Compte rendu projet SDAA

Lucas MICOUD et Nino DAUVIER

27 mars 2021

Objectifs

L'objectif de ce projet est d'implémenter une classe de graphe dotée des fonctionnalités classiques et d'algorithmes de plus court chemin. On cherchera ensuite à tester la rapidité de ces algorithmes et à les comparer avec les algorithmes de la classe DiGraph du module *networkx* de Python. Finalement on les appliquera au *Reddit Hyperlink Network* pour voir une application de la classe.

1 Retour d'expérience

Ce projet s'est déroulé en trois parties. La première a été de coder les fonctions demandées et de les tester sur des exemples simples, de sorte à ce que le résultat rendu soit juste. Dans un second temps, on cherche à optimiser les algorithmes, à la fois d'un point de vue théorique que d'un point de vue pratique en prenant en compte les particularités du langage Python. Par exemple pour parcourir vider une liste en tirant à chaque fois une valeur aléatoire, il vaut mieux faire un *shuffle* puis n fois *pop* plutôt que n fois *choice* puis *remove*. Enfin, nous avons utilisé nos algorithme sur un le *Reddit Hyperlink Network*. Toutes les figures se trouvent en annexe, et sont tracées en échelle log-log.

Ce projet nous a aussi permis de nous servir du gestionnaire de version git (associé à github), grâce auquel nous avons pu travailler chacun sur une partie du projet sans se gêner.

2 Évaluations de la rapidité des algorithmes

Une fois nos algorithmes implémentés, nous les avons évalués et comparés les uns par rapport aux autres selon les critères suivants :

- Le temps de génération de graphes aléatoires en fonction du nombre d'arêtes avec un nombre de noeuds fixé (figure 1).
- Le temps de génération de graphes aléatoires en fonction du nombre de noeuds avec une proportion d'arêtes fixée (figure 2).
- Le temps de calcul du plus court chemin avec l'algorithme de Dijkstra sur des graphes en fonction du nombre d'arêtes avec un nombre de noeuds fixé (figure 3).
- Le temps de calcul du plus court chemin avec l'algorithme de Dijkstra sur des graphes en fonction du nombre fonction du nombre de noeuds avec une proportion d'arêtes fixée (figure 4).

De plus avec les groupes qui le souhaitaient, nous avons mis en compétition les algorithmes, avec en plus les algorithmes de la librairie *networkx* (figure 5). Cette initiative a eu un fort intérêt car pour que les tests se fassent en un temps court, nous avons employé des threads : l'algorithme de chacun s'exécute dans un thread, qui renvoie lorsqu'il a fini le temps de traitement.

On peut faire quelques commentaires sur ces images. Tout d'abord on remarque que tous les tracés sont des droites en échelle log-log. Cela signifie que nos algorithmes sont de complexité polynomiale, et donc pas trop mauvais.

Le temps de génération de graphe augmente rapidement lorsque l'on cherche à générer des graphes qui s'approchent de graphes complets.

D'autre part, on voit que notre algorithme de Dijkstra implémenté avec des tas est bien plus performant que la version classique, et qu'il s'approche de la version du module *networkx*. On a aussi constaté que la durée de l'algorithme de Dijsktra dépend énormément de la forme du graphe. Cet effet a été nuancé dans les tests en faisant des moyennes sur plusieurs itérations.

3 Résultats sur le *Reddit Hyperlink Network*

Notre traitement du *Reddit Hyperlink Network* nous a permis de constater que 10380 subreddits n'ont aucun lien avec d'autres subreddits.

On a aussi trouvé que les dix subreddits les plus liés aux autres sont :

1. bestof, avec 3111 liens
2. subreddutdrama, avec 2659 liens
3. titlegore, avec 2468 liens
4. drama, avec 1179 liens
5. switcharoo, avec 916 liens
6. shitredditsays, avec 879 liens
7. botsrights, avec 785 liens
8. shitpost, avec 782 liens
9. hailcorporate, avec 772 liens
10. shitamericanssay, avec 767 liens

De plus, on a calculé que les 2% des subreddits les plus liés aux autres représentent 47.50% des liens entre subreddits.

Enfin, on a trouvé que la distance de disney à vegan est de 2 liens et celle greenbaypackers à missouripolitics est de 3 liens.

