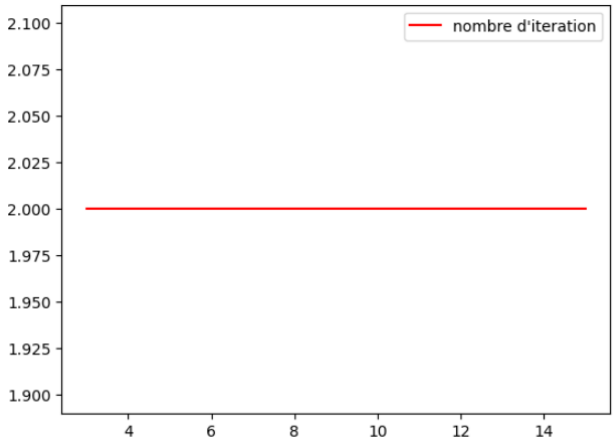


FIGURE 1.6 – idem fig 1,5 mais différentes fonctions de poids pour H et les graphes d'apprentissages

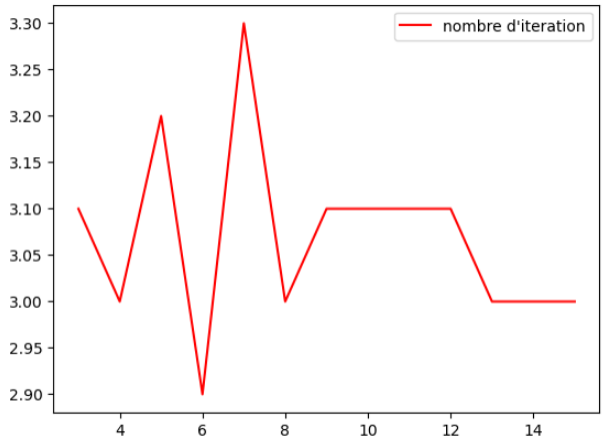


FIGURE 1.7 – idem fig 1.5 pour un autre modèle généré aléatoirement

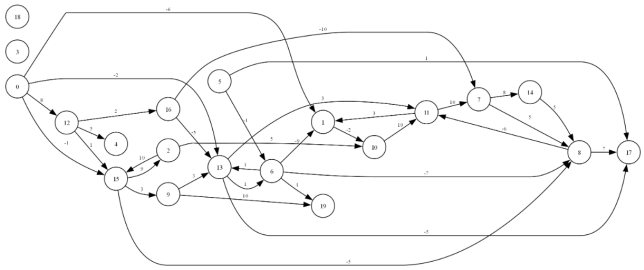

FIGURE 1.8 – Graphe H utilisé dans le cas de fig 1.7

Par contre pour le modèle prédéfini de la fig 1.5 et 1.6, on remarque que l'allure descendante est plus commune que les autres.

On pourrait conjecturer que le nombre de graphe fourni à l'algorithme permettrait de minimiser les chances que l'ordre obtenu entraine un nombre d'itération bien plus grand que l'ordre optimal.

Mais étant donné le nombre d'échantillon et le nombre de graphe fourni qui peuvent être relativement, nos résultats ne nous permettent pas de construire un avis confiant.

