Faculté des Sciences et Ingénierie - Sorbonne université

Master Informatique parcours - IMA / DAC

MOGPL - Modélisation, Optimisation, Graphes, Programmation linéaire

Rapport de projet

Amélioration de l'algorithme de Bellman-Ford

Réalisé par :

MAOUCHE Mounir - M1 IMA

SAID Faten Racha - M1 DAC

Supervisé par :

Thomas Bellitto

Evripidis Bampis

Patrice perny

Décembre 2023

TABLE DES MATIÈRES

TABLE DES FIGURES

INTRODUCTION

Dans divers contextes de la vie réelle, la connaissance des plus courts chemins entre différents points d'un réseau revêt une importance cruciale. Que ce soit pour optimiser les itinéraires de transport, minimiser les coûts de communication dans un réseau informatique, ou encore planifier des trajets efficaces dans des systèmes logistiques, la résolution du problème des plus courts chemins offre des avantages significatifs en termes d'efficacité, d'économie de ressources et de gain de temps.

À cet effet, différents algorithmes ont été introduits afin de résoudre ce problème, visant à trouver les chemins les plus courts entre deux points dans un réseau, représenté sous forme de graphe.

Notre projet se base sur une étude visant à améliorer l'algorithme de Bellman-Ford en introduisant une étape de prétraitement dans le but de réduire son temps d'exécution. Ainsi, ce document repose sur trois axes principaux ; le premier définit la problématique du sujet, le second présente l'approche de résolution avec prétraitement et la compare à l'approche classique, et enfin le troisième qui décrit le déroulement des tests et les principales fonctions implémentées.

PROBLÉMATIQUE

L'algorithme de Bellman-Ford est un algorithme classique et efficace pour le calcul des plus courts chemins sur des graphes à pondérations à la fois positives et négatives. Il a la propriété de converger vers la solution optimale en au plus $k = n - 1$ itérations ; avec n le nombre de sommets du graphe.

Sachant que k dépend de l'ordre dans lequel l'algorithme choisit de traiter les sommets à chaque itération, est-il possible d'implémenter une approche permettant de réduire le nombre k d'itérations nécessaires à sa convergence ?

CHAPITRE 1

APPROCHE DE RÉSOLUTION

1.1 Modélisation

La partie algorithmique de ce projet a été implémentée en Python en utilisant les structures de données natives du langage. Nos graphes seront représentés sous forme de listes des sommets et des arêtes, dont exemple est donné dans ce qui suit :

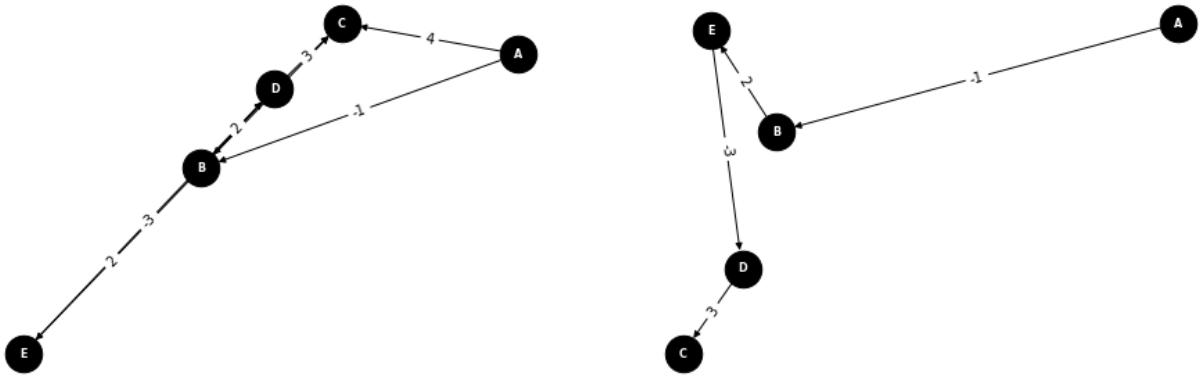

FIGURE 1.1 – Exemple de graphe et son arborescence des plus courts chemins.

Nous modélisons le graphe G de la figure 1.1 par :

$$G = [5, \{'A','B','C','D','E'\}, 8, \{('A','B') : -1, ('A','C') : 4, ('B','C') : 3, ('B','D') : 1, ('D','B') : 2, ('B','E') : 2, ('E','D') : -3, ('D','C') : 3\}]$$

Notre structure de graphe se présente sous la forme d'une liste de 4 éléments, comme suit : le nombre de sommets du graphe, un *set* de ses sommets, le nombre de ses arêtes, et un *dictionnaire* de ses arêtes. Chaque arête est une clé représentée pas un tuple $(sommet1, sommet2)$ et est associée à une valeur qui est son poids.

L'arborescence des plus courts chemins de G à partir du sommet source A, est représentée par la figure 1.1.

$$Arborescence = \{('A','B') : -1, ('B','E') : 2, ('E','D') : -3, ('D','C') : 3\}.$$

1.2 Question 1

Nous rappelons ci-dessous le principe de l'algorithme de Bellman-Ford :

Algorithm 1 Algorithme de Bellman-Ford

Entrée : G(E, V) : Graphe sans circuit négatif

Sortie : Arborescence des plus courts chemins, nombre d'itérations avant convergence

pred : Dictionnaire associant à chaque sommet son parent dans l'arborescence des plus courts chemins

etiquette : Dictionnaire associant à chaque sommet le poids de son plus court chemin

- Initialiser l'étiquette de la source à 0

- Initialiser les étiquettes des autres sommets à l'infini

```
for i in range(|V| + 1) do
    for each arc (u,v) in E do
        if etiquette[u] + poids(u,v) < etiquette[v] then
            etiquette[v] = etiquette[v] + poids(u,v)
            pred[v] = u
        end
    end
    if Aucun sommet modifié then
        break
    end
end
if i = |V| then
    # Circuit absorbant détecté
    Retourner -1
end
else
    Transformer pred en arborescence
    Retourner arborescence, i
end
Fin
```

La complexité de cet algorithme est de l'ordre de $O(n * m)$, tel que n est la taille du graphe et m est le nombre d'arêtes.