4 Annexe

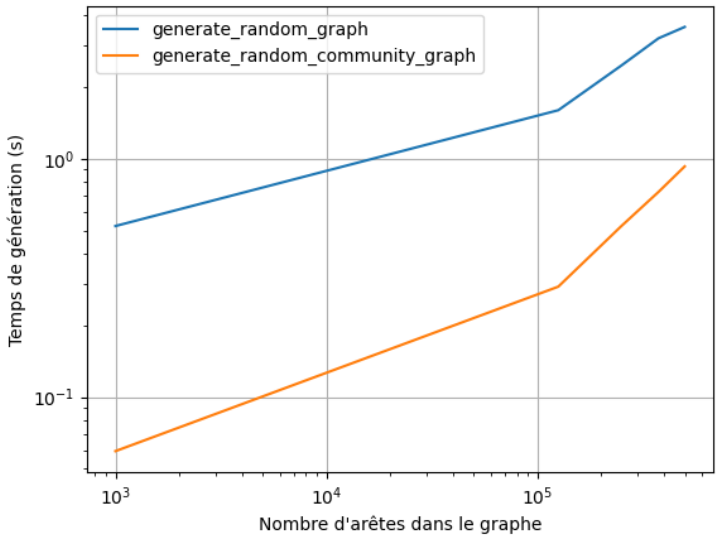

FIG. 1 – ***Temps de génération du graphe en raison du nombre d'arêtes dans celui-ci.*** *Le nombre d'arêtes va du nombre minimal pour que le graphe soit connexe, au nombre d'arêtes pour lequel le graphe est complet. Figure réalisée avec un graphe 1000 noeuds.*

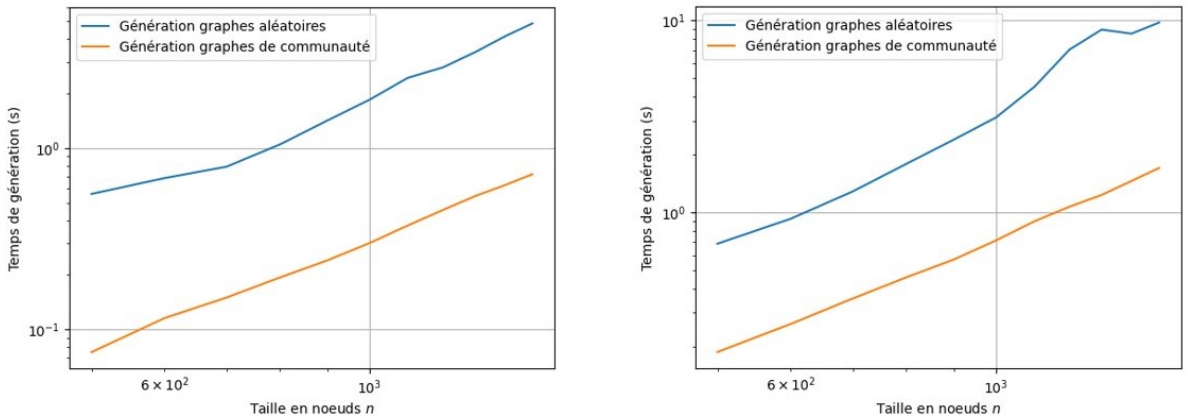

FIG. 2 – ***Temps de génération du graphe en raison du nombre de noeuds.*** *A gauche une proportion d'arêtes de 0.3, à droite de 0.7 (par rapport au nombre d'arêtes dans un graphe complet).*

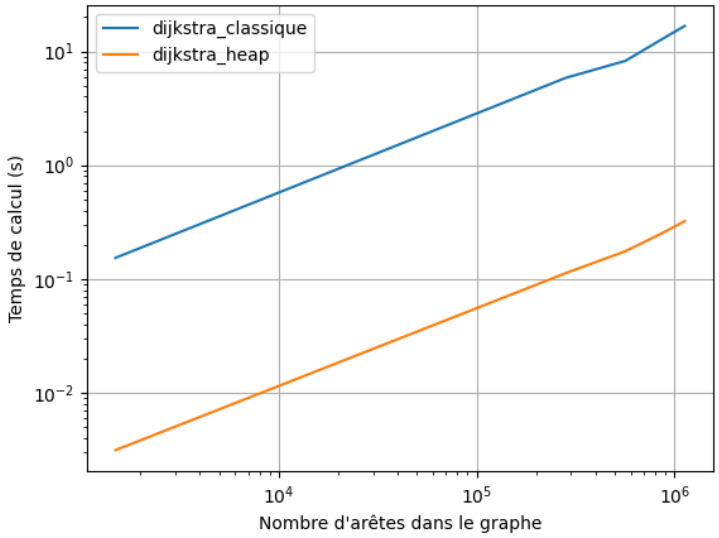

FIG. 3 – ***Temps de calcul de Dijkstra en raison du nombre d'arêtes dans celui-ci.*** *Le nombre d'arêtes va du nombre minimal pour que le graphe soit connexe, au nombre d'arêtes pour lequel le graphe est complet. Figure réalisée avec un graphe 1500 noeuds*

