ABITBOL Yossef , 3804139
ZARROUK Yaël, 21107919

SORBONNE UNIVERSITÉ

RAPPORT PROJET MOGPL

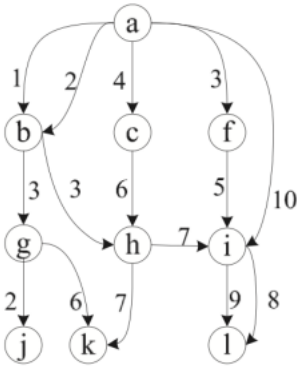
Langage choisi: **python \ Jupyter notebook.**

I. INTRODUCTION :

Nous allons considérer des **multi graphes orientés pondérés** par le temps. Soit G un tel multigraphe. On va supposer que G est **sans circuit** et nous allons définir **4 types de "plus courts" chemins** qui ont chacun sa propre importance. Ensuite, nous transformerons ce multigraphe en un graphe classique qui va nous permettre de concevoir des algorithmes afin de calculer les 4 types de chemins minimaux. Enfin, nous allons implémenter et tester un algorithme du type 4 par programmation linéaire en faisant appel à GUROBI.

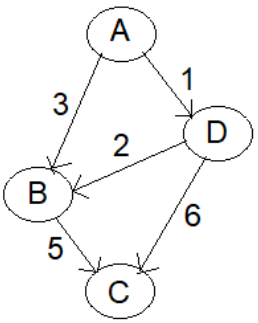
II. PREUVES DES INSTANCES:

En utilisant l'instance suivante montrer que les assertions suivantes sont vraies :



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

Pour montrer que cette assertion est vraie on va prendre le multi graphe suivant :



Le chemin d'arrivée au plus tôt à **C** en partant de A est :

P(A,C,[3,6]) = ((a,b,3,1),(b,c,5,1)).

Or le sous chemin préfixe de ce chemin qui est P1=(a,b,3,1) n'est pas le chemin d'arrivée au plus tôt car le chemin:

P1'(A,B,[1,3]) = ((a,d,1,1),(d,b,2,1)) est meilleur.

(car fin(P1') = 3 < fin(P1)= 4)

Cela montre qu' un sous chemin préfixe d'un chemin d'arrivée au plus tôt peut ne pas être un chemin d'arrivée au plus tôt.

ABITBOL Yossef , 3804139
ZARROUK Yaël, 21107919

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

Si on prend le multigraphe de base, le chemin avec le départ le plus tard de **A** à **L** est :

P(A,L,[4,9])=((a,c,4,1),(c,h,6,1),(h,i,7,1),(i,L,8,1))

de début(P) = 4

Or, si on prend le sous chemin de i à I (du chemin de départ au plus tard de A à L) qui est P(I,L,[8,9])=(*(i,L,8,1)*) de début(P) = 8 n'est pas le chemin de départ au plus tard car **P'(i,I,[9,10]) = (***(i,L,9,1)***)** de début(P') = 9 est meilleur.
Donc, un sous-chemin d'un chemin le plus rapide peut ne pas être le chemin le plus rapide.

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

Si on reprend le multigraphe de base, le chemin le plus rapide de **A** à **L** est :

P(A,L,[4,9])=((a,c,4,1),(c,h,6,1),(h,i,7,1),(i,L,8,1))

de durée(P) = 9 - 4 = **5**

Or, si on prend le sous chemin de A à I (du chemin le plus rapide de A à L) qui est P(A,I,[4,8])=((a,c,4,1),(c,h,6,1),(h,i,7,1)) de durée(P) = 8 - 4 = **4** n'est pas le chemin le plus rapide car **P'(A,I,[10,11]) = ((a,i,10,1))** de durée(P') = 11 - 10 = **1** est plus rapide.
Donc, un sous-chemin d'un chemin le plus rapide peut ne pas être le chemin le plus rapide.

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

Toujours sur le même graphe, le chemin le plus court de A à L est :

P(A,L,[3,9]) = ((a,f,3,1),(f,i,5,1),(i,L,8,1)) d'une **dist(P) = 3**

Or, le sous chemin de A à I P1(A,I,[3,6])=((a,f,3,1),(f,i,5,1)) d'une **dist(P1) = 2** n'est pas le plus court car le chemin **P1'(A,I,[10,11])=((a,i,10,1))** d'une **dist(P1') = 1** est plus court.
Donc, un sous-chemin d'un plus court chemin peut ne pas être un plus court chemin.

