Rapport Projet MOGPL 2021-2022

Tristan BERSOUX, Luka LAVAL

Version : décembre 2021

Table des matières

1 Question 1

Dans cette section nous nous basons sur l'instance de la figure gauche de l'exemple 1 de l'énoncé.

1.1 Assertion 1

"Un sous chemin préfixe d'un chemin d'arrivée au plus tôt peut ne pas être un chemin d'arrivée au plus tôt"

Soit P =[(a,b,2,1),(b,g,3,1),(g,k,6,1)], qui est un chemin d'arrivée au plus tôt de A vers G, et soit le sous chemin préfixe P'=[(a,b,2,1)]$\subset P$. P' n'est pas un chemin d'arrivée au plus tôt de A vers B, car il existe P"=[(a,b,1,1)]. Ainsi, P' est sous chemin préfixe d'un chemin d'arrivée au plus tôt mais n'est pas chemin d'arrivée au plus tôt. □

1.2 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"

Soit P=[(a,i,10,1),(i,l,8,1)], qui est un chemin de départ au plus tard de a vers l, et soit le sous chemin postfixe P'=[(i,l,8,1)]$\subset P$. P' n'est pas un chemin de départ au plus tard de i vers l car il existe P"=[(i,l,9,1)]. Ainsi, P' est sous chemin postfixe d'un chemin de départ au plus tard mais n'est pas chemin de départ au plus tard. □

1.3 Assertion 3

"Un sous chemin d'un chemin le plus rapide peut ne pas être un chemin le plus rapide"

Soit P=[(a,c,4,1),(c,h,6,1),(h,i,7,1),(i,l,8,1)], chemin le plus rapide de a à l, et soit P'=[(a,c,4,1),(c,h,6,1),(h,i,7,1),(i,l,8,1)]$\subset P$. P' n'est pas le chemin le plus rapide de a à i car P"=[(a,i,10,1)] existe. Ainsi, P' est sous chemin d'un chemin le plus rapide mais n'est pas le chemin le plus rapide. □

1.4 Assertion 4

"Un sous chemin d'un plus court chemin peut ne pas être un plus court chemin"

Soit P=[(a,f,3,1),(f,i,5,1),(i,l,9,1)] le plus court chemin de a à l, et P'=[(a,f,3,1),(f,i,5,1)]$\subset$P. P' n'est pas le plus court chemin de a à i car P"=[(a,i,10,1)] existe. Ainsi, P' est sous chemin d'un chemin le plus court mais n'est pas le chemin le plus court. □

2 Question 2

2.1 Approche générale

Pour chacun des chemins nous avons utilisé l'algorithme de Dijkstra non-modifié. La différence entre les propositions ci-dessous réside dans l'adaptation du graphe transformé $\tilde{G}$ avant application de Dijkstra, la sélection du sommet de départ pour le graphe transformé, puis dans la sélection du meilleur chemin parmi ceux retournés par Dijkstra. Il est important de remarquer que la transformation du graphe nous garantit que les chemins potentiels sont légaux, il permet de supprimer tous les chemins absurdes qui ne respecteraient pas la linéarité de la notion de temps du problème.

2.2 Chemin d'arrivée au plus tôt

Soit G un graphe, on cherche le chemin d'arrivée au plus tôt de A à Z dans l'intervalle $[ta, tz]$.

On transforme le graphe G en $\tilde{G}$ comme explicité dans l'exemple 2 de l'énoncé. Pour cela on parcourt les arcs et les sommets de G pour construire le tableau $\tilde{V}$ explicité au même exemple. Une fois le tableau $\tilde{V}$ construit, on parcourt les listes de $\tilde{V}$. Si la liste est liée à A, on parcourt cette liste en ajoutant les sommets à $\tilde{G}$ et en les reliant par un arc de valeur 0 au sommet suivant (de valeur la plus petite des valeurs supérieur) s'il y en a un. Tout cela uniquement pour les éléments dont valeur numérique associée est supérieur ou égale à ta. On a le même cas particulier pour Z sauf qu'on vérifie que la valeur associée aux éléments de la liste sont inférieur ou égaux à tz. Enfin le cas général, on parcourt les classes de sommets (les listes de $\tilde{V}$) en ajoutant les sommets à $\tilde{G}$ et en les reliant par un arc de valeur 0 au sommet suivant s'il y en a un.
Jusqu'ici on a construit les arcs liants les sommets d'une même classe. Maintenant on construit le reste des arcs en parcourant la liste des arcs de G qui vérifient dans le cas où l'arc est lié à A ou Z qu'il respcte l'intervalle.
À présent $\tilde{G}$ est construit, le sommet de départ pour l'algorithme de Dijkstra correspond au sommet de $\tilde{G}$ de classe A ayant la plus petite valeur associée.
On applique l'algorithme de Dijkstra à $\tilde{G}$ depuis le sommet sélectionné précédemment.
Parmi les chemins renvoyés par Dijkstra, on sélectionne celui dont le sommet d'arrivée est de classe Z et a la valeur associée la plus petite. Il correspond au chemin d'arrivée au plus tôt depuis A jusqu'à Z. On gardera ensuite de ce chemin uniquement un sommet par classe d'équivalence en priorisant les sommets de grande valeur (remarquons qu'il n'y aura qu'un sommet dans la classe d'équivalence de Z sur ce chemin).

