Rapport de Projet

MOGPL

MU4IN200

Encadrant: Patrice Perny

Juliette LING

Jérémy DUFOURMANTELLE

Introduction

Ce projet a été réalisé dans le cadre de l'UE de MOGPL du Master ANDROIDE par Jérémy DUFOURMANTELLE et Juliette LING. Dans ce projet, nous considérons des multigraphes orientés pondérés par le temps. A travers de projet, nous cherchons à trouver des chemins d'arrivée au plus tôt (appelé chemin de type I), des chemins de départ au plus tard (appelé chemin de type II), des chemins de durée minimale (appelé chemin de type III) et des plus courts chemins (appelé chemin de type IV). Dans une première partie, nours montrerons que les assertions des chemins de type I, II, III et IV sont vraies. Puis nous proposerons et étudierons la complexité des algorithmes permettant de trouver des solutions à ces chemins. Nous analyserons ces résultats et pour finir nous proposerons un algorithme alternatif du chemin de type IV.

Question 1

Assertion 1: Un sous-chemin préfixe d'arrivée au plus tôt peut ne pas être un chemin d'arrivée au plus tôt.

Pour démontrer que cette assertion soit vraie, prenons un chemin P partant du sommet a au sommet k dans le multigraphe de l'exemple 1. D'après la définition du chemin d'arrivée au plus tôt, c'est-à-dire un chemin de type I, il faut que P soit défini tel qu'on a la proposition suivante qui soit vérifiée : $fin(P) = min\{fin(P') : P' \in P(a, k, [t_\alpha, t_\omega]]\}$.

Ainsi, parmi toutes les chemins possibles du sommet a au sommet k, nous avons P le chemin du type I passant par $P_1 = (a, b, 2, 1)$, $P_2 = (b, g, 3, 1)$ et $P_3 = (g, k, 6, 1)$ avec $fin(P) = t_3 + 1 = 6 + 1 = 7$

Nous nous intéressons à l'ensembles des sous-chemins préfixe de P, en particulier les chemins du sommet a vers le sommet b, noté P_{ab}. Le chemin d'arrivé au plus tôt de P_{ab} est le chemin passant par $P_{ab_1} = (a, b, 1, 1)$ avec $fin(P_{ab_1}) = t_1 + 1 = 1 + 1 = 2$.

Nous remarquons que $fin(P_{ab_1}) = 2 < fin(P_1) = 2 + 1$, alors que le chemin P ne passe pas par P_{ab_1}, d'où l'assertion vérifiée.

Assertion 2 : Un sous-chemin postfixe d'un chemin de départ au plus tard peut ne pas être un chemin de départ au plus tard.

Pour démontrer que cette assertion soit vraie, prenons un chemin du sommet a au sommet l du mutligraphe de l'exemple 1. D'après la définition du chemin de type II, nous avons le chemin du type II qui vérifie : $debut(P) = max\{debut(P') : P' \in P(a, k, [t_\alpha, t_\omega]]\}$

Parmi toutes les chemins possibles du sommet a au sommet l, nous avons P le chemin du type II qui passe par $P_1 = (a, c, 4, 1)$, $P_2 = (c, h, 6, 1)$, $P_3 = (h, i, 7, 1)$ et $P_4 = (i, l, 8, 1)$ avec $debut(P) = t_1 = 4$.

Nous nous intéressons à l'ensembles des sous-chemins postfixe de P du sommet i au sommet l, noté P_{il}. Le chemin de départ au plus tard de P_{il} est le chemin $P_{il_1} = (i, l, 9, 1)$ avec $debut(P_{il_1}) = 9$. Or $debut(P_{il_1}) = 9 > debut(P_4) = 8$ et P est un chemin de départ au plus tard ne passant pas par P_{il_1}, donc l'assertion 2 est vérifiée.

Assertion 3 : Un sous-chemin d'un chemin le plus rapide peut ne pas être un chemin le plus rapide.

Nous rappelons qu'un chemin est appelé chemin le plus rapide (ou type III)s'il vérifie : $duree(P) = min\{duree(P') : P' \in P(a, k, [t_\alpha, t_\omega]]\}$.
Soit P, le chemin le plus rapide du sommet a au sommet k du multigraphe de l'exemple 1.

Plus précisement, P passe par $P_1 = (a, c, 4, 1)$, $P_2 = (c, h, 6, 1)$ et $P_3 = (h, k, 7, 1)$ avec $duree(P) = fin(P) - debut(P) = (7 + 1) - 4 = 4$.

Intérressons nous aux sous-chemins de P du sommet a au sommet h, noté P_{ah}. Le chemin le plus rapide de P_{ah} est le chemin passant par $P_{ah_1} = (a, b, 2, 1)$ et $P_{ah_2} = (b, h, 3, 1)$ avec $duree(P_{ah_1}, P_{ah_2}) = (3 + 1) - 2 = 2$.

Nous avons $duree(P_{ah_1}, P_{ah_2}) = 2 < duree(P_1, P_2) = (6 + 1) - 4 = 3$. Pourtant le chemin P est un chemin de départ au plus tard ne passant pas par ces arcs, donc l'assertion 3 est vérifiée.

Assertion 4: Un sous-chemin d'un plus court chemin peut ne pas être un plus court chemin.

Pour démontrer que cette assertion est vraie, prenons le chemin du sommet a au sommet l du mutligraphe de l'exemple 1. Un chemin est dit de plus court chemin (ou type IV) s'il vérifie: $dist(P) = min\{dist(P') : P' \in P(a, k, [t_\alpha, t_\omega]\}$

Nous avons donc le chemin P le plus court passant par: $P_1 = (a, f, 3, 1)$, $P_2 = (f, i, 5, 1)$ et $P_3 = (i, l, 8, 1)$ avec $dist(P) = 1 + 1 = 2$.

Intérressons nous aux sous-chemin de P passant par le sommet a au sommet i, noté P_{ai}. Le plus court chemin de P_{ai} est le chemin passant par $P_{ai_1} = (a, i, 10, 1)$ avec $dist(P_{ai_1}) = 1$.

Nous remarquons que $dist(P_{ai_1}) = 1 < dist(P_1, P_2) = 2$. Pourtant le chemin $P_{ai_1} = (a, i, 10, 1)$ n'est pas un sous-chemin parmi les chemins possibles des plus court chemin du sommet a au sommet i, donc l'assertion 4 est vérifiée.