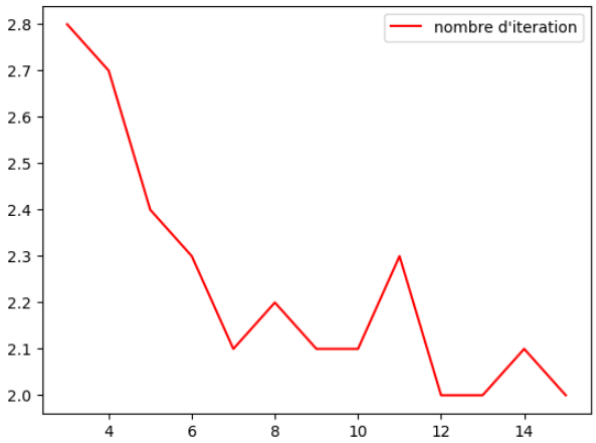


FIGURE 1.9 – idem fig 1.5 pour un autre modèle généré aléatoirement

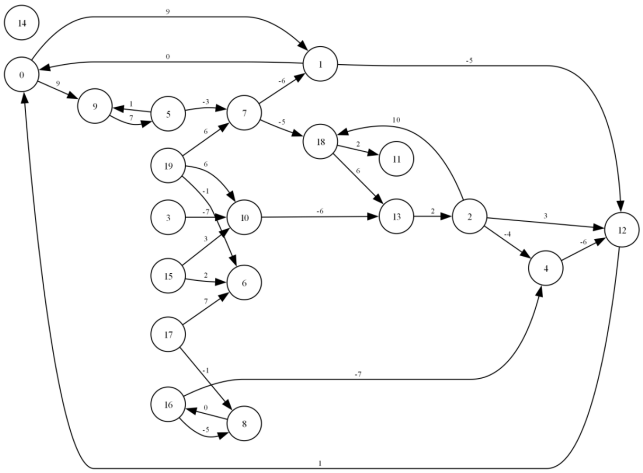


FIGURE 1.10 – Graphe H utilisé dans le cas de fig 1.9

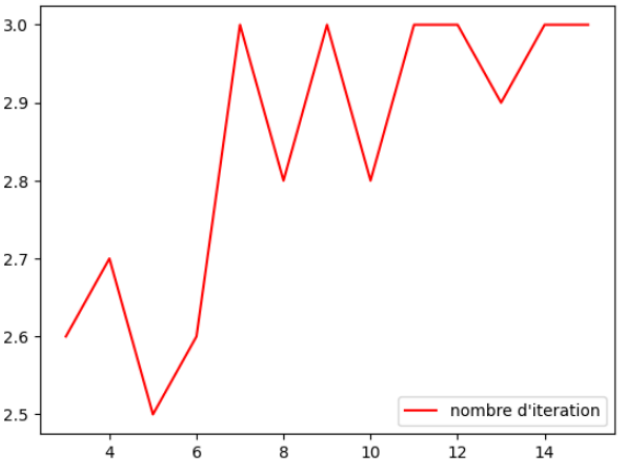


FIGURE 1.11 – idem fig 1.5 pour un autre modèle généré aléatoirement

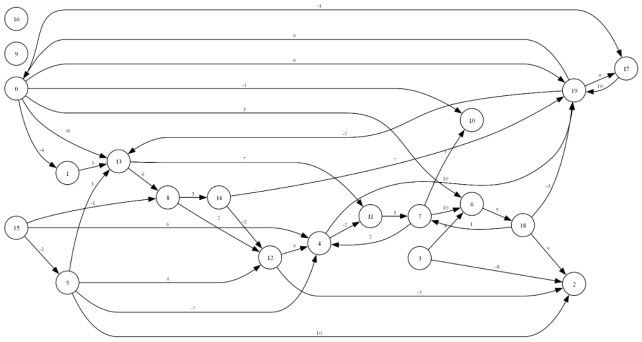

FIGURE 1.12 – Graphe H utilisé dans le cas de fig 1.11

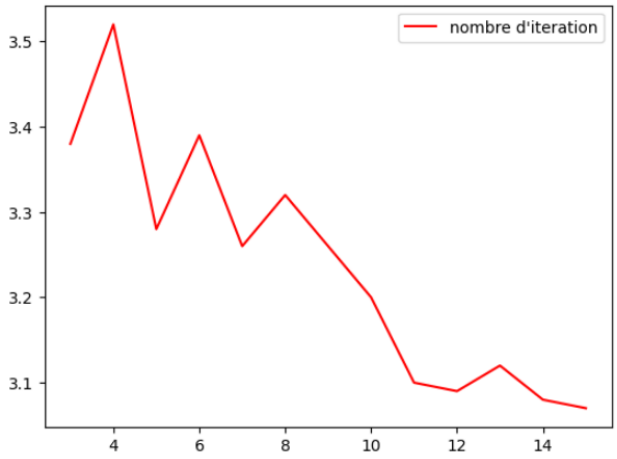


FIGURE 1.13 – idem fig 1.5 pour un autre modèle généré aléatoirement

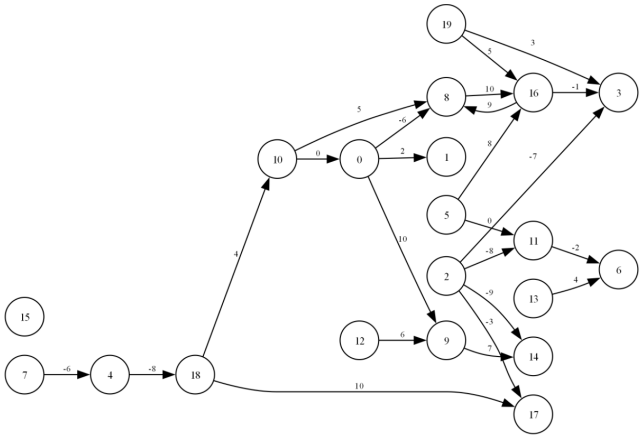

FIGURE 1.14 – Graphe H utilisé dans le cas de fig 1.13

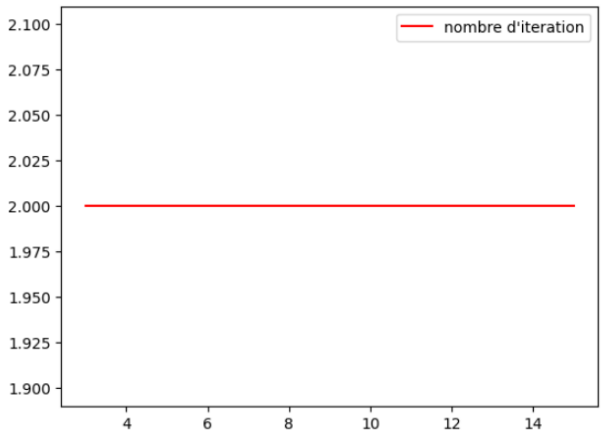


FIGURE 1.15 – idem fig 1.5 pour un autre modèle généré aléatoirement