L'exemple affiché dans la figure 1.1 a été obtenu en utilisant cet algorithme.

1.3 Question 2

L'algorithme GloutonFas est comme son nom l'indique une méthode gloutonne proposée afin pour résoudre MVP dont le principe a été décrit dans l'énocé du sujet.

Algorithm 2 Algorithme GloutonFas

Entrée : un graphe orienté G

Sortie : une permutation des sommets de G

$s1 \leftarrow \emptyset,\ s2 \leftarrow \emptyset$

while *G n'est pas vide* **do**
  while *G contient une source u* **do**
    $s1 \leftarrow s1u$
    Retirer u de G
  end
  while *G contient un puits u* **do**
    $s2 \leftarrow us2$
    Retirer u de G
  end
  Choisir le sommet u qui maximise $\delta(u) = d^+(u) - d^-(u)$
  $s1 \leftarrow s1u$
  Retirer u de G
end
Retourner $s = s1s2$

La compléxité de GloutonFas est de $O(m*n^2)$; pour n le nombre de sommets du graphe en entrée et m le nombre de ses arcs.

Cet algorithme repose sur le principe de tri des sommets en commençant par les sommets sources et terminant par les sommets puits. Ceci peut s'apparenter au tri topologique de sommets, sauf que cette nouvelle approche traite le cas où tous les sommets du graphe possèdent à la fois des prédécesseurs et des successeurs. Elle sélectionne dans ce cas le sommet avec un maximum d'arêtes sortantes et un minimum d'arêtes entrantes ; soit le sommet u qui maximise $\delta(u) = d^+(u)d^-(u)$.

A titre d'exemple, nous appliquons GloutonFas sur l'instance de graphe donnée dans l'énoncé :

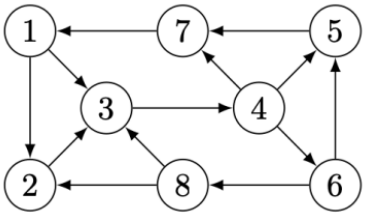

FIGURE 1.2 – Instance de graphe pour l'algorithme GloutonFas

Le graphe est représenté comme suit :

$$Ge = [8, \{1,2,3,4,5,6,7,8\}, 13, \{(1,2):1,(1,3):1,(2,3):1,(3,4):1,(4,5):1,(4,6):1,(4,7):1,(5,7):1,(6,5):1,(6,8):1,(7,1):1,(8,2):1,(8,3):1\}]$$

L'algorithme GloutonFas retourne l'ordre total suivant : $[4,6,5,7,1,8,2,3]$

Notons que certains sommets sont inversés par rapport à la solution de l'énoncé, cela s'explique par le fait qu'une fois les sommets 4, 6 et 5 successivement supprimés, l'algorithme a le choix entre les sommets 8 et 7 en tant que source, il peut donc soit :

— Supprimer 7 et ensuite choisir comme source entre les sommets 1 et 8 .

— Supprimer 8 et dans ce cas il va forcement sélectionner le sommet 7 à la prochaine itération ; car 7 est dans ce cas le seul sommet source restant. Et ainsi de suite jusqu'à ce que tous les sommets de G soient dans s.