Question 2

Dans cette partie, nous considérons un multigraphe orienté $G = (V, E)$ et le graphe transformé $G' = (V', E')$.

Algorithme permettant de calculer le chemin de type I :

Entrée : un graphe $G' = (V', E')$, deux sommets $a, b \in V$
Sortie : un chemin P d'arrivée au plus tôt du sommet a au sommet b

Début :
- Recherche des sommets *depart* et *arrivee* avec $depart = (a, min\{t/(a,t) \in V'\})$ et $arrivee = (b, max\{t/(b,t) \in V'\})$.
- Tranformation du graphe $G' = (V', E')$ en graphe $G'' = (V'', E'')$ avec $V'' = V'$ et $\forall((u,t_u),(v,t_v),t) \in E'$, si $u \neq v \neq b$ alors $((u,t_u),(v,t_v),(t_v - t_u)) \in E''$ sinon on a $u = v = b$ et $((b,t_u),(b,t_v),t) \in E''$.
- Application de l'algorithme de Dijkstra.
Fin

Explication La transformation de G'' value les arcs avec toutes les "pertes de temps" entre les différents sommets de G', hormis celle de l'arrivé. Cela rend la recherche avec l'algorithme de Dijkstra optimale.

Algorithme 1 permettant de calculer le chemin de type II :

Entrée : un graphe $G' = (V', E')$, deux sommets $a, b \in V$
Sortie : un chemin P de départ au plus tard du sommet a au sommet b

Début :
- Tranformation du graphe $G' = (V', E')$ en graphe $G'' = (V'', E'')$ avec $V'' = V'$ et $\forall((u,t_u),(v,t_v),t) \in E'$, si $u = v = a$ alors $((u,t_u),(v,t_v),0) \in E''$ sinon $((u,t_u),(v,t_v),(t_v - t_u)) \in E''$
- Recherche des sommets *depart* et *arrivee* avec $depart = (a, min\{t/(a,t) \in V'\})$ et $arrivee = (b, max\{t/(b,t) \in V'\})$.
- Application de l'algorithme de Dijkstra.
Fin

Explication La transformation de G'' value les arcs de départ avec des 0 pour permettre la recherche la plus lointaine au sein du même sommet. Néanmoins l'application de l'algorithme de Dijkstra reste optimale car nous considérons ici le graphe comme un arbre enraciné, donc sans cycle et qui a comme noeud initial a. L'algorithme va donc préférer explorer d'abord les fils avec comme valeur 0 afin de partir au plus tard.

Algorithme 2 permettant de calculer le chemin de type II :

Entrée : un graphe $G' = (V', E')$, deux sommet $a, b \in V$
Sortie : un chemin P de départ au plus tard du sommet a au sommet b

Début :
- Tri des indices des sommets de départ du plus tard au plus tot avec un algorithme de tri rapide tel que : $\forall (s,t) \in V'$ et $s = a$, on obtient une liste de la forme : $[(a,t_1),(a,t_2),...,(a,t_n)]$, ou $\forall t_i, t_j, i < j$ alors $t_i > t_j$
- Recherche du sommet *arrivee* avec $arrivee = (b, max\{t/(b,t) \in V'\})$
- Répéter algorithme de BFS sur (a,t_1),...,(a,t_n) et s'arrête lorsque un chemin est trouvé pour un sommet (a,t_i) ou si aucun chemin n'est valide.
Fin

Explication Cet algorithme est optimal car nous appliquons une recherche BFS en partant des noeuds les plus prometteurs au moins prometteurs grâce au tri rapide.

Algorithme 1 permettant de calculer le chemin de type III :

Entrée : un graphe $G' = (V', E')$, deux sommets $a, b \in V$
Sortie : un chemin P le plus rapide du sommet a au sommet b

Début :
- Tranformation du graphe $G' = (V', E')$ en graphe $G'' = (V'', E'')$ avec $V'' = V'$ et $\forall ((u,t_u),(v,t_v),t) \in E'$, si $u = v = a$ ou si $u = v = b$ alors $((u,t_u),(v,t_v),0) \in E''$ sinon $((u,t_u),(v,t_v),(t_v - t_u)) \in E''$
- Recherche des sommets *depart* et *arrivee* avec $depart = (a, min\{t/(a,t) \in V'\})$ et $arrivee = (b, max\{t/(b,t) \in V'\})$.
- Application de l'algorithme de Dijkstra.
Fin

Explication La transformation de G'' value les arcs de départ et d'arrivé à 0 et value les autres arcs avec la différence de temps entre les noeuds finaux et initiaux. Cela permet une recherche de chemin du Type 4 par Dijkstra optimale car on s'interesse à minimiser uniquement le trajet après le départ et avant l'arrivée.

Algorithme 2 permettant de calculer le chemin de type III :

Entrée : un graphe $G' = (V', E')$, deux sommets $a, b \in V$
Sortie : un chemin P le plus rapide du sommet a au sommet b

Début :
- Tri des indices des sommets de départ en fonction de leur rapidité avec les différents sommet d'arrivé avec un algorithme de tri rapide tel que : $\forall (s,t) \in V'$ et $s = a$, on obtient une liste de la forme : $[(a,t_1),(a,t_2),...,(a,t_n)]$, ou $\forall t_i, t_j, i < j$ alors $t_i > t_j$

- Répéter algorithme de BFS sur (a, t_1),...,(a, t_n) et s'arrête lorsque un chemin est trouvé pour un (a, t_i) ou si aucun chemin n'est valide.
Fin

Explication Cet algorithme est optimal car nous appliquons une recherche BFS en partant des noeuds les plus prometteurs au moins prometteurs grâce au tri rapide.