2.3 Chemin de départ au plus tard

Soit G un graphe, on cherche le chemin de départ au plus tard de A à Z dans l'intervalle $[ta, tz]$.

L'algorithme est sensiblement similaire au précédent. La différence réside d'abord dans la transformation du graphe. Pour relier les sommets de classe A entre eux, on va les lier de la valeur la plus grande à la valeur la plus petite avec des poids supérieur à la somme de tous les poids du graphe. Tout le reste du graphe se construit comme précédemment.
L'autre différence est lors du choix du sommet de départ dans $\tilde{G}$, cette fois-ci on prend le sommet de classe A de plus grande valeur.
Enfin on va séléctionner le chemin arrivant en un sommet de classe Z de plus grande valeur. On gardera ensuite de ce chemin uniquement un sommet par classe d'équivalence en priorisant les sommets de grande valeur.

2.4 Chemin le plus rapide

Pour un chemin le plus rapide, dans $\tilde{G}$, les arcs entre deux sommets de la même classe n'ont plus un poids de 0, mais un poids correspondant au nombre de jours d'attente. On applique donc la transformation proposée dans l'énoncé avec, pour chaque arc entre paire de sommets$\{(s_1, t_1), (s_2, t_2)\}$,un poids $t_2 - t_1$ si $s_1 = s_2$ et un poids λ correspondant à la distance entre s_1 et s_2 si $s_1 \neq s_2$. Une exception est faite si s_1 et s_2 sont des sommets de départ possible : En effet, si on part de A, l'attente en A ne doit pas rentrer dans le calcul de la durée du voyage. Ainsi, les arcs entre les sommets de départ possibles ont un poids nul. De cette manière, lors de l'appel à Dijkstra, on s'assure qu'une attente en un sommet est bien prise en compte dans le calcul de la durée, sans que le départ dès le premier jour soit priorisé sur les départs ultérieurs. On lance ensuite Dijkstra sur le sommet de départ ayant la date au plus tôt dans $\tilde{G}$. On récupère le chemin qui a le plut petit coût total et qui arrive à un sommet de la classe du sommet d'arrivées.

2.5 Plus court chemin

Soit G un graphe, on cherche le plus court chemin de A à Z dans l'intervalle $[ta, tz]$.

On construit $\tilde{G}$ de la même manière que pour le chemin d'arrivée au plus tôt. On applique Dijkstra à $\tilde{G}$. On récupère le chemin qui dont le sommet d'arrivée appartient à la classe de Z et qui a le pus faible coût total.

3 Question 3

Les quatre types de chemins sont trouvés à l'aide des transformations de graphe de proposées ci-dessus, d'une recherche d'un sommet de départ, de l'algorithme de Djikstra, puis d'une selection de meilleur chemin parmi ceux proposés par l'algorithme de Dijkstra. Les algorithmes étant très proches, leurs complexités sont naturellement les mêmes.

3.1 Transformation du graphe

La première étape est la construction du tableau explicité dans l'exemple 2 dans l'énoncé en $O(E+V)$. S'en suit le parcours du tableau construit pour créer les arcs de poids 0 (entre une même classe de sommets) en $O(E)$. Enfin le parcours des arcs pour lier les nouveaux sommets aux sous-sommets de classes différentes en $O(E)$.
On a donc une compléxité en $O(E+V)$ pour la transformation de G.

3.2 Sélection du sommet de départ

Pour la suite on note respectivement E' et V' le nombre d'arcs et de sommets du graphe après transformation. La selection du sommet de départ se fait en parcourant la liste des nouveaux sommets en $O(V')$.

3.3 Algorithme de Dijkstra

L'algorithme de Dijkstra a une complexité en $O((E'+V') * log(V'))$.

3.4 Sélection du meilleur chemin

Dans cette partie nous étudions la complexité de la sélection du meilleur chemin parmi ceux retournés par l'algorithme de Dijkstra. Pour cela un simple parcours de la liste des chemins renvoyés suffit soit une complexité en $O(V')$.