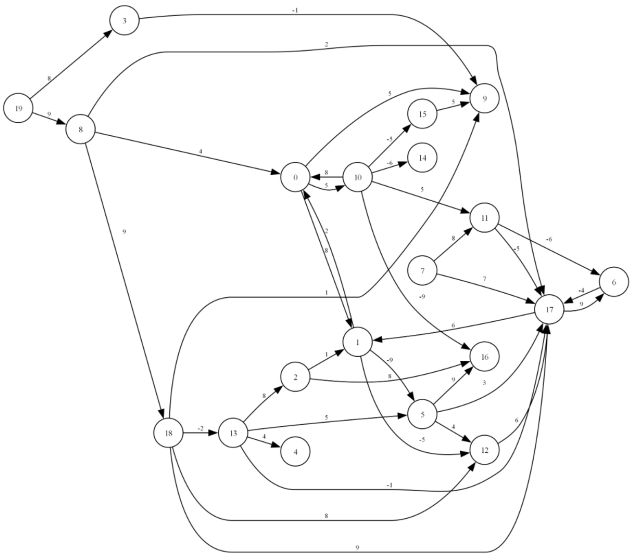

FIGURE 1.16 – Graphe H utilisé dans le cas de fig 1.15

## 1.11 Question 11

En moyenne, en choissisant un des pire cas en terme de source (c'est-à-dire les sommets des premiers niveaux, par exemple le sommet 0, car ce sont eux qui aurant les plus long "plus court chemins"), le motif d'instances suivant nous donne en moyenne environ 1250 itérations de Bellman-Ford.

Une exécution de l'algorithme de Bellman-Ford sans ordre particulier prends environ 13 secondes.
Déterminer un ordre nous prend environ 1 minute pour cette famille d'instance avec 3 instances fournies, 20 pour une seule instance fournie, l'ordre obtenu permet d'avoir 2 itérations possible avec 50ms en temps d'exécution environ dans les deux cas.