Algorithme 1 permettant de calculer le chemin de type IV :

Entrée : un graphe $G' = (V', E')$, deux sommets $a, b \in V$
Sortie : un chemin P qui est le plus court du sommet a au sommet b

Début :
- Tranformation du graphe $G' = (V', E')$ en graphe $G'' = (V'', E'')$ avec $V'' = V'$ et $\forall((u, t_u), (v, t_v), t) \in E$, $\forall((u', t_{u'}), (v', t_{v'}), t') \in E$, si $u = u'$ et $t_u < t_{u'}$ alors $((u, t_u), (v', t_{v'}), t') \in E''$
- Recherche des sommets *depart* et *arrivee* avec $depart = (a, min\{t/(a, t) \in V'\})$ et $arrivee = (b, max\{t/(b, t) \in V'\})$.
- Retourne un chemin P en appliquant de l'algorithme BFS (c'est-à-dire un parcours en largeur).
Fin

Explication Dans le graphe G'' il existe forcement pour tous sommet un arc vers un autre sommet différent. La recherche en largeur d'abord est donc optimal car la valuation sur les arcs n'est plus prise en compte et on s'interesse désormais uniquement aux arcs entre sommet.

Algorithme 2 permettant de calculer le chemin de type IV :

Entrée : un graphe $G' = (V', E')$, deux sommets $a, b \in V$
Sortie : un chemin P qui est le plus court du sommet a au sommet b

Début :
- Recherche des sommets *depart* et *arrivee* avec $depart = (a, min\{t/(a, t) \in V'\})$ et $arrivee = (b, max\{t/(b, t) \in V'\})$.
- Retourne un chemin P en appliquant de l'algorithme de Dijkstra.
Fin

Explication Bien qu'on ne modifie pas G' l'application de cet algorithme retourne bien un chemin optimal car uniquement les arcs reliant deux sommet différent sont valués à 1. L'algorithme cherchant à minimiser le cout sur les arcs va retourner le chemin avec le moins de sommet possible.

Question 3

Soit $G = (V, E)$ un multigraphe avec $n = |V|$ et $m = |E|$

Nous appliquons une transformation du graphe G en graphe orientée $G' = (V', E')$ qui va servir pour la recherche de tous les chemins. Sa compléxité de la transformation est en $O(|V'| + |E'|)$.

Nous avons $|V'| \leq 2m$, puisque pour chaque arc de E nous créons au plus deux sommets dans $|V'|$.

Pour les arcs de E', le nombre d'arc de valeur 1 est $|E| = m$ et le nombre d'arc de valeur 0 est au plus $|V'| = m$, puisque les arcs de valeur de 0 se créent seulement entre chaque sommet ayant la même lettre et vont être retiré de la pile, par conséquent nous avons $\sum_{i \in V} deg(i) \leq 2m$ arcs crées.

Donc nous avons le graphe orientée $G' = (V', E')$ avec $|V'| \leq 2m$ et $|E'| \leq 3m$, par conséquent la complexité de la transformation est en $O(2m + 3m) = O(5m) = O(m)$

Complexité chemin type I Dijsktra:

Nous appliquons encore une deuxième transformation sur les arcs de G', d'où une complexité de $O(|E'|)$, puis la complexité des fonctions getVertexWithIdMax et getVertexWithIdMin (utilisé pour la recherche de sommet *départ* et *arrivée*) est en $O(|V'|)$. En appliquant l'algorithme de Dijkstra en $O(|E'| + |V'| * log(|V'|)$, donc la complexité est en $O(|V'| * log(|V'| + 2|V'| + 2|E'|) = O(|V'| * log(|V'| + |V'| + |E'|) = O(2m * log(2m)) = O(m * log(m))$.

Complexité chemin type II BFS:

Nous faisons une boucle pour trouver les sommets de *depart* et *arrivee* en $O(|V'|)$.

Ensuite, nous créons une liste de taille k des possibles chemins en $O(deg(depart) + deg(arrivee)) = O(k)$ dont nous le trions $O(k * log(k))$.

Pour finir, nous bouclons sur la liste et pour chaque couple $(depart, arrivee)$ possible nous appliquons l'algorithme de BFS en $O(|E'|)$ (car notre structure de graphe est un arbre enraciné) et même en $O(b^d)$ avec $b = \frac{|E'|}{|V'|}$ le facteur de branchement et d la profondeur du chemin entre *depart* et *arrivee*.

Puis la fonction getVertexWithIdMax est utilisé pour la recherche de sommet *arrivée* en $O(|V'|)$.

Nous avons une complexite $O(|V'| + k * log(k) + 2|E'| + k * \frac{|E'|^d}{|V'|^d})$.

Complexité chemin type II Dijsktra:

Nous appliquons encore une deuxième transformation sur les arcs de G', d'où une complexité de $O(|E'|)$, puis la complexité des fonctions getVertexWithIdMax et getVertexWithIdMin (utilisé pour la recherche de sommet *départ* et *arrivée*) est en $O(|V'|)$.

En appliquant l'algorithme de Dijkstra en $O(|E'| + |V'| * log(|V'|)$, donc la complexité est en $O(|V'| * log(|V'| + 2|V'| + 2|E'|) = O(|V'| * log(|V'| + |V'| + |E'|) = O(2m * log(2m)) = O(m * log(m))$.

Complexité chemin type III BFS:

Nous faisons une boucle pour trouver les sommets de *depart* et *arrivee* en $O(|V'|)$.

Ensuite, nous créons une liste de taille k des possibles chemins en $O(deg(depart)+deg(arrivee)) = O(k)$ dont nous le trions $O(k * log(k))$.

Pour finir, nous bouclons sur la liste et pour chaque couple $(depart, arrivee)$ possible nous appliquons l'algorithme de BFS en $O(|E'|)$ et même en $O(b^d)$ avec $b = \frac{|E'|}{|V'|}$ le facteur de branchement et d la profondeur du chemin entre *depart* et *arrivee*. Nous avons une complexite $O(|V'| + k * log(k) + 2|E'| + k * \frac{|E'|^d}{|V'|^d})$.

Complexité chemin type III Dijsktra:
Nous appliquons encore une deuxième transformation sur les arcs de G', d'où une complexité de $O(|E'|)$, puis la complexité des fonctions getVertexWithIdMax et getVertexWithIdMin (utilisé pour la recherche de sommet *départ* et *arrivée*) est en $O(|V'|)$.
En appliquant l'algorithme de Dijkstra en $O(|E'| + |V'| * log(|V'|)$, donc la complexité est en $O(|V'| * log(|V'| + 2|V'| + 2|E'|) = O(|V'| * log(|V'| + |V'| + |E'|) = O(2m * log(2m)) = O(m * log(m))$.

Complexité chemin type IV BFS:
Nous utilisons des fonctions getVertexWithIdMax et getVertexWithIdMin (utilisé pour la recherche de sommet *départ* et *arrivée*) de complexité en $O(|V'|)$.