3.5 Complexité totale

Au final, la complexité pour chacun des algorithmes proposés est en $O((E'+V') * log(V'))$.

3.6 En plus...

Une autre façon de faire pour les types 1 et 2 (qui a été implémentée) est l'utilisation de BFS. En effet, pour ces chemins, la valuation des arcs ne compte pas, et on peut alors utiliser BFS pour trouver les sommets d'arrivée au plus tôt ou au plus tard. Néanmoins notre implémentation est peu efficace (Car beaucoup de chemins sont générés, la borne supérieur du nombre de chemin étant $2^{NSommets}$), et n'est laissée ici que comme "bonus". La complexité ne sera pas calculée précisément, mais on l'estime exponentielle en nombre de sommets (à cause du nombre de chemins possibles).

4 Question 5

4.1 Modélisation d'un PCC de type 4 en programmation linéaire

Pour passer à un problème linéaire, on part du graphe non-transformé G.
Dans le PL, les variables de décision seront les arcs du graphe avec leurs poids t. Ainsi pour le graphe de l'exemple 2 de l'énoncé, les variables de décision sont : $ab1$, $ab2$, $ac2$, $ac4$, $bf5$, $cf6$ et $cg7$. Si un arc est emprunté, il aura sa variable mise à 1 sinon à 0.

Intéressons nous maintenant à la fonction objectif : on est naturellement dans un problème de minimisation puisqu'on veut le plus court chemin d'un sommet à un autre. L'idée de plus court chemin se réfère au poids λ de chaque arc. Ainsi notre fonction objectif sera la somme des arcs multipliés par leur poids respectif λ. Ainsi dans notre exemple la fonction objectif s'écrit : $Min\ z = ab1 + ab2 + ac2 + ac4 + bf5 + cf6 + cg7$.

Passons maintenant aux contraintes. On veut d'abord influer sur les arcs du sommet de départ et du sommet d'arrivée. On veut que au moins un des arcs partant du sommet de départ et au moins un des arcs arrivant dans le sommet d'arrivée soient empruntés pour forcer un chemin entre ces deux sommets. Dans notre exemple pour un chemin de a à f, ces contraintes se traduisent par : $ab1 + ab2 + ac2 + ac4 >= 1$ et $bf5 + cf6 >= 1$.

Il reste les contraintes liées à la structure du graphe. On veut imposer qu'un arc peut être emprunté si et seulement si au moins un des arcs menant à lui l'a été (sauf pour le sommet de départ qui n'a pas à avoir ce genre de contrainte). Evidemment cette contrainte ne peut lier deux arcs entre eux uniquement si c'est possible, c'est-à-dire si le temps de départ $t1$ de l'arc entrant additionné au temps de traversé $\lambda 1$ de l'arc entrant est inférieur ou égal au temps de départ $t2$ de l'arc sortant du sommet en question. Dans le cas contraire, on ne tient pas compte de l'arc entrant dans la contrainte. Ainsi pour notre exemple nous avons les contraintes suivantes qui s'ajoutent : $bf5 >= ab1 + ab2$, $cf6 >= ac2 + ac4$ et $cg7 >= ac2 + ac4$.

Enfin il ne manque plus que de faire respecter l'intervalle imposé. Pour cela, on force toutes les variables liées aux sommets de départ ou d'arrivée en dehors de l'intervalle à être mises à 0 ou on ne tient pas compte de celle-ci. Si on utilise le premier choix, pour l'exemple précédent avec un intervalle $[2, 7]$, on ajoute les contraintes suivantes : $ab1 <= 0$ et $cf6 <= 0$. Lorsqu'on vérifie que la variable est bien dans l'intervalle il ne faut pas oublier d'additionner le t au λ pour un arc entrant dans le sommet d'arrivée.

Pour résumer, on a le PL suivant pour un chemin de a à f dans l'intervalle $[2, 7]$:

$$
\begin{aligned}
Minz = ab1 + ab2 + ac2 + ac4 + bf5 + cf6 + cg7 \\
ab1 + ab2 + ac2 + ac4 \geq 1 \\
bf5 + cf6 \geq 1 \\
bf5 \geq ab1 + ab2 \\
cf6 \geq ac2 + ac4 \\
cg7 \geq ac2 + ac4 \\
ab1 \geq 0 \\
cf6 \geq 0
\end{aligned}
$$

4.2 Programmation avec Gurobi

Pour l'implémentation de l'idée ci-dessus, on a réutilisé le programme fournit lors du TME. Il nous suffit donc de calculer la matrice des contraintes a, le second membre b et les coefficients de la fonction objectif c. Bien-sûr il est impératif de passer à un problème de minimisation ($m.setObjective(obj, GRB.MINIMIZE)$) pour des variables de décision binaires ($vtype = GRB.BINARY$).