III. CALCUL DES 4 TYPES:

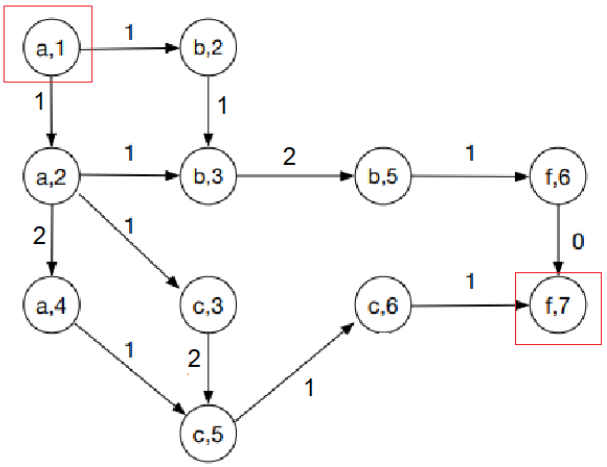
a) Pour calculer le chemin **d'arrivée au plus tôt (TYPE I)** :

Entrée: Graphe transformé normal **G**, **départ** (sommet de départ), **arrivee** (sommet final) , t_alpha, t_omega
Début: - On **change le poid** des arêtes ((u,tu), t, (v,tv)) par la valeur de $T_v - T_u$ sauf pour les arcs ((arrivee,tu), t, (arrivee,tv)) qui **garde la valeur de 0.**
- On applique l'algorithme de Dijkstra sur le min des sommets de départs et le max des sommets d'arrivée en fonction de t.
Sortie: On renvoie le chemin correspondant.

Pourquoi cette méthode fonctionne-t-elle ?

Pour montrer l'efficacité de cette méthode, appliquons la sur un exemple.
Prenons le graphe transformé normal suivant :

ABITBOL Yossef , 3804139
ZARROUK Yaël, 21107919



Un chemin de A a F ?

Tout d'abord, on peut voir le choix important de partir du min de départ (ici (a,1)) au max d'arriver (ici (f,7)), cela va nous permettre de parcourir tous les chemins possible du début à la fin car si par exemple on aurait pris (a,2) on aurait jamais pu tester avec (a,1).
Ensuite, le choix de mettre des 0 aux sommets d'arrivées, cela revient a trouver le chemin le plus court **au premier f accessible**, dans notre cas le plus petit donc l'arrivée au plus tôt.
Donc, en partant de (a,1) on trouve le chemin le plus court et ainsi le plus tôt qui est le suivant :

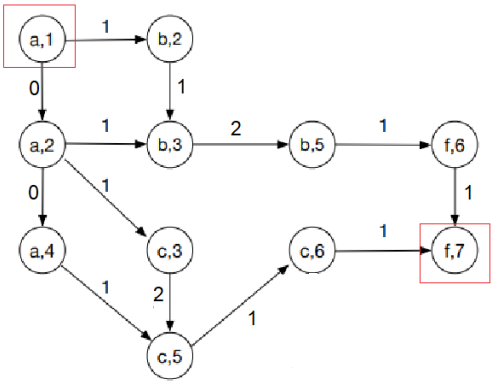
(a,1) -> (b,2) -> (b,3) -> (b,5) -> (f,6)

b) Pour calculer le chemin de **départ au plus tard (TYPE II)**:

Entrée: Graphe transformé normal **G**, **départ** (sommet de départ), **arrivee** (sommet final) , t_alpha, t_omega
Début: - On **change le poid** des arêtes ((départ ,tu), t, (depart ,tv)) par la valeur de 0 et les autres arcs restants ((u,tu), t, (v,tv)) par $T_v - T_u$.
- On applique l'algorithme de Dijkstra sur le min des sommets de départs et le max des sommets d'arrivée en fonction de t.
Sortie: On renvoie le chemin correspondant.

Pourquoi cette méthode fonctionne-t-elle ?

Pour montrer l'efficacité de cette méthode, appliquons la sur un exemple.
Prenons le graphe transformé normal suivant :



Un chemin de A a F ?