Ensuite, nous appliquons l'algorithme de BFS en $O(|E'|)$ et même en $O(b^d)$ avec $b = \frac{|E'|}{|V'|}$ le facteur de branchement et d la profondeur du chemin entre *depart* et *arrivee*. Nous avons une complexite totale de $O(b^d + 2|V'|) = O(\frac{|E'|^d}{|V'|^d} + |V'|)$.

Complexité chemin type IV Dijsktra:
Nous appliquons encore une deuxième transformation sur les arcs de G', d'où une complexité de $O(|E'|)$, puis la complexité des fonctions getVertexWithIdMax et getVertexWithIdMin (utilisé pour la recherche de sommet *départ* et *arrivée*) est en $O(|V'|)$.
En appliquant l'algorithme de Dijkstra en $O(|E'| + |V'| * log(|V'|)$, donc la complexité est en $O(|V'| * log(|V'| + 2|V'| + 2|E'|) = O(|V'| * log(|V'| + |V'| + |E'|) = O(2m * log(2m)) = O(m * log(m))$.

Question 4

Nous avons choisi d'implémenter nos algorothmes de recherche en Python. La représentation des graphes se décompose en deux éléments : les sommets avec un dictionnaire et les arcs par un ensemble.

Le code pour des fonctions à implémenter en python se trouve dans les dossiers suivants :

Type 1 version Dijkstra : cheminType1.py
Type 2 version Dijkstra : cheminType2bis.py
Type 2 version BFS : cheminType2.py
Type 3 version Dijkstra : cheminType3bis.py

Type 3 version BFS : cheminType3.py
Type 4 version Dijkstra : cheminType4bis.py
Type 4 version BFS : cheminType4.py

Pour tester nos algorithmes avec une instance de graphe contenu dans un fichier, il faut se rendre dans le fichier main.py et modifier la ligne de code suivante :

```
13  G = utils.graphfromFile('test.txt')
14  V,E = G
15  Gt = utils.transformG_avecDico(G)
16  Vt, Et = Gt
```

Vous devez remplacer 'test.txt' par votre fichier de test qui doit se situer dans le même repertoire que le fichier main.py

Pour créer une instance via le terminal, vous devez alors décommenter les lignes de codes suivantes dans le fichier 'main.py'.

```
35  """
36  V,E = utils.saisieClavier() # A décommenter pour utilisation
37  Gt = utils.transformG_avecDico((V,E))
38  """
```

Cependant, à la moindre erreur une exception sera déclanchée et il faudra relancer l'execution du fichier.

Question 5

Nous allons résoudre le problème du plus court chemin de d à a du multigraphe G par programmation linéaire de la manière suivante :
- Transformation de $G = (V, E)$ en $G' = (V', E')$
- Définir la matrice d'incidence I de G'
- Les variables de décisions du PL sont les $x_{i,j} \in \{0, 1\} \forall i, j \in V'$ tel que $x_{i,j}$ vaut 1 si l'arc de i à j fait parti du chemin le plus court, vaut 0 sinon.
- Nous avons donc $|V'|^2$ variables de décisions.
- La fonction objectif doit alors minimiser le nombre d'arête séléctionnés, ce qui donne :

$$f(x) = \min \sum_i^{|V'|} \sum_j^{|V'|} I(i,j) x_{i,j}$$

Définitions des contraintes :
Contrainte 1 : Il doit y avoir une et une seule arête qui part du sommet de départ

$$\sum_j^{|V'|} x_{d,j} - x_{j,d} = 1$$

Contrainte 2 : Il doit y avoir une et une seule arête qui arrive sur sommet de d'arrivé.

$$\sum_j^{|V'|} x_{a,j} - x_{j,a} = -1$$

Contrainte 3 : Le nombre d'arc entrant doit être égale au nombre d'arête sortant

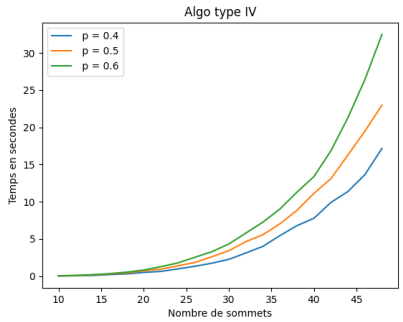
$$\forall\ i \in \{1, ..., |V'|\} \backslash \{\text{a,d}\}, \sum_j^{|V'|} x_{i,j} - x_{j,i} = 0$$

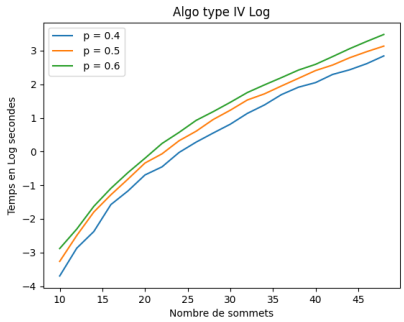
- Ce qui fait au total $|V'|$ contraintes
- Sans oublier les contraintes sur le domaine des variables $\forall i, j \in \{1, ..., |V'|\}$, $x_{i,j} \in \{0, 1\}$. Ce qui fait $|V'|^2$ contraintes en plus.

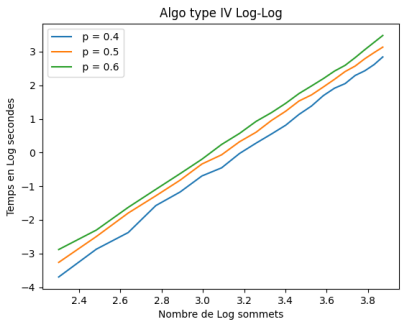
Donc nous avons au total $|V'| + |V'|^2$ contraintes

Question 6

Nous avons tracé le temps d'execution du programme linéaire. Le graphe à gauche est le temps d'execution en fonction du nombre de sommet, celle à droite est la même avec les valeurs logarithmes sur le temps d'execution et la dernière dont nous appliquons le logarithme sur le nombre de sommet et le temps d'execution.







D'après les graphes de test sur des sommet allant de 10 à 45 par pas de 2, nous observons que, quelque soit la probabilité p, le temps d'execution augmente quand le nombre de sommet augmente.