1.4 Question 3

Génération d'un graphe orienté aléatoire

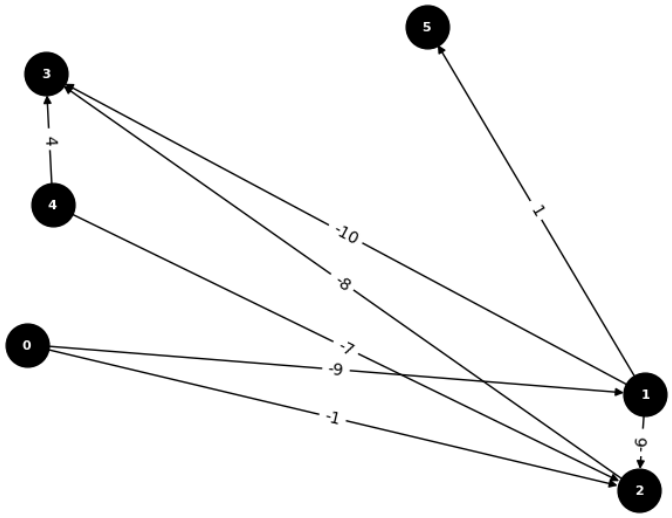

FIGURE 1.3 – Graphe généré aléatoirement

Création des graphes d'apprentissage Gi

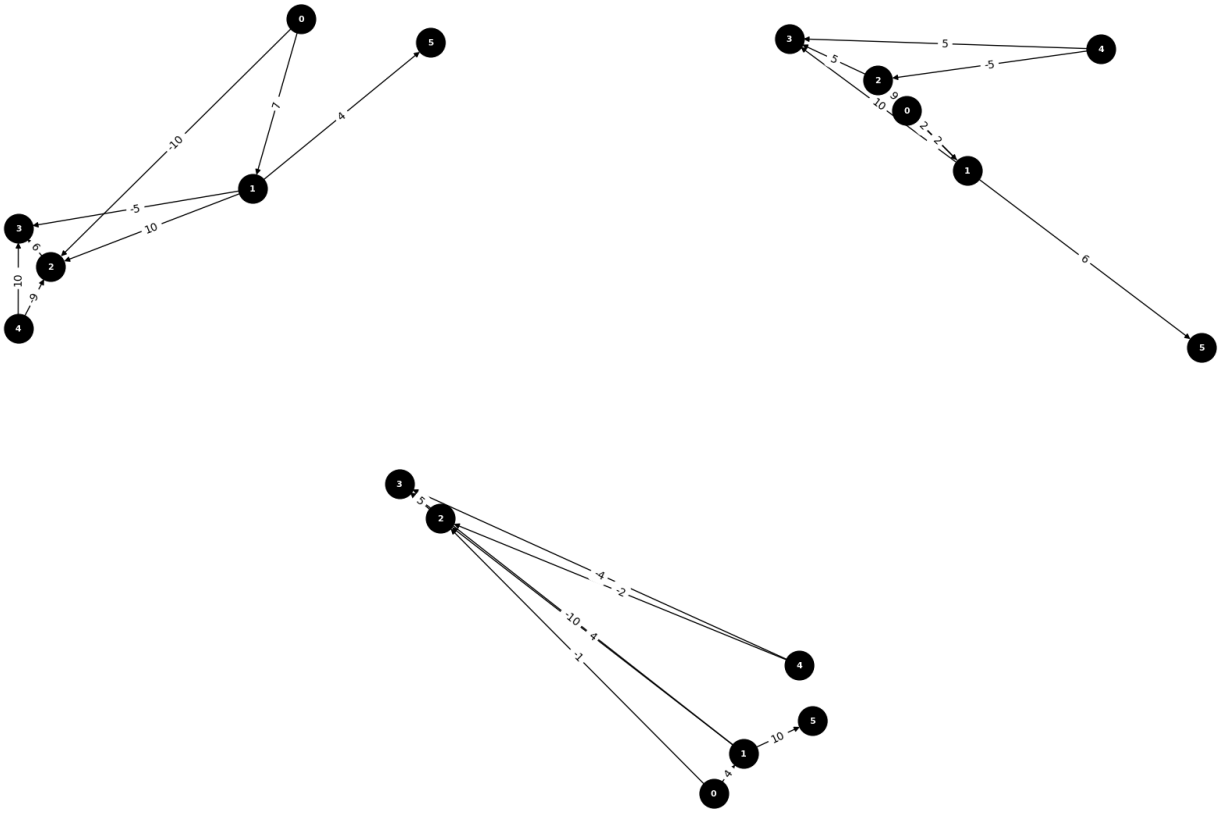

FIGURE 1.4 – Graphes d'apprentissage.

Création du graphe de test H

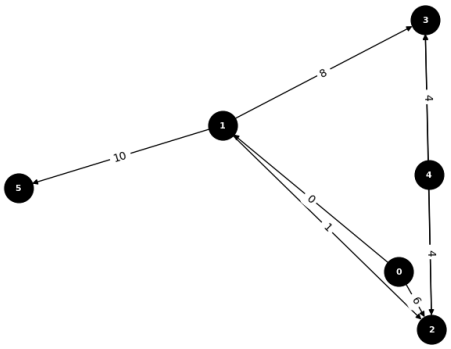

FIGURE 1.5 – Graphe de test

Critères sur les graphes

Comme demandé dans l'énoncé, nous ajoutons quelques restrictions sur les graphes que nous créons aléatoirement :

Nombre de sommets atteignables par la source

Pour chaque graphe créé, nous calculons sa source selon l'algorithme suivant :

Algorithm 3 Algorithme de détection de source

Entrée : G(E, V) : Graphe sans circuit négatif

Sortie : Un sommet source

sources_potentielles = Initialiser la liste aux sommets sans prédécesseur dans G

for each sommet in sources_potentielles **do**

  for each arc (u,v) in E **do**

    sommets atteignables= getSommetsAtteignables(G, source_potentielle)

    if *if len(sommets_atteignables)* $\geq$ *|V|/2*: **then**

      return *source_potentielle*

    end

  end

end

while source_potentielle != None **do**

  source_potentielle = sommet qui maximise $\delta(u) = d^+(u)d^-(u)$

  sommets_atteignables= getSommetsAtteignables(G, source_potentielle)

  if *if len(sommets_atteignables)* >= $|V|/2$ *:* **then**

    return source_potentielle

  end

end

Fin

Cet algorithme donne la priorité aux sommets qui ne possèdent aucun prédécesseur, et itère sur les sommets restants dans l'ordre de ceux qui maximisent la différence de leurs degrés.
Il fait appel à la fonction récursive **getSommetsAtteignables(G, s)** qui retourne les sommets atteignable à partir d'un chemin commençant par s.

Cette validation nous permet de nous assurer que le sommet de départ pour Bellman Ford permettra d'atteindre un nombre acceptable de sommets, et ne pas tomber dans des cas où le sommet utilisé comme source est isolé des autres.

La source trouvée est sauvegardée dans la structure de graphe afin de ne pas avoir à la recalculer plus tard.

Existence d'un circuit absorbant

Ensuite, nous vérifions qu'il n'existe pas de circuit absorbant dans le graphe en utilisant l'algorithme de Bellman-Ford.