Ici, le choix de mettre les 0 aux sommets de départs va nous permettre de nous déplacer de (a,1) à (a,2) à (a,4) et ainsi **commencer avec le sommet de départ au plus tard** qui est (a,4) et de ce sommet on va essayer de trouver le chemin le plus court grâce à l'algorithme de Dijkstra.

c) Pour calculer le chemin **le plus rapide (TYPE III)**:

Entrée: Graphe transformé **G**, **u** (sommet de départ),**v** (sommet final)

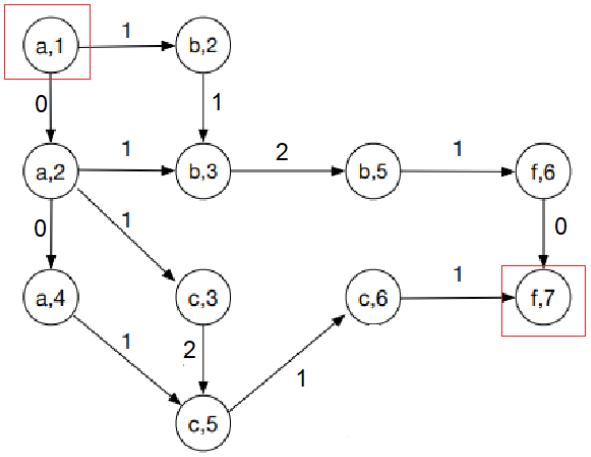
Début: - On **change le poid** des arêtes ((départ ,tu), t, (depart ,tv)) et ((arrivee,tu), t, (arrivee,tv)) par la valeur de 0 et les autres arcs restants ((u,tu), t, (v,tv)) par $T_v - T_u$.

- On applique l'algorithme de Dijkstra sur le min des sommets de départs et le max des sommets d'arrivée en fonction de t.

Sortie: On renvoie le chemin correspondant.

Pourquoi cette méthode fonctionne-t-elle ?

Pour montrer l'efficacité de cette méthode, appliquons la sur un exemple.
Prenons le graphe transformé normal suivant :



Un chemin de A a F ?

Dans ce cas, on cherche le chemin le plus rapide et cela signifie le min { fin - début }. Le but recherché dans notre transformation, est qu'en partant de n'importe quel sommet de départ (prenons (a,2)) nous arrivons au premier sommet d'arrivée accessible (ici (f,6)) avec pour valeur (6-2 = 4), ((a,1) à (f,6) avec une valeur de 5 qui est correct car (((a,1),**1**,(b,2)), ((b,2),**1**,(b,3)), ((b,3),**2**,(b,5)), ((b,5),**1**,(f,6)), ((f,6),**0**,(f,7)) avec une valeur de 1+1+2+1+0 = 5) et enfin de (a,4) à (f,7) d'une valeur de 3 qui est le min et donc le chemin le plus court, cela nous renvoi bien le chemin le plus rapide.

d) Pour calculer le **plus court chemin (TYPE IV)**:

Entrée: Graphe transformé **G**, **u** (sommet de départ),**v** (sommet final)

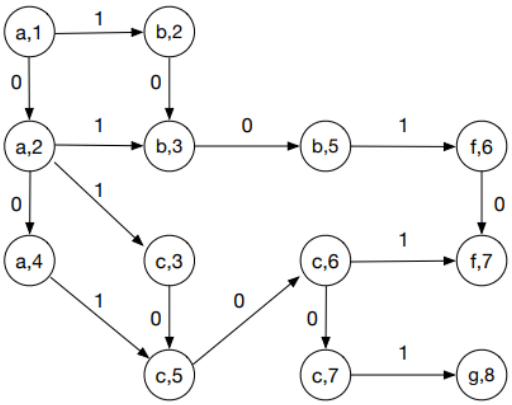
Début: - On applique l'algorithme de Dijkstra sur le min des sommets de départs et le max des sommets d'arrivée en fonction de t.

Sortie: On renvoie le chemin correspondant.

Pourquoi cette méthode fonctionne-t-elle ?

Pour montrer l'efficacité de cette méthode, appliquons la sur un exemple.
Prenons le graphe transformé normal suivant :

ABITBOL Yossef , 3804139
ZARROUK Yaël, 21107919



Un chemin de A a F ?