De la même manière, quelque soit le nombre de sommet, plus la probabilité p augmente plus le temps d'execution augmente. En effet, la courbe bleue (probabilité de $p = 0.4$) est toujours en dessous de la courbe verte (probabilité de $p = 0.6$).

Nous pouvons voir sur la figure en haut a gauche que le comportement de la courbe du temps est exponentielle en fonction du nombre de sommet du graphe.

Pour cela nous avons analyser les variations de la courbe sur une échelle logarithmique du temps et nous obtenons une courbe concave.

Pour obtenir une valeur approché de la compléxité de l'algorithme développé pour le programme linéraire du chemin de type 4, nous avons placé en échelle logarithmique le temps et le nombre de sommet. Nous obtenons la figure du bas. D'après nos calculs, nous obtenons les coefficients directeurs respectifs suivants 4.169 , 4.079 et 4.046 pour les droites correspondants aux trois probabilités $p = 0.4$,$p = 0.5$ et $p = 0.6$. Nous arrondissons ce coefficient directeur à la valeur 4.

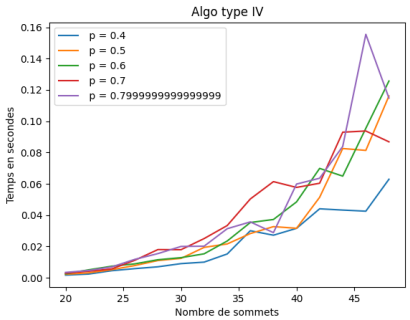
On exprime la compléxité expérimentale :

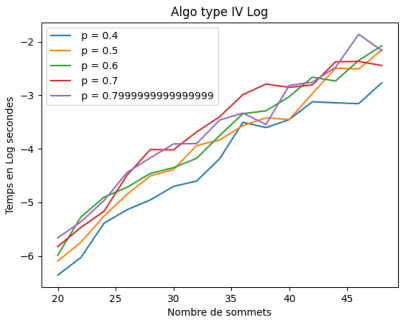
$$log(temps(n)) = 4 * log(n)$$
$$log(temps(n)) = log(n^4)$$
$$temps(n) = n^4$$

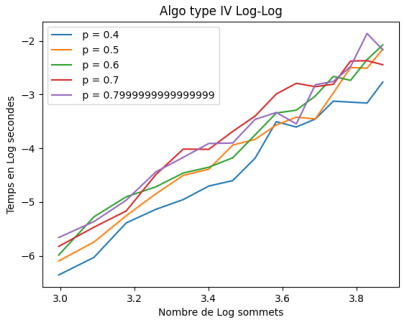
Notre algorithme est donc en $O(n^4)$, avec n la taille de l'instance.

Question 7

Voici des échantillons de tests pour les algorithmes de Type 4 avec la version Dijkstra.



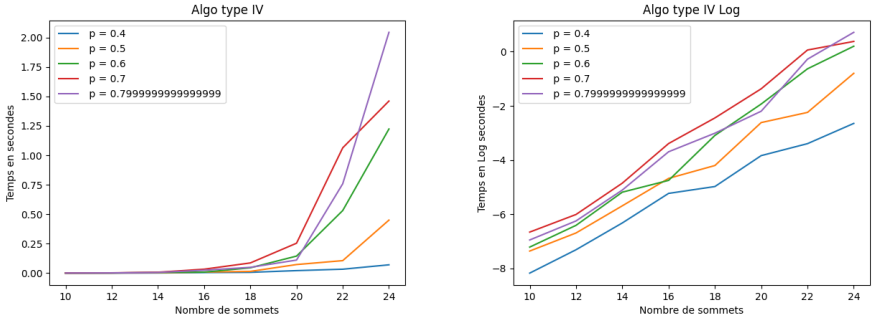


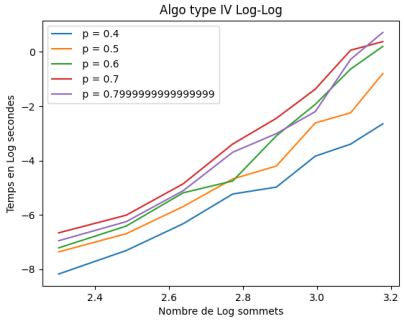


Nous observons que, quelque soit la probabilité p de présence d'arcs, plus le nombre de sommet augmente plus le temps d'execution augmente.

De la même manière, quelque soit le nombre de sommet, plus la probabilité p augmente plus le temps d'execution augmente. En effet, la courbe bleue (probabilité de $p = 0.4$) est toujours en dessous de la courbe rouge (probabilité de $p = 0.7$).

Voici des échantillons de tests pour les algorithmes de Type 4 avec la version BFS.





Nous remarquons que, quelque soit la probabilité p de présence d'arcs, plus le nombre de sommet augmente plus le temps d'execution augmente.

De la même manière, quelque soit le nombre de sommet, plus la probabilité p augmente plus le temps d'execution augmente. En effet, la courbe bleue (probabilité de $p = 0.4$) est toujours en dessous de la courbe rouge (probabilité de $p = 0.7$).

Mais en comparant les deux algorithmes, nous constatons que le temps d'exécution avec le BFS est largement plus supérieur qu'avec Dijkstra: pour $p = 0.7999$ et $n = 24$ sommets, l'algorithme avec Dijkstra prend moins d'une seconde alors que l'algorithme avec BFS prend 2 secondes.

Bonus

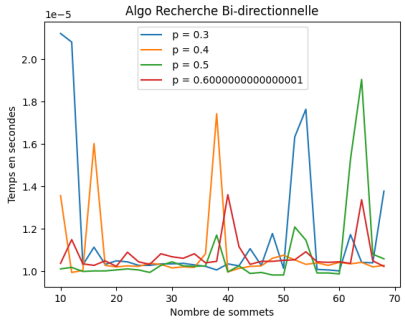
Nous proposons ici une implémentation d'un algorithme inspiré du livre "Artificial Intelligence : A Modern Approach" de Peter Norvig et Stuart Russel. C'est un algorithme de recherche du plus court chemin (Type IV) qui utilise la recherche bi-directionnelle en appliquant deux algorithmes de recherche en largeur (BFS).