Processus de Validation des graphes

Algorithm 4 Algorithme de génération d'un graphe valide

Entrée : n, d, intervalle : taille et densité du graphe et intervalle des poids des arcs

Sortie : Graphe avec source et sans circuit négatif

G = générer nouveau graphe aléatoire

while *G ne possède pas de source* **do**
| G = générer nouveau graphe aléatoire

end

while *G contient un circuit négatif* **do**
| Changer les poids du graphe G

end

return G

Fin

1.5 Questions 4 et 5

Union des arborescences des plus courts chemins

Afin de construire l'arborescence T demandée, nous allons dans un premier temps appliquer l'algorithme de Bellman-Ford sur chacun des Gi créés à la question précédente, puis faire l'union des arcs de toutes les arborescences obtenues, en leur affectant un poids de 1.
L'arborescence T obtenue à partir des instances précédentes est la suivante :

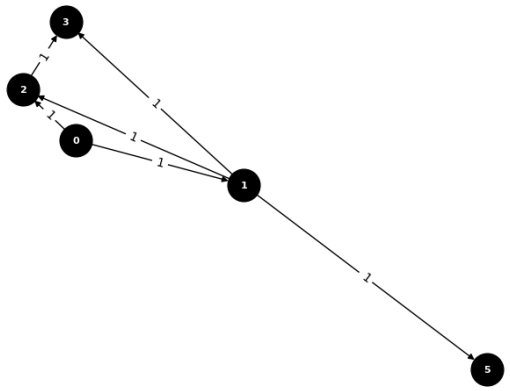

FIGURE 1.6 – Unions des arborescences des plus courts chemins pour les Gi

Ordre total retourné par gloutonFas : $\{0, 1, 2, 3, 5\}$

Nous pouvons remarquer que le sommet 4 n'apparaît pas dans l'arborescence et par conséquent dans l'ordre total, et ce car il est lui aussi une source au même titre que le sommet 0 et qui ne peut être atteint par celui-ci.

1.6 Questions 6 et 7

Afin de poursuivre nos expérimentaions, nous avons modifié l'algorithme de Bellman-Ford précédent en introduisant un ordre spécifique sur lequel mettre à jour les sommets.

Algorithm 5 Algorithme de Bellman-Ford avec ordre

Entrée : Graphe orienté $G(E, V)$, ordre total $ordre$

Sortie : Arborescence des plus courts chemins, Nombre d'itérations avant convergence k

pred : Dictionnaire associant à chaque sommet son parent dans l'arborescence des plus courts chemins

etiquette : Dictionnaire associant à chaque sommet le poids de son plus court chemin

- Initialiser l'étiquette de la source à 0

- Initialiser les étiquettes des autres sommets à l'infini

for i in range($|V| + 1$) **do**
    for each u in $ordre$ **do**
        for each arc (v, u) in E **do**
            if *etiquette[v] + poids(v,u)* $<$ *etiquette[u]* **then**
                etiquette[u] = etiquette[v] + poids(v,u)
                pred[u] = v
            end
        end
    end
    if *Aucun sommet modifié* **then**
        break
    end
end
if $i = |V|$ **then**
    # Circuit absorbant détecté
    Retourner -1
end
else
    - Transformer pred en arborescence
    Retourner $arborescence, i$
end

La complexité de l'algorithme Bellman-Ford avec ordre est en $O(m * n^2)$. Remarquons que cette dernière est certes dans le pire cas supérieure à celle de Bellman-Ford décrite dans la question 1, cependant, l'introduction de l'ordre permet de réduire le nombre d'itérations nécessaires à la convergence de l'algorithme, ce qui en pratique, rend la complexité avoisinant $O(m * n)$ avec un m plus petit que celui de Bellman-Ford sans ordre ; car seuls les arcs entrants sont considérés.

Application de l'algorithm Bellman-Ford en utilisant l'ordre total

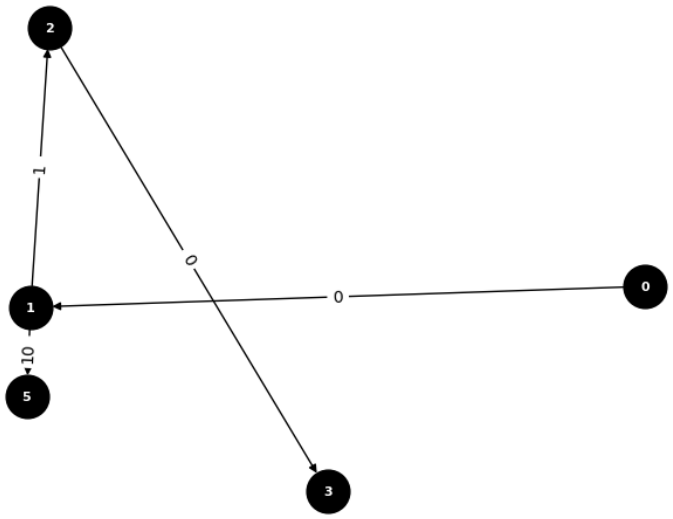

FIGURE 1.7 – Arborescence obtenue avec un ordre total

Application de l'algorithme Bellman-Ford en utilisant un ordre aléatoire

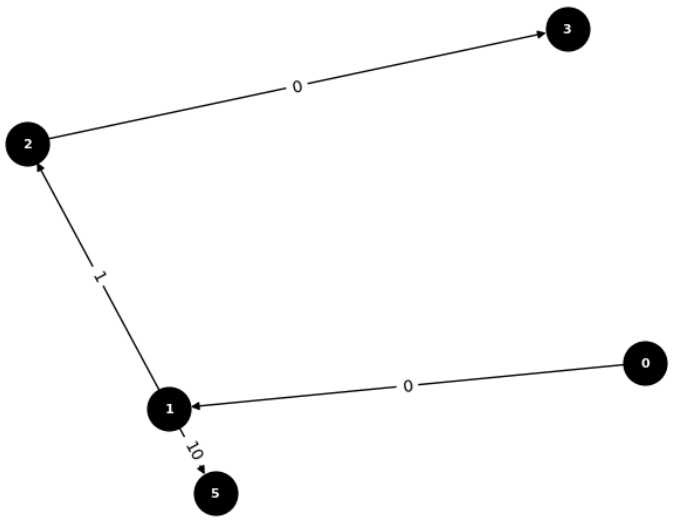

FIGURE 1.8 – Arborescence obtenue avec un ordre aléatoire

Les deux arborescences sont identiques car les plus courts chemins de chaque sommet de H à partir de la source considérée sont uniques, la seule différence étant le nombre d'itérations, 2 avec l'ordre total et 3 avec l'ordre aléatoire.