On ne fait aucune transformation de plus sur le graphe transformé normal car on va prendre en compte que les coûts des arêtes (la distance qui correspond a lambda) entre les sommets (u,t) qui ont un u différent, c'est à dire que les arcs **((u,t1),lambda,(v,t2)) avec u != v.** Il nous suffit ensuite d'appliquer dijkstra et on obtient ainsi le plus court chemin en termes de distance lambda.

III. Complexité:

Ici, on va calculer la complexité de chaque algorithme. Avant tout, on a un multi graphe G = (V,E) avec |V| = s le nombre de sommets et |E| = a le nombre d'arcs qu'on va **transformer en graphe classique** G' = (V',E') avec |V'| = s' le nombre de sommets et |E'|= a' le nombre d'arcs.
De cette transformation, on va l'adapter pour satisfaire nos besoins comme vu plus haut en II.
Dans les 3 premiers types de calculs des chemins minimaux:

- On fait une **modification de la valeur des arêtes**, c'est à dire qu'on parcourt toutes les arêtes en **O(|E'|),**
- On cherche avant d'appliquer Dijkstra le **min {sommet de départ},** qui consiste tout simplement à parcourir les différents sommets et à prendre le minimum donc en **O(|V'|)**
- De même pour le **max {sommet de fin},** qui consiste tout simplement à parcourir les différents sommets et à prendre le maximum donc en **O(|V'|)**
- Puis on applique l'algorithme de Dijkstra qui a une complexité connue en **O(|E'| + |V'|log(|V'|)).**

  Cela nous donne donc une complexité commune aux 3 algorithmes :
  O(|E'| + |V'| + |V'| + |E'| + |V'|log(|V'|)) = O(2|E'| + 2|V'| + |V'|log(|V'|))

Pour le dernier algorithme, on ne modifie pas les arêtes donc : **O(|E'| + 2|V'| + |V'|log(|V'|))**

IV. Implémentation en python:

Le langage choisi ici, est le langage **python** et nous avons utilisé Jupyter. De plus, chaque fonction est précédée d'un titre, contient des commentaires pour une compréhension plus simple et rapide des fonctions et est suivie de tests.
Tout d'abord nous avons pris comme exemple de fichier texte, un fichier d'une de ces formes :

ABITBOL Yossef , 3804139
ZARROUK Yaël, 21107919

```
2
2
a
b
(a,b,3,1)
(a,b,2,1)
```

```
5 % nombre de sommets
7 % nombre d'arcs
sommet a
sommet b
sommet c
sommet f
sommet g
arc (a,b,1,1) % sous la forme (u,v,t,lambda)
arc (a,b,2,1)
arc (a,c,2,1)
arc (a,c,4,1)
arc (b,f,5,1)
arc (c,f,6,1)
arc (c,g,7,1)
```

Pour traduire le fichier en graphe de gauche nous avons créé la fonction : **ouvrir_fichier(nom_fichier)**
et celui de droite : **ouvrir_fichier2(nom_fichier)**

qui va nous renvoyer un graphe sous la forme d'un dictionnaire pour les sommets avec pour clés les sommets 'a' et pour valeur un index commençant à 0.
Tout le reste à savoir sur les fonctions est sur le fichier jupyter joint à ce pdf.

V. Programmation linéaire:

Maintenant, nous allons proposer une modélisation du problème de plus court chemin du type 4 par programmation linéaire et nous allons l'implanter en faisant appel à **GUROBI**.
Soit G = (V,E) un graphe avec comme sommet de départ u et sommet d'arrivée v.
Premièrement, on va transformer ce graphe en un graphe G' = (V',E') classique pour ensuite créer la **matrice d'incidence A** tel que chaque case de la matrice correspond à une liaison :

- A[i,j] **= 1** → on a un arc de i a j,
- A[i,j] **= -1** → on a un arc de j a i,
- A[i,j] **= 0** → on n'a pas d'arc entre i et j.