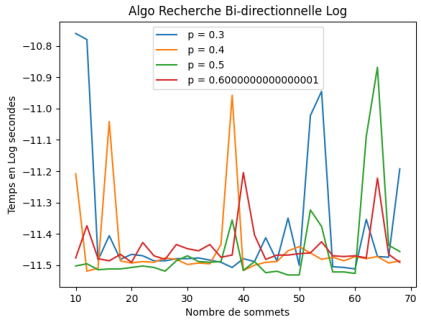
Entrée : un graphe $G = (V, E)$, deux sommets $a, b \in V$
Pseudo - code :
- Déclaration de deux algorithmes BFS notés B_1(G,a,b),B_2(G,a,b)
- Tant que non connexion(B_1,B_2) faire :
- (1) Evaluation le premier noeud n_1 de la frontiere de B_1
- si connexion(B_1,B_2) : aller en (2)
- sinon on évalue le premier noeud n_2 de la frontière de B_2
- si connexion(B_1,B_2) : aller en (2)
- sinon $frontiere(B_1) = frontiere(B_1)/\{n_1\}$ et $frontiere(B_2) = frontiere(B_2)/\{n_2\}$
- revenir en (1)
- (2) retourner $buildSolution(B_1, B_2)$

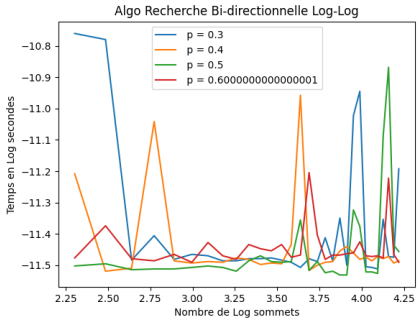
La fonction connexion doit renvoyer Vrai s'il existe un noeud n qui est dans l'intersection de $frontiere(B_1)$ et $frontiere(B_2)$ et faux sinon.

L'implémentation de cet algorithme se situe dans le fichier cheminType4BiDirection.py. Cette implémentation prend bien évidement les contraintes de temps de notre contexte.

Compléxité temporelle et spatiale en $O(p^{d/2})$ ou p représente le facteur de branchement du graphe et d la profondeur maximale de l'arbre de recherche. Cette approche bi-directionnelle est donc exponentiellement plus rapide que nos simples algorithmes de BFS adoptés aux questions précédentes.

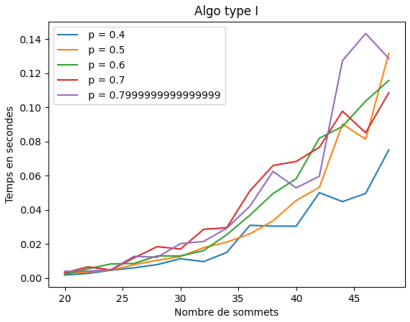


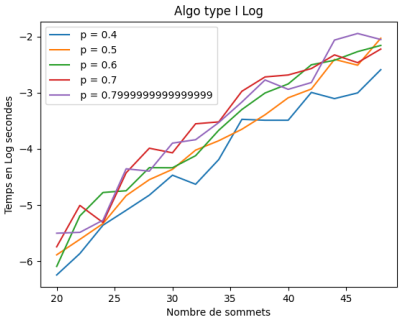


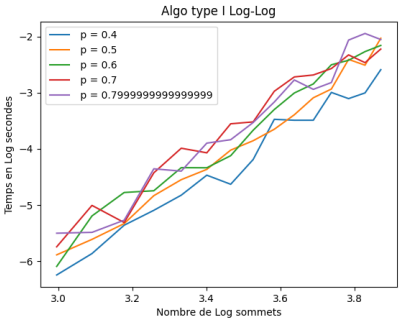


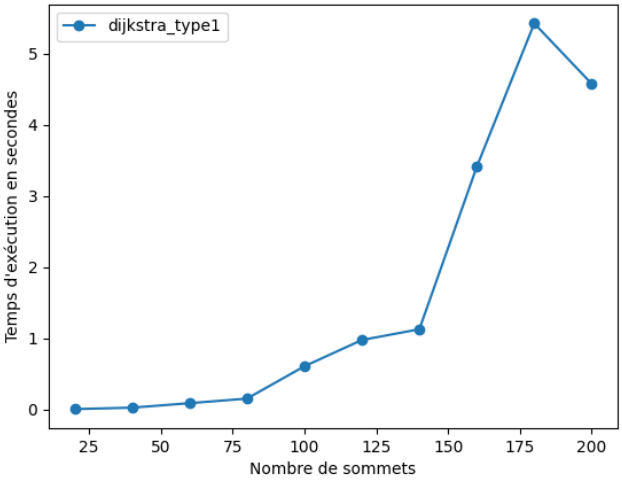
Test sur les autres chemins

1 Chemin Type I



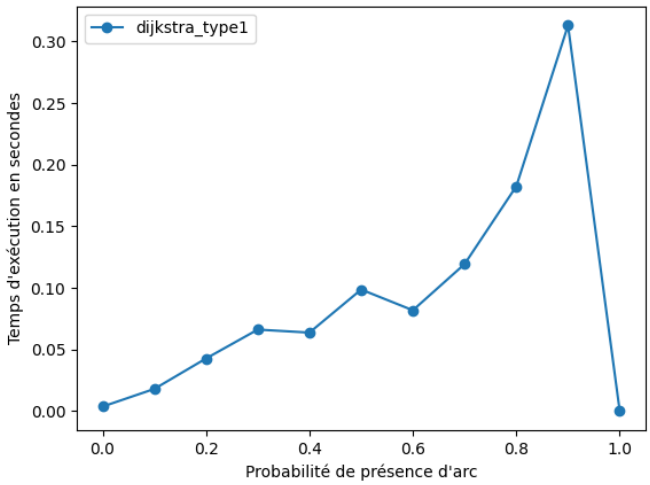






Nous calculons le temps d'exécution des algorithmes avec une probabilité $p = 0.5$ de présence d'arcs entre deux sommets en fonction du nombre de sommet avec 20 instances de graphes.