5 Question 6

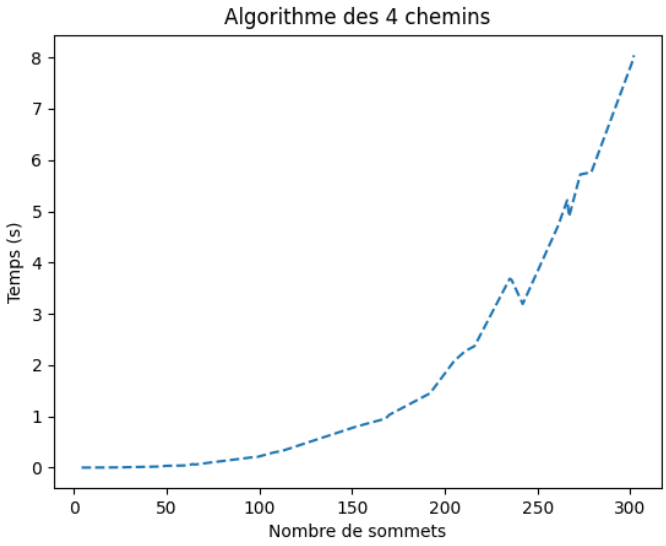

FIGURE 1 – Temps d'exécution de la fonction donnant les 4 types de chemins en fonction du nombre de sommets

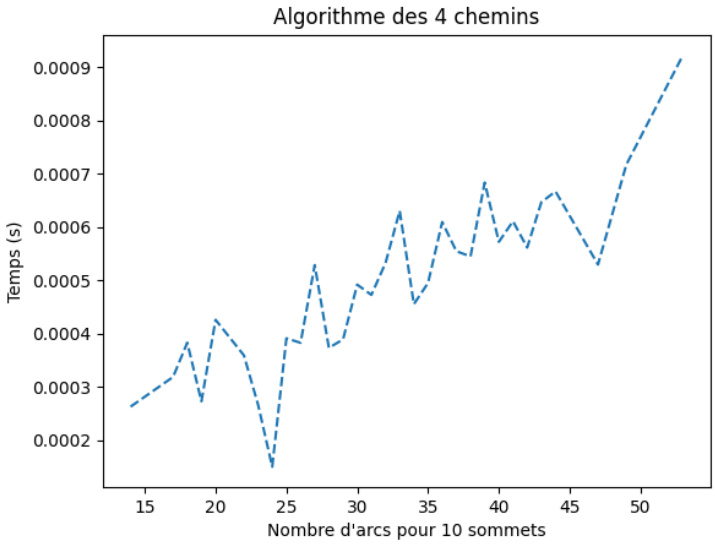

FIGURE 2 – Temps d'exécution de la fonction donnant les 4 types de chemins en fonction du nombre d'arêtes, avec 10 sommets

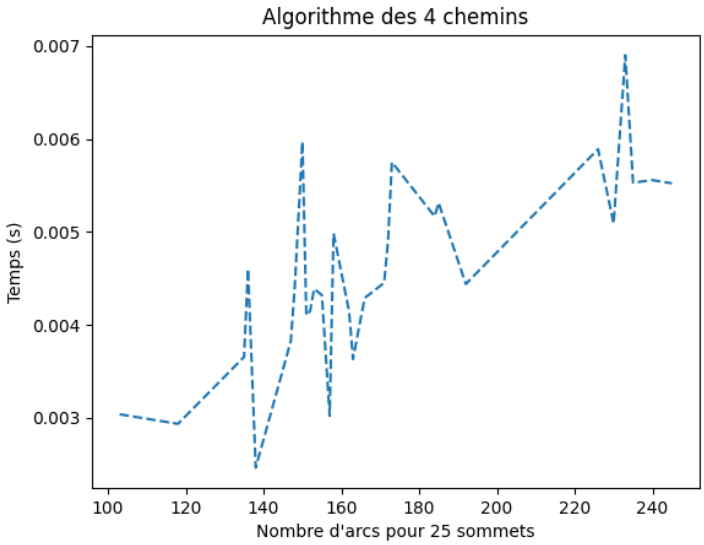

FIGURE 3 – Temps d'exécution de la fonction donnant les 4 types de chemins en fonction du nombre d'arêtes, avec 25 sommets

Comme on pouvait s'y attendre avec le calcul théorique de la complexité, on constate qu'augmenter le nombre de sommets ou le nombre d'arcs augmente, la courbe pour le nombre de sommet ayant une allure quadratique (Les courbes dépendant des arcs sont difficilement interprétables, car même en répétant de nombreuses fois l'expérience, les résultats obtenus sont très variables).