Une fois cette matrice créée on peut définir notre fonction objectif qui va correspondre à la minimisation du nombre d'arcs choisis pour trouver le plus petit chemin en fonctions des **variables de décisions** $x_{i,j} \in \{0, 1\}$ tel que $x_{i,j} = 1$ si l'arc est choisi :

$$Min \sum_{i,j \in E'} (A[i,j] * x_{i,j})$$

Une fois la fonction objectif défini, trouvons maintenant les différentes contraintes.
Un chemin le plus court correspond a un chemin qui commence à un sommet de **départ u**, passe par des sommets **intermédiaires i** et finit par le sommet d'**arrivée v:**

u → i1 → i2 → i3 → …→ ik → **v**

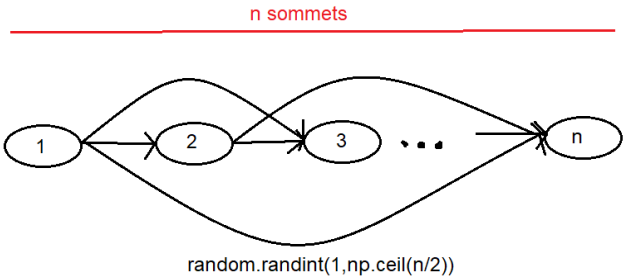
- $\sum_{j \in E'} x_{u,j} - \sum_{j \in E'} x_{j,u} = 1$
- $\sum_{j \in E'} x_{v,j} - \sum_{j \in E'} x_{v,u} = -1$
- $\sum_{j \in E'} x_{i,j} - \sum_{j \in E'} x_{i,u} = 0$
- $x_{i,j} \geq 0$

ABITBOL Yossef , 3804139
ZARROUK Yaël, 21107919

SORBONNE UNIVERSITÉ

VI. Mesure du temps d'exécution:

Pour pouvoir faire des tests et obtenir des résultats assez parlant nous devons faire varier le nombre de sommets, le nombre d'arcs et les étiquettes sur les sommets. Pour cela, nous allons générer des graphes aléatoirement, qui bien sûr seront sans circuit. Nous allons procéder de la manière suivante :



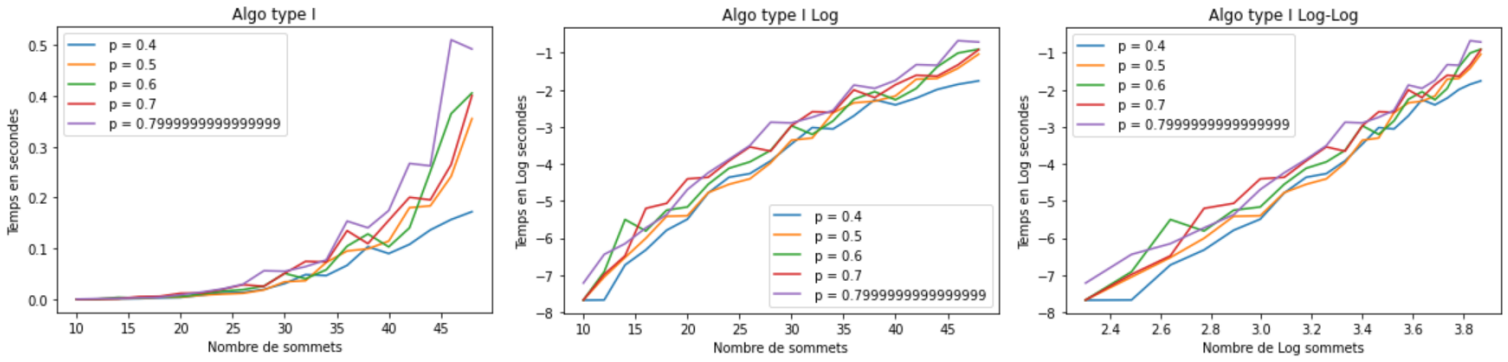
-On crée n sommets, et pour chaque sommet on crée un arc vers les sommets suivants avec une valeur choisie aléatoirement entre 1 et la partie entière de n/2, cela va nous permettre d'obtenir un graphe sans circuit.
- On pourra facilement faire varier le nombre de sommets avec n, chaque arc aura une probabilité P d'apparition, il nous suffira de faire varier P.
Tous les tests ont été effectués sur les mêmes graphes générés aléatoirement entre **10 à 50 sommets** par pas de 2 et une probabilité d'apparition des arcs qui varie de **0,4 à 0,8** par pas de 0,1. Chaque cas a été **exécuté 15 fois** pour pouvoir faire la moyenne et ainsi obtenir des résultats cohérents.
Chaque test a été effectué avec un ordinateur dont les paramètres sont les suivants :