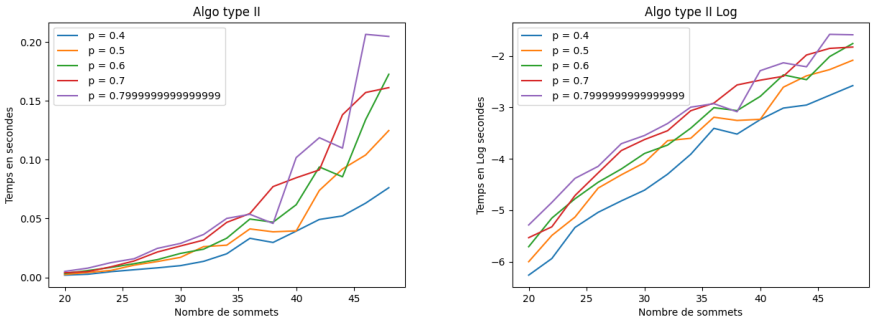
D'après la courbe du chemin d'arrivée au plus tôt, nous observons que l'algorithme de Dijkstra prend moins d'une seconde pour graphe de 100 sommets. Mais à partir de 140 sommets, il commence à prendre plus d'une seconde. Dans notre cas, il prend au plus de 5.42 secondes pour 180 sommets.

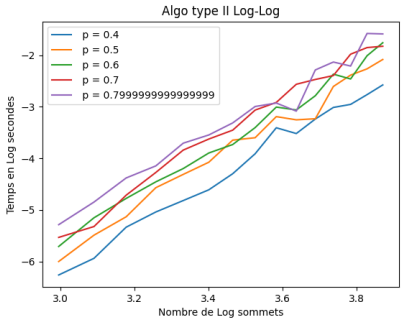


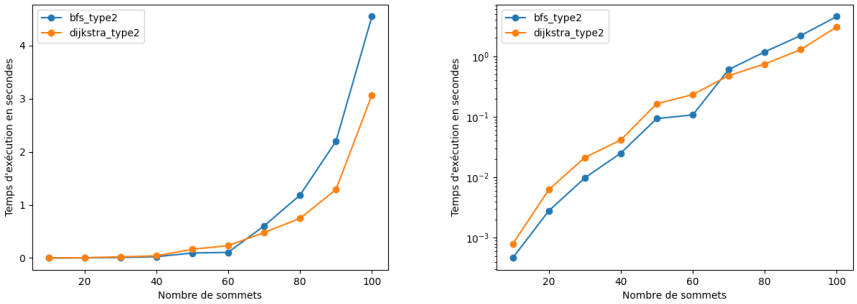
Nous calculons le temps d'exécution des algorithmes d'un graphe de 60 sommets avec 20 instances en fonction d'une probabilité p de présence d'arcs entre deux sommets .

Nous observons que le temps d'execution augmente quand la probabilité augmente mais pour $p = 1$, l'algorithme retourne un chemin immédiatement puisque quand $p = 1$, alors il existe un arc entre le sommet *depart* et le sommet *arrivee* dans le graphe.

2 Chemin Type II



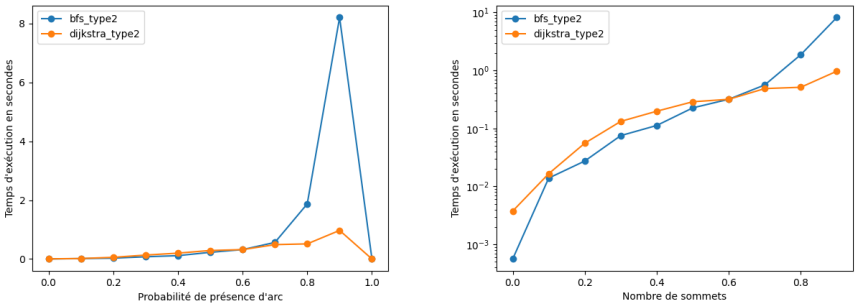




Nous calculons le temps d'exécution des algorithmes avec une probabilité $p = 0.5$ de présence d'arcs entre deux sommets en fonction du nombre de sommet avec 20 instances de graphes.

Pour moins de 40 sommets, nous ne différencions pas le temps entre les deux algorithmes. Avec le deuxieme graphe, qui est le même avec les valeurs mise sous logarithme, nous observons bien que l'algorithme de BFS est légèrement rapide que Dijkstra.

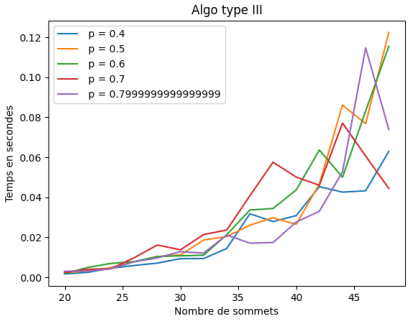
Dans les deux graphes, nous observons bien qu'à partir de 70 sommets, l'algorithme de Dijkstra est plus rapide que l'algorithme de BFS.

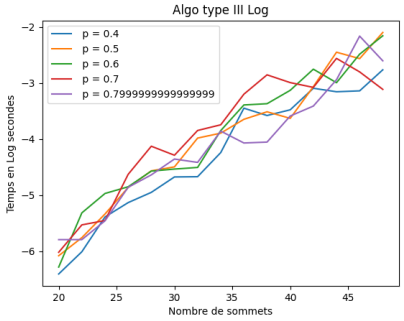


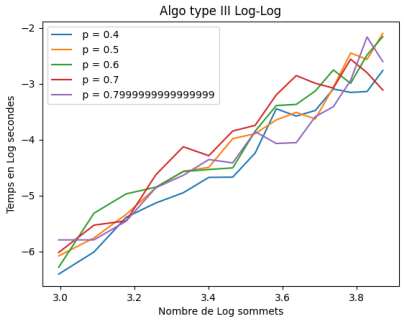
Nous calculons le temps d'exécution des algorithmes d'un graphe de 60 sommets avec 20 instances en fonction d'une probabilité p de présence d'arcs entre deux sommets .

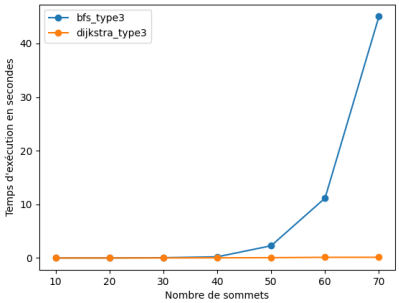
Le deuxième graphe est le même que le premier avec le logarithme donc nous ne prenons pas en compte le temps d'execution vallant 0, c'est-à-dire nous excluons la probabilité $p = 1$.