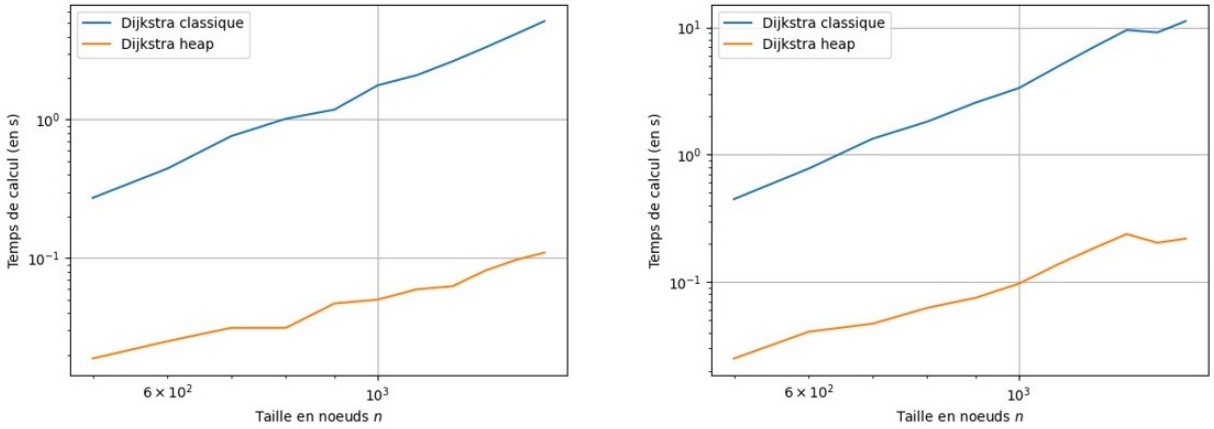

FIG. 4 – ***Temps de calcul des algorithmes de Dijkstra en raison du nombre de noeuds dans le graphe.*** *A gauche une proportion d'arêtes de 0.3, à droite de 0.7 (par rapport au nombre d'arêtes dans un graphe complet).*

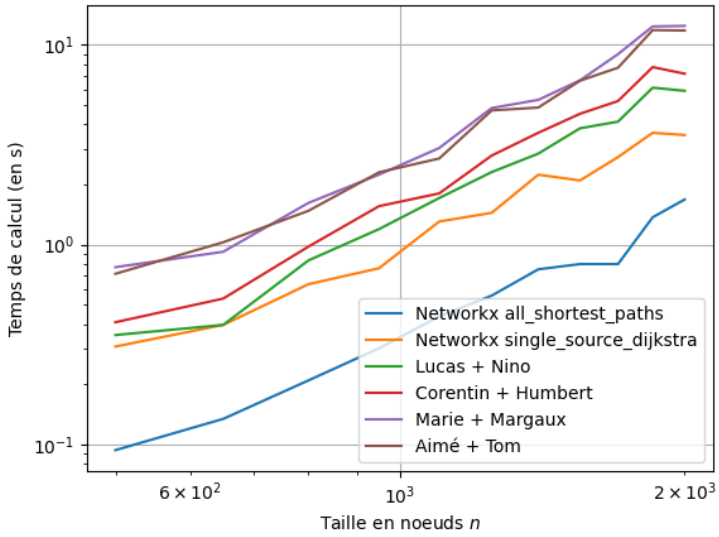

FIG. 5 – ***Mise en compétition des algorithmes des étudiants et de networkx.*** *On peut voir plusieurs choses : premièrement Dijkstra n'est pas l'algorithme le plus efficace. Ensuite tous les algorithmes des étudiants sont moins efficaces que le Dijkstra de networkx. Enfin, parmi ceux qui ont bien voulu participer à la compétition, notre algorithme est le plus efficace.*