Remarque : Le sommet de départ de l'ordre aléatoire a été fixé au même sommet de départ de l'ordre total. En effet, si le sommet de départ de l'ordre aléatoire n'est pas forcé au même que pour l'ordre total, il se peut que le sommet alors sélectionné aléatoirement soit un sommet puits qui ne permettrait d'atteindre aucun autre sommet et donc de retourner une arborescence vide en une seule itération ce qui biaiserait les résultats.

1.7 Question 8

En se basant sur les résultats des deux questions précédentes, nous pouvons dire que sur cette instance aléatoire de graphe, l'approche basée sur un ordre total est meilleure que celle basée sur un ordre aléatoire, car $k\_total < k\_aleatoire$, avec k_total et $k\_aleatoire$ le nombre d'itérations selon respectivement la première et la deuxième approche.
Nous allons dans ce qui suit tenter de justifier ce résultat et plus tard effectuer des tests plus larges sur un nombre plus important et varié d'instances de graphes afin de confirmer notre proposition.
Comme mentionné dans la problématique, le nombre k d'itérations dépend de l'ordre dans lequel l'algorithme choisit de traiter les sommets à chaque itération, ainsi, choisir un bon ordre réduit la valeur de k.
L'ordre total est un bon ordre car il trie successivement les sommets en priorisant les sommets ayant un maximum d'arêtes sortantes et un minimum d'arêtes entrantes. Cette propriété est particulièrement intéressante étant donné qu'avoir des sommets en début de parcours qui ont un nombre minimal d'arcs entrants permet de minimiser les cas où un sommet en fin de parcours permet d'accéder à ces sommets-là avec un chemin plus court, car cela a pour effet de mettre à jour les sommets en début de parcours et donc tous les sommets suivants qui possèderont ces sommets-là dans leur plus court chemin.

1.8 Méthodologie de test

Afin de comparer de façon plus générale les résultats des deux approches, la première basée sur un ordre tiré aléatoirement et la seconde basée sur un ordre total, nous réalisons différents types de tests à travers lesquels nous faisons varier à chaque fois un paramètre afin d'étudier son impact sur le nombre d'itérations nécessaires à l'exécution de chacune des deux approches.

Les paramètres étudiés sont :

— n : le nombre de sommet du graphe G initial ; taille du graphe.

— p : la probabilité d'avoir un arc entre deux sommets ; densité du graphe.

— $borneInterval$: l'intervalle de valeurs sur lesquel les poids des arcs sont sélectionnés.

— Ni : nombre de Gi utilisés pour le prétraitement, soit la taille de l'ensemble d'apprentissage.

— $nb\_niveaux$: Nombre de niveaux du graphe. Le tableau suivant indique les intervalles de valeurs sur lesquels nous avons fait varié nos paramètres :

Paramètre	**N**	**P**	**borneInterval**	**Ni**	**nb_niveaux**
Intervalle	[2, 10]	[0.2, 0.8]	[2, 30]	[2, 120]	[4, 100]
Pas	1	0.2	4	4	30

TABLE 1.1 – Description des paramètres

1.9 Question 9

Afin d'effectuer les tests demandés, nous créons des instances de graphes aléatoires différentes en terme de taille, densité, ou valeurs de poids et les soumettons à tour de rôle à la procédure étudiée dans les questions précédentets afin de récupérer le nombre d'itérations de l'exécution de l'algorithme de BellmanFord avec un ordre total et avec un ordre aléatoire. Ces résultats sont sauvegardés dans une matrice tridimensionnelle (taille, densité, intervalle de poids) qui sera utilisée pour dessiner les graphes des performances des deux versions de l'algorithme selon chaque paramètre. Ces évaluations consistent à effectuer pour chaque paramètre séparément, une moyenne sur le reste des paramètres afin de récupérer le nombre d'itérations moyen en fonction de chaque valeur du paramètre.

Résultats et interprétations

Les résultats expérimentaux que nous présentons dans cette partie montrent à l'unanimité que l'approche basée sur un ordre total est plus performante, en terme de vitesse de convergence, que celle basée sur un ordre aléatoire. Nous allons dans ce qui suit tenter de justifier ce résultat.

1.9.1 Evaluation en fonction de la taille du graphe

D'après la figure 1.9, Nous pouvons observer que plus le nombre de sommets du graphe augmente plus le nombre d'itérations nécessaire à la convergence de Bellman-Ford augmente. Cela se justifie par le fait que plus nous avons de sommets moins nous avons de chance de les étiqueter de manière ordonnée (en niveaux) car plus de sommets implique plus de chance d'avoir des cycles et donc de devoir mettre à jour un même sommet sur plusieurs itérations.