CPU :
Intel(R) Core(TM) i5-8264U CPU @ 1.60Ghz
Coeur(s) : 4
Thread(s) : 8

Fréquences :
Vitesse Coeur : 600 Mhz Mult. : x6.0(4-39)
Vitesse de bus 99.94 Mhz

Cache :
Données L1 : 4 * 32 Ko, 8 voies, 64 octets par ligne @ 10^5 MB/s
Instr. L1 : 4 * 32 Ko, 8 voies, 64 octets par ligne @ $6.5 * 10^4$ MB/s
Niveau L2 : 4 * 256 Ko, 4 voies, 64 octets par ligne @ $5 * 10^4$ MB/s
Niveau l3 : 6 Mo, 12 voies

Pour le type I :

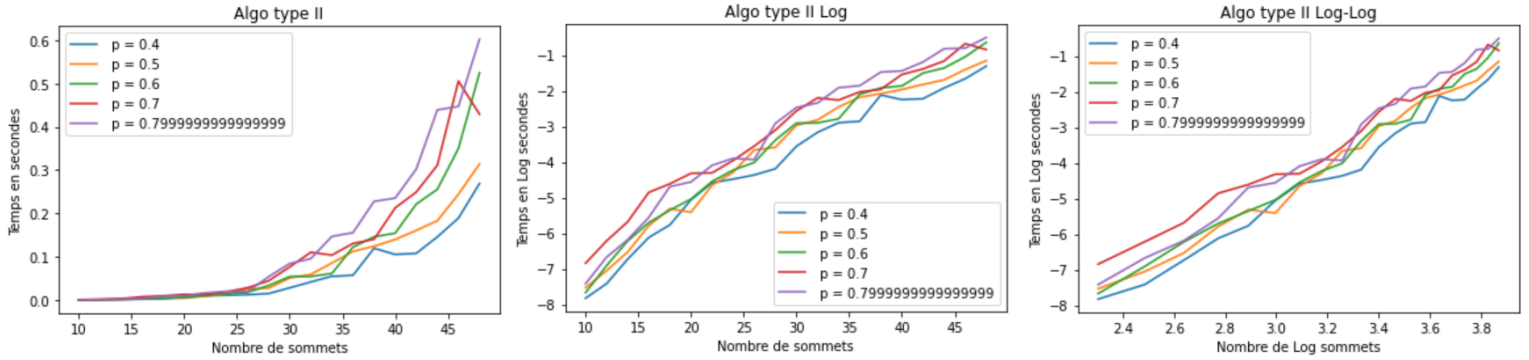


Sur la première figure de gauche on peut voir qu'entre 10 et 30 sommets le temps de réponse est très proche de 0 mais à partir de 35 le temps commence à augmenter et atteint 0.5 pour 45 sommets et p=0.8.

ABITBOL Yossef , 3804139
ZARROUK Yaël, 21107919

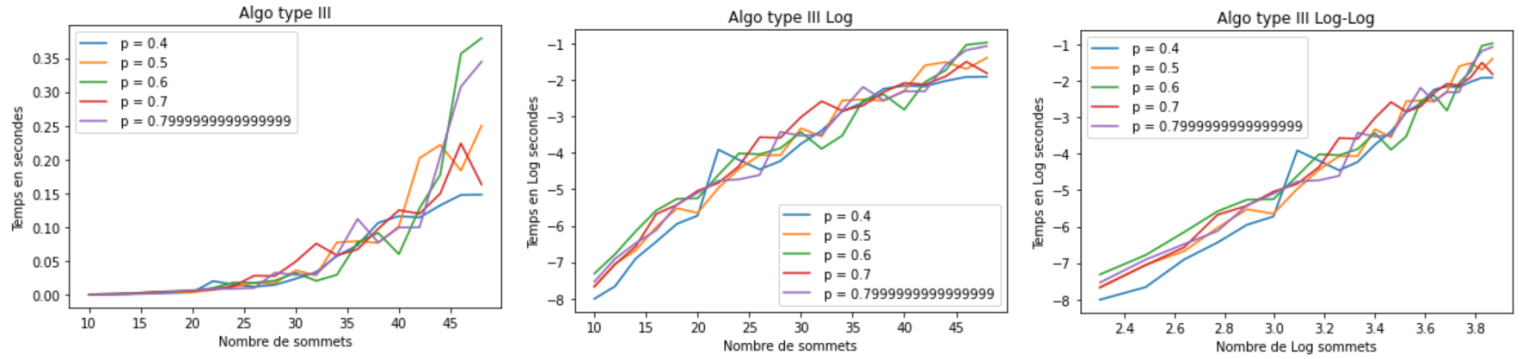
Plus le nombre de sommets augmente, plus le temps augmente. On remarque aussi que la droite bleu (p=0.4) est toujours inférieur à la droite mauve (p=0.8), cela montre que **plus on augmente p plus le temps est long,** cela peut être expliqué par le fait que plus p et n est grand plus l'algorithme doit tester et donc c'est plus long.
En termes de complexité, on a besoin de voir les 2 autres graphes qui montrent que l'algorithme a une **complexité polynomiale** du par la fonction affine du graphe log/log de **coeff directeur égale à 4.3**.