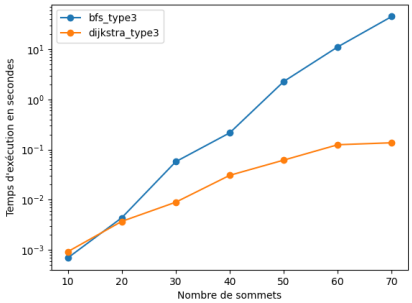
3 Chemin Type III







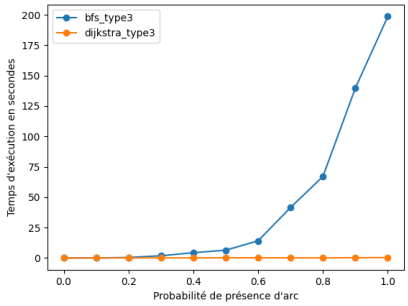




Nous calculons le temps d'exécution des algorithmes avec une probabilité $p = 0.5$ de présence d'arcs entre deux sommets en fonction du nombre de sommet avec 20 instances de graphes.

En observant le graphe des valeurs logarithmiques, nous remarquons bien que l'algorithme de BFS est moins efficace que l'algorithme de Dijkstra, notamment

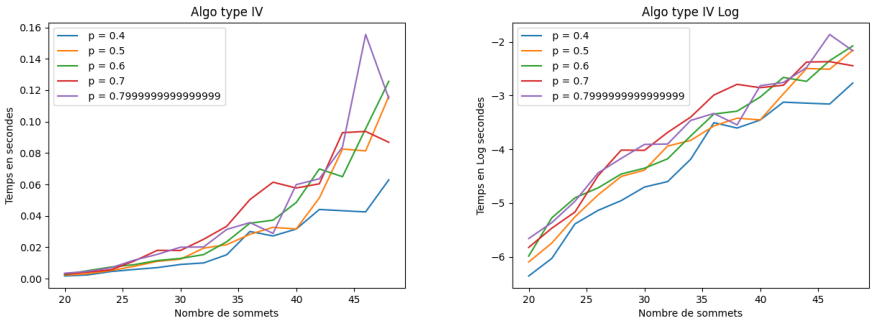
pour un nombre de sommet supérieur à 40.

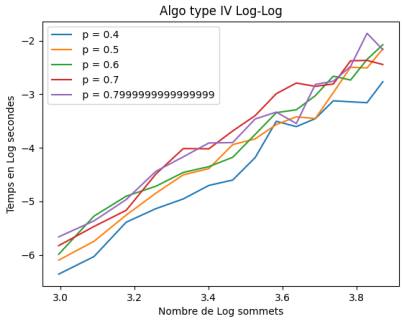


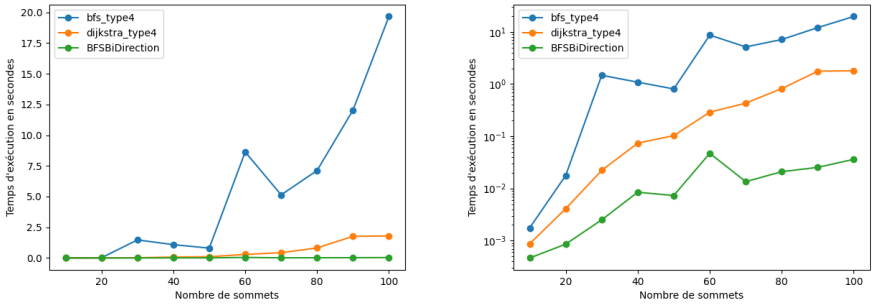
Nous calculons le temps d'exécution des algorithmes d'un graphe de 60 sommets avec 20 instances en fonction d'une probabilité p de présence d'arcs entre deux sommets.

Nous remarquons que l'algorithme de BFS est largement plus lent que l'algorithme de Dijsktra. En effet, pour $p = 1$, le temps d'execution prends plus de 200 secondes pour le BFS alors que pour Dijsktra le temps est inférieur à 25 secondes.

4 Chemin Type IV

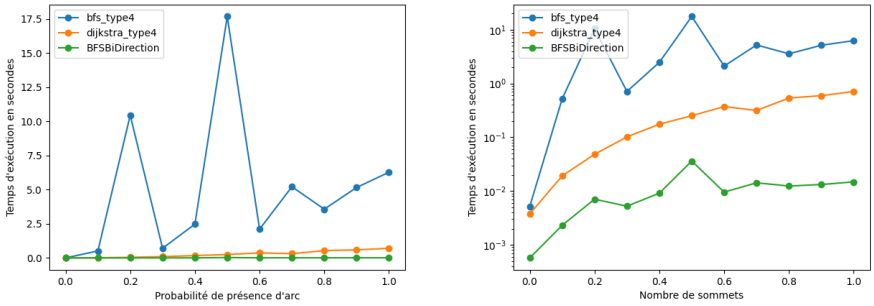






Nous calculons le temps d'exécution des algorithmes avec une probabilité $p = 0.5$ de présence d'arcs entre deux sommets en fonction du nombre de sommet avec 20 instances de graphes.

Nous remarquons l'algorithme de BFS est moins efficace que les algorithmes de Dijkstra et de Bidirectionelle, notemment quand nous avons 100 sommets. Avec le deuxième graphe, celle avec le logarithme, nous observons que l'algorithme Bidirectionelle est légèrement rapide que l'algorithme de Dijkstra.



Nous calculons le temps d'exécution des algorithmes d'un graphe de 60 sommets avec 20 instances en fonction d'une probabilité p de présence d'arcs entre deux sommets.

D'après le graphe, l'algorithme de BFS est moins efficace que les algorithmes de Dijkstra et de Bidirectionelle. Avec le graphe avec les valeurs en logarithme, nous observons que l'algorithme Bidirectionelle est légèrement rapide que l'algorithme de Dijkstra.

Conclusion

Dans un multigraphe orienté, trouver des chemins d'arrivée au plus tôt, de départ au plus tard, de durée minimale ou du plus court chemins peut être de temps exponentiel à cause de l'énumération de toutes les chemins possibles.

Mais grâce à une transformation du graphe, les contraintes de temps sont facilement vérifiées. Puis l'application de l'algorithme de Dijsktra permet de trouver ces chemins, en temps polynomial.