Nous n'avons pas ajouté la courbe où l'on fait varier le poids des arcs, puisqu'il semble qu'il n'y ait aucun lien entre poids des arcs et temps d'exécution.

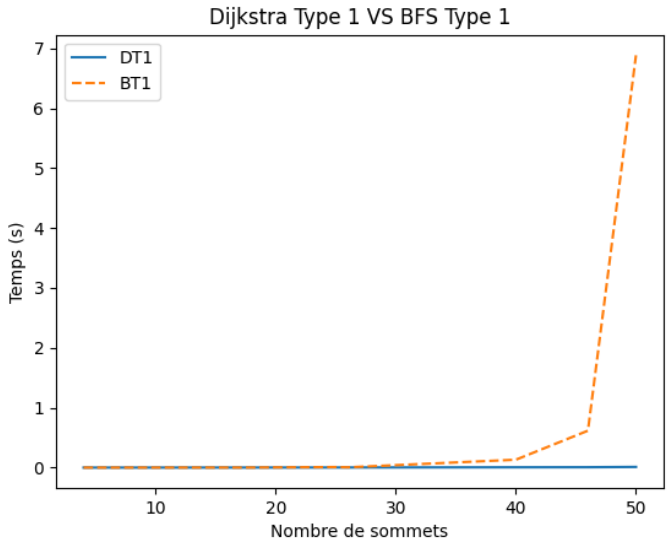

FIGURE 4 – Temps d'exécution de la résolution du problème du chemin de type 1, en utilisant Dijkstra (bleue), et notre tentative avec BFS (orange)

Comme on peut le constater avec cette courbe, notre tentative de solution utilisant BFS s'est soldée par un échec puisque la complexité temporelle est bien trop élevée. Le gain que nous espérions par rapport à notre implémentation n'était pas si élevé,et le temps venant à manquer, nous avons décidé de la laisser de côté.

6 Question 7

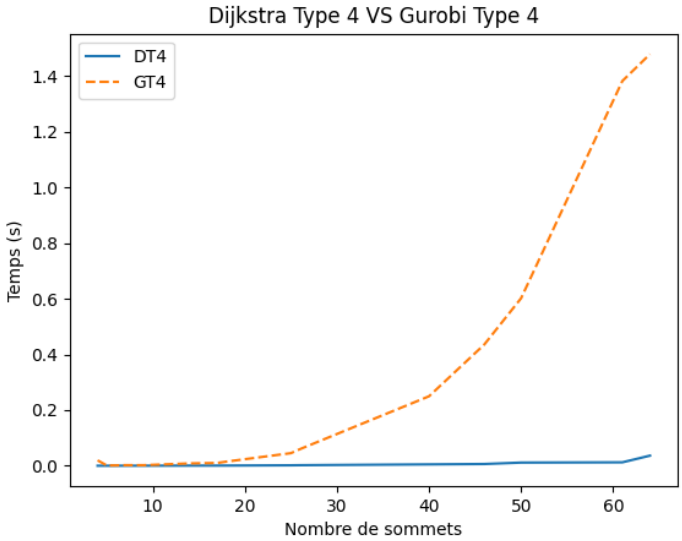

FIGURE 5 – Temps d'exécution des deux différents algorithmes pour le type 4, la version avec transformation de code et Dijkstra (bleue), et la version avec Gurobi (orange)

Comme on peut l'observer, la fonction utilisant Gurobi est bien plus lente que la fonction utilisant la transformation proposée. En effet, avec l'augmentation du nombre de sommets (et donc d'arcs), on augmente considérablement à la fois le nombre de variables du programme linéaire, mais aussi le nombre de contraintes. La mise sous forme de programme linéaire pour ce problème de plus court chemin est donc très inefficace.

7 Conclusion

Nous avons proposé quatre algorithmes similaires de calcul de quatre types de chemins différents. Cela nous a permis de constater que les applications de l'algorithme de Dijkstra sont très variées. La transformation de graphe proposée est intéressante, et permet de grandement simplifier la résolution. En passant un peu plus de temps sur ce projet, nous aurions aussi pu tester des méthodes probabilistes ou d'autres genre (Nous n'avons que des esquisses d'idées, trop floues pour tenter une implémentation). Enfin, passer par un programme linéaire est bien trop coûteux, et n'est pas envisageable pour ce type de problème.