Pour le type II :



Sur la première figure de gauche on peut voir qu'entre 10 et 25 sommets le temps de réponse est très proche de 0 mais à partir de 35 le temps commence à augmenter et atteint 0.6 pour 50 sommets et p=0.8.
Plus le nombre de sommets augmente, plus le temps augmente. On remarque aussi que la droite bleu (p=0.4) est toujours inférieur à la droite mauve (p=0.8), cela montre que **plus on augmente p plus le temps est long,** cela peut être expliqué par le fait que plus p et n est grand plus l'algorithme doit tester et donc c'est plus long.
En termes de complexité, on a besoin de voir les 2 autres graphes qui montrent que l'algorithme a une **complexité polynomiale** du par la fonction affine du graphe log/log de **coefficient directeur égale 4.2**.

Pour le type III :



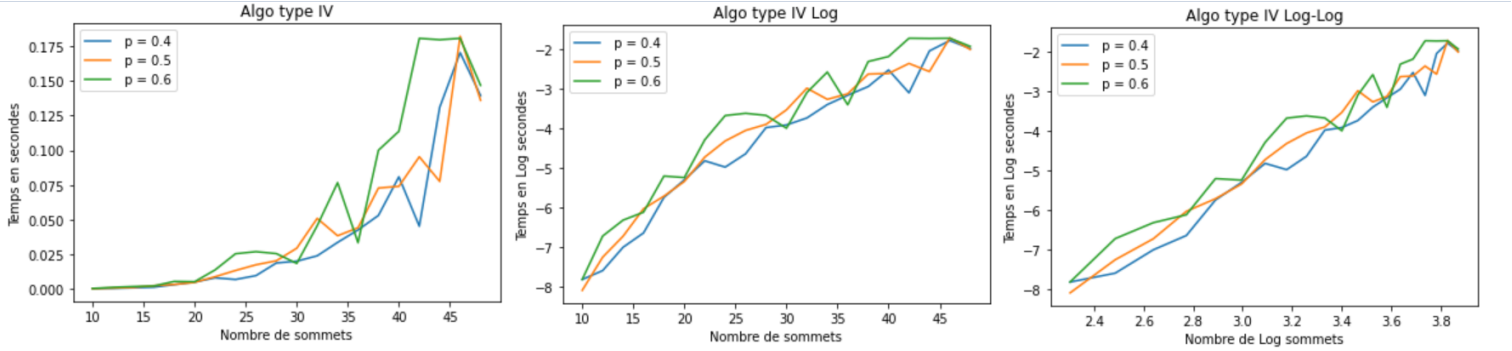
Sur la première figure de gauche on peut voir qu'entre 10 et 20 sommets le temps de réponse est très proche de 0 mais à partir de 30 le temps commence à augmenter et atteint 0.35 pour 45 sommets et p=0.6.
Plus le nombre de sommets augmente, plus le temps augmente. On remarque ici que p = 4 est des fois supérieur a p=0.8, cela signifie que le nombre d'arcs n'influence pas énormément le temps de réponse.
En termes de complexité, on a besoin de voir les 2 autres graphes qui montrent que l'algorithme a une **complexité polynomiale** du par la fonction affine du graphe log/log de **coefficient directeur égale à 3.9.**

Pour conclure sur ces 3 premiers types, le calcul du **type 3 est le plus rapide** a calculer, **(n=40,t=0.10)** suivi du type 1 mais très proche du type 2, ca dépend de p. On a aussi montré que la complexité théorique (complexité **polynomiale)** est en accord avec l'expérimentation.