La méthode de prétraitement pour traiter une seule instance de cette famille n'est pas pertinente car elle implique donc au moins deux exécution de Bellman-Ford standard

Cependant elle est très intérêssante quand on est dans le cas de figure où l'on veut traiter au moins deux instances ou plus de cette famille. En effet, même en utilisant seulement une instance pour obtenir un ordre, cet ordre là nous permet d'avoir 2 itérations (dans nos tests nous avons toujours eu 2 itérations avec un ordre généré par GloutonFas dans ce cas spécifique),

Le gain en terme de temps d'exécution est alors extrêmement important dans le cas ou l'on va traiter un grand nombre de graphes de cette familles.

# Chapitre 2

# Bilan

En conclusion, on relève une certaine pertinence de cette méthode, particulièrement dans les cas où plusieurs instances d'un graphe sont traitées, donnant alors un gain important en termes de temps d'exécution.

Cependant, il faut noter que cette méthode n'est pas infaillible, en effet, lorsque l'on traite une instance de graphe unique, il n'est pas pertinent de déterminer un ordre. De plus, la nature variable des résultats obtenus dans la question 10 avec les modèles générés aléatoirement pourrait aussi mettre à l'évidence qu'il y a une erreur au niveau de l'implémentation ou bien de la méthodologie des tests.
Mettant alors à l'évidence une certaine nécéssité pour nous d'approfondir notre compréhension de cette méthode.

Malgré cela, les résultats peuvent suggérer que cette approche peut offrir de vraies améliorations dans des contextes où le traitement de multiples instances de graphes est requis comme nous l'avons vu dans la question 11.

# Annexe

## 1 Test réalisés

### 1.1 Tests éffectués dans la question 8 et 9

Ces tests, définis par la fonction comparaisonOrdre, cherche à évaluer les performances de l'algorithme de Bellman-Ford en fonction de l'ordre dans lequel les sommets du graphe sont traités. Il compare les résultats obtenus en utilisant un ordre spécifique (fourni en argument) avec ceux obtenus en utilisant des ordres aléatoires.

La fonction génère n ordres aléatoires, applique l'algorithme de Bellman-Ford à chaque ordre, puis compare le nombre d'itérations nécessaires dans chaque cas. Les résultats sont classifiés en trois catégories : "better" (meilleur), "eq" (égal), et "worse" (pire), en fonction du nombre d'itérations par rapport à l'ordre spécifique. Les statistiques telles que la moyenne (avg), le meilleur résultat (best), et le nombre d'itérations avec l'ordre spécifique (gloutonFas) sont ensuite affichées.

### 1.2 Temps d'exécution

Le temps d'exécution est mesuré à travers jupyter notebook, le notebook mesure le temps d'exécution d'une cellule avec la clause %%time et on ajoute à l'intérieur de cette cellule les opérations que l'on veut mesurer. À l'occurence, l'application de l'algorithme GloutonFas, l'application de l'algorithme Bellman-Ford avec et sans ordre.

### 1.3 Corrélation entre nombre d'instances fournies à GloutonFas et nombre d'itérations

La fonction correlation_nb_instance prend en charge ce test. À partir d'un modèle de graphe, on génère un graphe H. Puis Pour un nombre d'instances i fournies à GloutonFas allant de 2 à nb_instance_max (par défaut 15), on génère nb_enchantillon fois un nouvel ordre puis on applique Bellman-FOrd avec cet ordre sur H. Le nombre d'itération sera évalué et on fera la moyenne par rapport au nombre d'échantillons.

Puis on représente sur un graphique ces moyenne par rapport au nombre d'instances.

## 2 Abus de langage

### 2.1 Modèle

En général, en dehors du contexte du sujet. On parle de simplement de l'ensemble de sommet et des arcs orienté sans pondération.

### 2.2 Apprentissage

On parle du procédé affectué pour obtenir un ordre, étant donné la similarité avec l'apprentissage statistique. Cependant nous sommes conscient que cela ne pourrait ne pas être un "apprentissage".