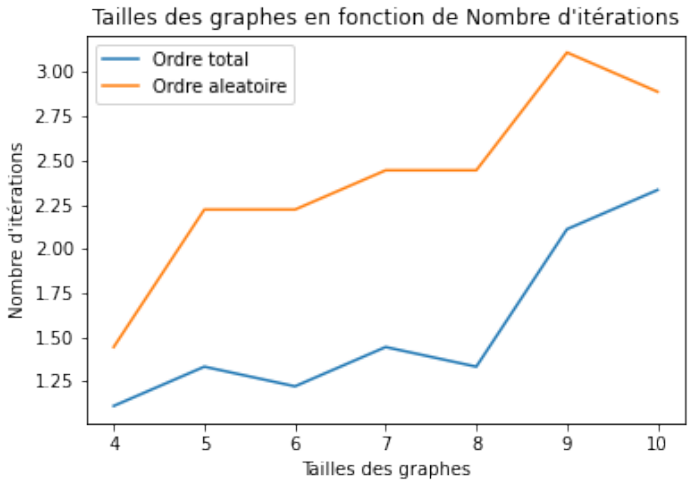

FIGURE 1.9 – Résultats de l'évaluation en fonction de la taille du graphe

1.9.2 Evaluation en fonction de la densité du graphe

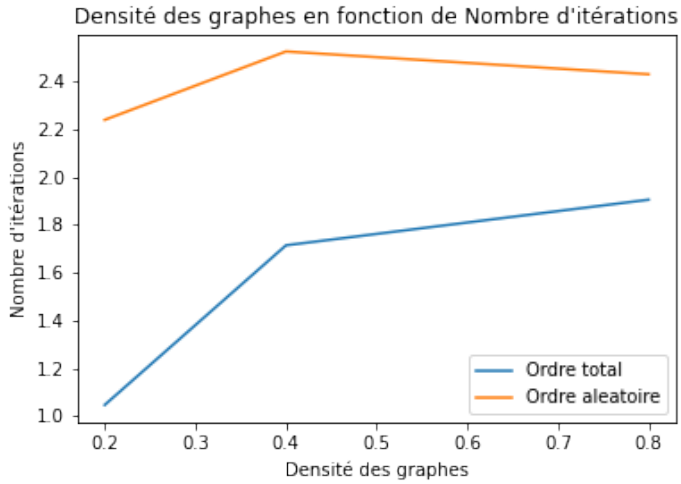

FIGURE 1.10 – Résultats de l'évaluation en fonction de la densité du graphe

Le nombre d'itérations augmente proportionnellement à la densité. En effet, plus il y a d'arcs dans le graphe plus il y a de liens entre les sommets et donc la mise à jour d'un sommet entrainera la nécessité de mettre à jour un plus grand nombre de sommets dans des itérations ultérieures.

1.9.3 Evaluation en fonction des intervalles de poids des arcs

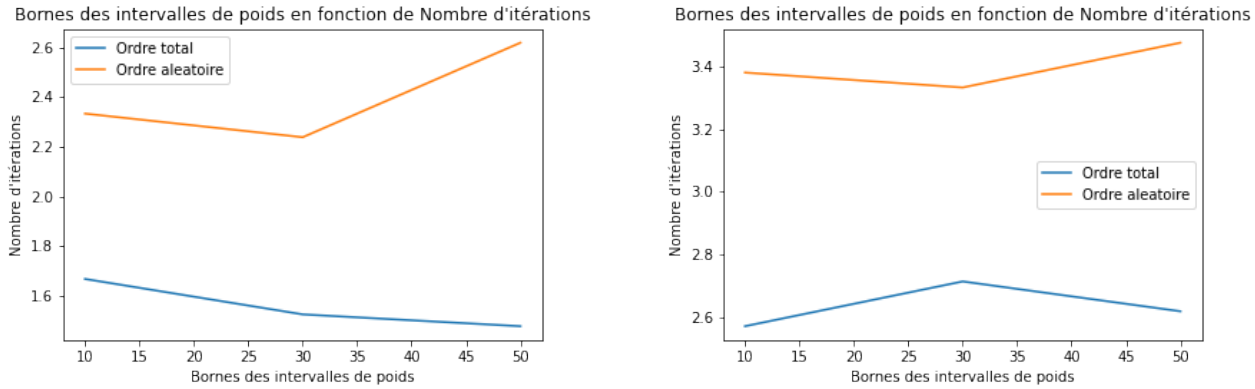

FIGURE 1.11 – Deux exécutions différentes de l'évaluation en fonction des intervalles de poids du graphe.

D'après les résultats obtenus, les bornes de l'intervalle de poids n'influencent pas le nombre d'itérations nécessaires car le calcul d'un plus court chemin dans un graphe sans circuit ne dépend pas des valeurs des poids.

1.10 Question 10

Pour les prochains tests, le but est de faire varier la taille de l'ensemble des graphes utilisé pour apprendre les plus courts chemins pour un même graphe.

Comme nous avons pu le voir dans les questions précédentes, la taille et la densité des graphes influencent la convergence de l'algorithme. Ainsi, nous allons effectuer nos tests sur différents cas de figure que nous présentons dans ce qui suit :