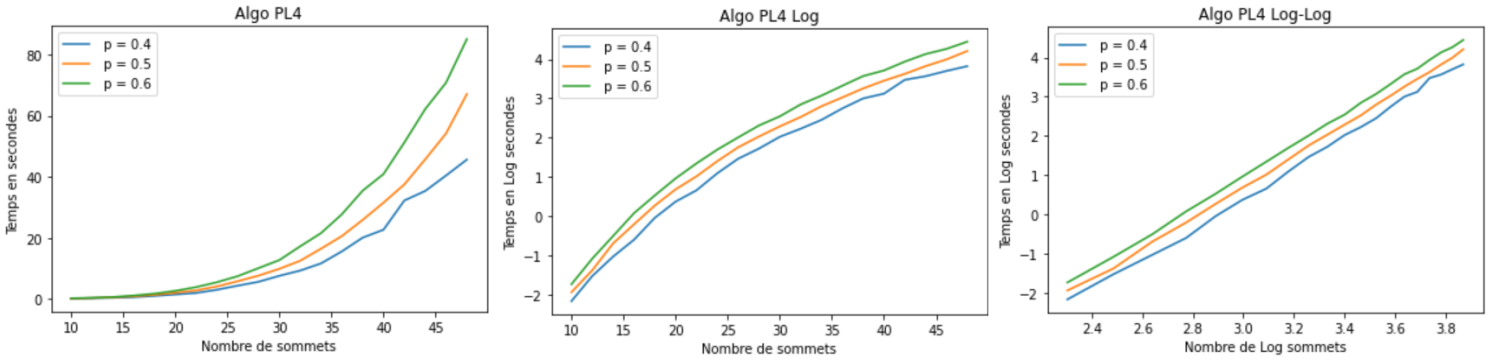
ABITBOL Yossef , 3804139
ZARROUK Yaël, 21107919

VII. Comparaisons des algos de type IV:

Les graphiques suivants ont été effectué avec un nombre de sommets qui varient entre 10 et 50 sommets pour un p qui varie de 0.4 à 0.6 par pas de 0.1. Chaque test a été effectué 10 fois pour pouvoir faire un moyenne.



Les graphiques suivants montrent les résultats de l'algo TYPE IV utilisant l'algorithme de Dijkstra. On peut voir sur la figure de gauche que le temps de réponse est assez rapide comparé aux autres algorithmes vu plus haut, cela peut être expliqué par le fait qu'on ne parcourt pas les arêtes pour changer leur valeur et nous utilisons directement le graphe classique. Plus le nombre de sommets augmente et plus p augmente , plus le temps augmente. De plus, la fonction plus ou moins affine sur la figure de droite fait référence à une complexité polynomiale de coefficient directeur 4.



Les graphiques suivants montrent les résultats de l'algo TYPE IV utilisant la programmation linéaire. On peut voir sur la figure de gauche que le temps de réponse est très lent comparé aux algorithmes plus haut et surtout comparé au calcul du même type mais d'une manière différente. Plus le nombre de sommets augmente et plus p augmente , plus le temps augmente. De plus, la fonction affine sur la figure de droite fait référence à une complexité polynomiale de coefficient directeur 4.

On a montré que le calcul du type IV par l'algorithme de dijkstra **était bien plus rapide** que le calcul par programmation linéaire et que les deux algorithmes ont une complexité polynomiale de degré 4.

IX. CONCLUSION

Pour conclure, on a vu comment calculer 4 types de chemins minimaux pour des multigraphes avec des contraintes de temps en transformant ce multigraphe en un graphe classique qu'on a adapté pour chaque type afin de répondre à nos besoins, en changeant la valeur des arêtes. En utilisant ensuite l'**algorithme de Dijkstra** cela nous a permis d'obtenir des résultats optimistes et efficaces en **complexité polynomiale**. On a aussi remarqué que les 4 types étaient calculables **rapidement,** le plus rapide étant le type 4 suivi du type 3 ensuite du type 2 et 1 jusqu'à un certain n dans les environs de 200. Plus le nombre de sommets n et plus la probabilité d'apparitions p des arcs est élevé, plus nos algorithmes prennent du temps. Enfin, on a vu que résoudre le chemin de type IV par programmation linéaire **prenait beaucoup plus de temps** que d'utiliser l'algorithme de Dijkstra.