1.10.1 Graphe de petite taille et faible densité

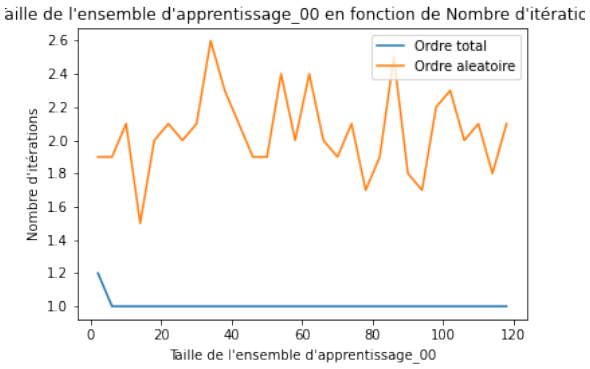

FIGURE 1.12 – Graphe de petite taille et faible densité

1.10.2 Graphe de petite taille et forte densité

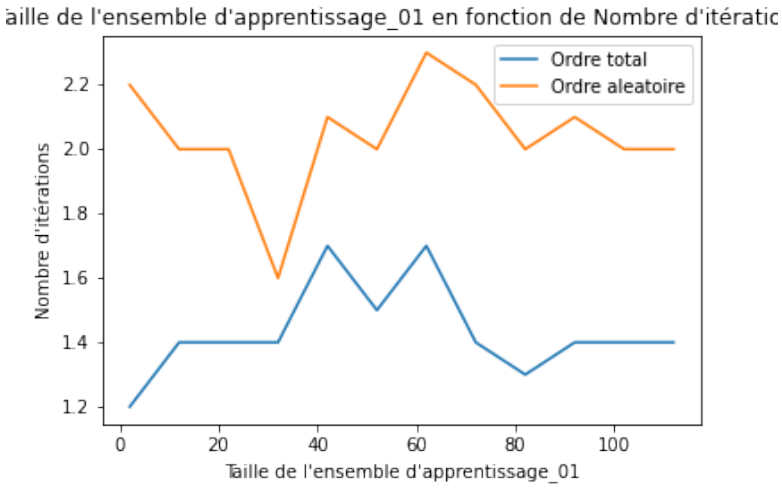

FIGURE 1.13 – Graphe de petite taille et forte densité

1.10.3 Graphe de grande taille et faible densité

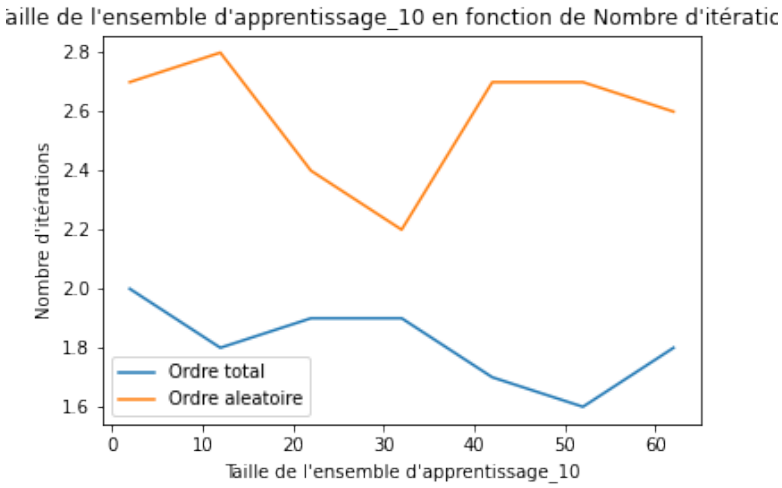

FIGURE 1.14 – Graphe de grande taille et faible densité

1.10.4 Graphe de grande taille et forte densité

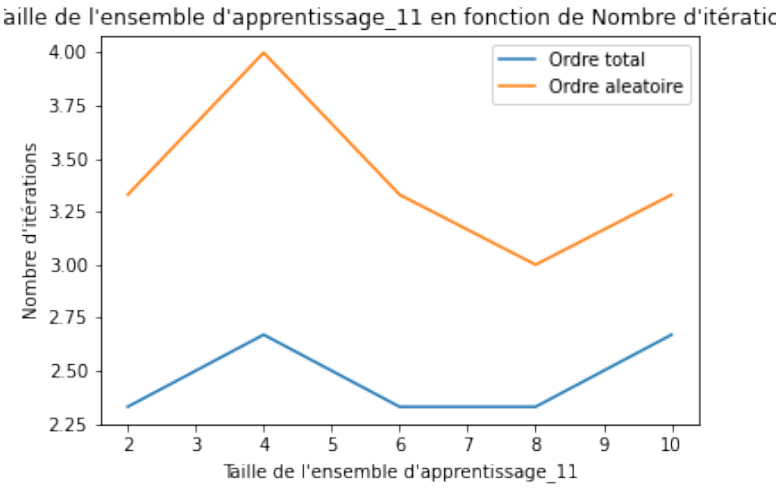

FIGURE 1.15 – Graphe de grande taille et forte densité

Les résultats obtenus étant très variables d'une exécution des tests à l'autre, cela nous pousse à croire que le nombre de graphes d'apprentissage n'a aucune influence sur le nombre d'itérations. Notons que pour le premier cas, l'algorithme converge après une itération seulement car cela est dû à la structure du graphe généré. Ce dernier étant petit et faiblement dense, les plus courts chemins sont plus facilement détectables, lorsqu'ils sont bien triés. En effet, quand on calcule le plus court chemin d'un sommet, il y a peu de chances qu'il soit mis à jour dans une itération suivante à cause d'une arête provenant d'un sommet pas encore traité.

1.11 Question 11

Afin de répondre à cette question nous avons implémenté une fonction qui génère un graphe en niveaux tel que les successeurs de tous les sommets d'un niveau j sont dans le niveau $j + 1$.

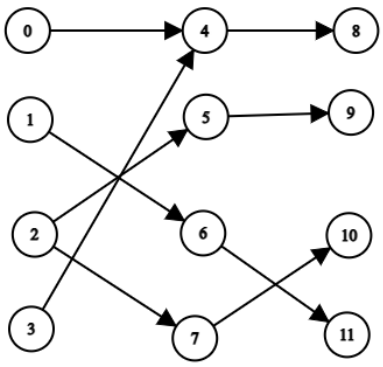

FIGURE 1.16 – Exemple d'un graphe à 3 niveaux

Une fois le graphe généré, nous avons créé une fonction qui fait varier le nombre de niveaux et calcul pour chaque niveau le nombre k d'itérations en moyenne nécessaires à la convergence de Bellman-Ford, à la fois sur un ordre total et sur un ordre aléatoire.

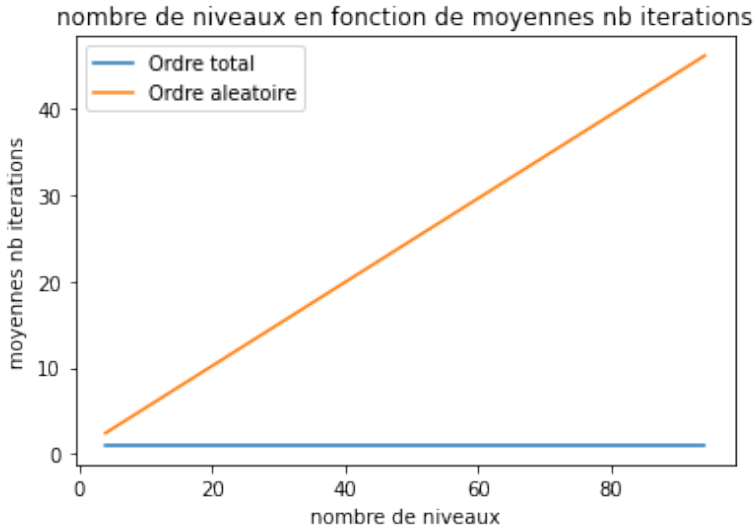

FIGURE 1.17 – Résultats de l'évaluation en fonction du nombre de niveaux

Les résultats de ces tests nous permettent de conclurent que la méthode avec prétraitement est adéquate à ce type de graphe. En effet, nous pouvons observer que sur un ordre total le nombre d'itérations est constant en le nombre de niveaux et est égal à 1. Cela signifie que l'algorithme n'a besoin que d'un parcours des sommets pour retrouver l'arbre des plus courts chemins. D'une autre part, sur un ordre aléatoire le nombre d'itérations augmente proportionnellement au nombre de niveaux du graphe. Cela s'explique par le fait qu'il ne parcourt pas les sommets niveau par niveau, ainsi, lorsqu'il traitera un sommet appartenant à un niveau dont le précédent n'a pas été traité (il saute un niveau), il ne le mettra pas à jour et devra attendre que le niveau précédent ait été traité dans une itération ultérieure.

Ces résultats sont suffisants pour généraliser à un graphe à 2500 niveaux.

Il est clair que le nombre d'itérations en ordre aléatoire est linéaire ($O(n)$) en le nombre de niveaux, dont la pente est égale à $\frac{1}{2}$, ce qui nous permet, par prolongement de la droite, d'estimer le nombre d'itérations nécessaire pour un graphe à 2500 niveaux à 1250 itérations.

Pour ce qui est de l'ordre total, son nombre d'itérations est constant ($O(1)$)

D'autre part, de part la manière dont le graphe est construit, on peut déduire par récursivité que la recherche d'un plus court chemin pour un graphe en niveaux de $n + 1$ niveaux se fait en un même nombre d'itérations que pour un graphe de n niveaux.

CONCLUSION GÉNÉRALE

Dans le cadre de notre projet, nous avons analysé le comportement de l'algorithme de Bellman-Ford soumis à différentes conditions en termes de taille, densité, et structure de graphe, et avons obtenu des résultats concluants quant à l'hypothèse qui a initié notre démarche, qui consistait à vérifier que le temps d'exécution de l'algorithme dépendait de l'ordre de parcours des sommets.

En effet, les résultats obtenus mettent en évidence l'importance de l'ordre des sommets dans lesquels l'algorithme s'exécute, en remarquant que le nombre d'itérations en utilisant un ordre total est dans la majorité des cas inférieur à celui obtenu en utilisant un ordre aléatoire. Ainsi, nous pouvons conclure que l'approche basée sur un ordre total améliore la convergence de l'algorithme de Bellman-Ford.

ANNEXE

Fichiers annexes

Les fichiers annexes à ce document comprennent :

— **Graphe.py** : ce fichier contient les différentes fonctions de base associées à la manipulation d'un graphe.

— **Solution.py** : ce fichier contient les fonctions implémentées pour répondre à l'énoncé du projet.

— **Tests.ipynb** : ce notebook présente le travail final. Il fait appel aux fonctions des deux fichiers précédents et contient les réponses à toutes les questions. Principales fonctions implémentées

Description des fonctions principales

Nous présentons dans ce qui suit un tableau qui décrit les principales fonctions implémentées.

Fonction	**Description**
bellmanFord/bellmanFordOrdre	Deux différentes implémentations de l'algorithme Bellman-Ford, la première est aléatoire et la seconde suit un ordre défini de sommets à traiter à chaque itération.
gloutonFas	Implémente l'algorithme GloutonFas donné dans l'énoncé.
genererGrapheValide	Retourne un graphe qui admet une source valide (vérifie la condition qui stipule qu'une source est source d'un sommet à partir duquel on peut atteindre au moins $\|V\|/2$ sommets) et qui ne contient pas de circuit absorbant.
construire_ListGi	Construit une liste contenant un nombre N_i de graphes G_i nécessaires à l'apprentissage. La fonction garantit que les G_i retournés ne contiennent pas de circuit négatif.
unionArborescence	Calcule l'arborescence des plus courts chemins de chaque G_i, fait l'union de leurs arêtes, et unifie tous les poids à 1.
getOrdreAleatoire	Génère un ordre aléatoire à partir de l'ensemble des sommets du graphe.
comparer_ordre_total_aleatoire	Génère une série de graphes aléatoires de différentes tailles, densités et bornes d'intervalles de poids, sur lesquels elle exécute l'algorithme de Bellman-Ford avec ordre total puis avec ordre aléatoire et sauvegarde les résultats selon chaque paramètre dans une matrice « résultats ».
genererGrapheNiveau	Génère un graphe qui respecte la structure demandée à la question 11 de l'énoncé ; soit un graphe en niveaux tel que les successeurs de tous les sommets d'un niveau j sont dans le niveau $j+1$.

TABLE 1.2 – Description des fonctions