Projet MOGPL

BENAHMED Neil Khalil
KADEM Hocine

9 décembre 2023

Table des matières

1 Problématique

L'algorithme de Bellman-Ford est utilisé pour résoudre le problème des plus courts chemins dans un graphe orienté pondéré sans circuits de poids négatif. Il permet de calculer les chemins les plus courts d'un sommet source à tous les autres sommets du graphe.

Dans cette étude, nous cherchons à affiner l'estimation du nombre d'itérations nécessaires pour que l'algorithme de Bellman-Ford converge. Pour ce faire, nous examinons l'impact d'un ordre spécifique des sommets dans le graphe sur le nombre d'itérations requis pour atteindre la convergence.

En modifiant l'ordre des sommets, nous explorons comment cela peut influencer la propagation de l'information sur les chemins les plus courts et donc le nombre d'itérations nécessaires pour obtenir les résultats finaux. Cette analyse nous permet de mieux comprendre les facteurs qui influencent l'efficacité de l'algorithme de Bellman-Ford dans différents contextes.

2 Introduction

Pour l'algorithme de Bellman-Ford sur le graphe G, le temps d'exécution avec l'ordre $< tot$ est $O(m \cdot \max_{v \in V} \text{dist}(< tot, Pv))$, soulignant l'importance d'un bon ordre des sommets.

Le choix de l'ordre pour Bellman-Ford implique la résolution du problème MVP (Minimum Violation Permutation). Il vise à trouver un ordre $< tot$ sur V minimisant $\max_{i \in [r]} \text{dist}(< tot, P_i)$. La résolution de MVP utilise une méthode gloutonne, GloutonFas, construisant un ordre linéaire en déplaçant les sommets avec un faible degré entrant au début et ceux avec un faible degré sortant à la fin.

3 Configuration expérimentale

3.1 Algorithme de Bellman-ford

La classe Bellman-Ford implémente l'algorithme de Bellman-Ford avec une condition de convergence. L'objectif est de construire les arbres des plus courts chemins depuis la source et d'arrêter l'algorithme dès que nous détectons une convergence.

Nous implémentant donc l'algorithme Greedyfas fourni pour trouver l'ordre $< tot$

3.2 Entraînement et tests

Dans nos expériences, nous commençons avec quatre graphes pondérés et orientés G_1, G_2, G_3 et H. Les graphes G_i serviront de jeu d'entraînement que nous utiliserons pour calculer une solution au problème MVP, et le graphe H représentera le test où nous évaluerons les performances de cette approche. Comme nous le verrons, plus les graphes G_i et H sont similaires, plus nous pourrons obtenir une amélioration significative.

Il reste à choisir le nœud de départ s. Remarquez que si s a un degré de sortie de 0, alors Bellman-Ford se terminera après une seule itération, peu importe l'ordre utilisé. Pour éviter de tels cas triviaux, nous commençons par identifier un ensemble de nœuds S dans H qui peuvent atteindre

Algorithm 1 Bellman-Ford

```
1: function BELLMAN-FORD(graphe, source)
2:     new_edges ← self.edges
3:     if order existe then
4:         new_edges ← utils.order_edges(self, order)
5:         self.edges ← new_edges
6:     dist ← {node : float('inf') for node in G.vertices}
7:     dist[src] ← 0
8:     paths ← {node : [] for node in G.vertices}
9:     paths[src] ← [src]
10:    converged_in ← 0
11:    for i dans la plage de 0 à len(G.vertices) - 1 do
12:        dist_copy ← dist.copy()
13:        for chaque arête (u, v, w) dans G.edges do
14:            if dist[u] ≠ float('inf') et dist[u] + w < dist[v] then
15:                dist[v] ← dist[u] + w
16:                paths[v] ← paths[u] + [v]
17:        EndFor
18:        if dist == dist_copy then
19:            converged_in ← i
20:            break
21:    EndFor
22:    retourner (dist, converged_in, paths)
23:    if utils.has_negative_cycle(new_edges, dist) then
24:        retourner None
25:    retourner (dist, converged_in, paths)
```

au moins la moitié des nœuds de H via des chemins dirigés. Nous sélectionnons ensuite un nœud $s \in S$ de manière uniforme au hasard

Nous utiliserons des métriques pour l'évaluation comme le nombre d'itérations effectuées par Bellman-Ford selon les différentes ordres de considération des sommets. Nous rapporterons l'amélioration relative du nombre d'itérations avant convergence. Formellement, soit $BF(\pi)$ le nombre de fois que l'algorithme Bellman-Ford parcourt la liste d'adjacence de H avant de converger lors de l'utilisation de l'ordre π. L'amélioration relative d'un ordre π_0 par rapport à une référence π va être de $100 \times \frac{BF(\pi)-BF(\pi_0)}{BF(\pi)}$.

3.2.1 Géneration du graph initial aléatoirement

Nous allons donc générer un graphe initial G avec un nombre de sommet et d'arcs données

Algorithm 2 generate_random_graph

```
1: function GENERATE_RANDOM_GRAPH(nb_vertices : entier, nb_edges : entier, bounds : entier)
2:     vertices ← liste de 1 à nb_vertices
3:     aretes ← liste vide
4:     random_graph ← Graph(vertices)
5:     for i dans la plage de 0 à nb_edges do
6:         while True do
7:             u ← choisir aléatoirement un entier entre 0 et nb_vertices
8:             v ← choisir aléatoirement un entier entre 0 et nb_vertices
9:             if (u, v, _) n'est pas dans random_graph.edges et u est différent de v then
10:                Sortir de la boucle
11:        endWhile
12:        random_graph.add_edge(u, v, 1)
13:    EndFor
14:    Retourner Graph.generate_weighted_graph(random_graph, bounds)
```

à chaque génération d'un graphe aléatoire il était possible d'avoir des cycles négatif, nous avons donc traiter ce cas en remplaçant les valeurs négatives des poids concernés causant le cycle négatif (Le code ce trouvant dans les fichiers python fourni)

3.2.2 Union arborescence des graphes

L'objectif des graphes d'entraînement est d'utiliser l'union de leurs arbres de plus courts chemins pour expérimenter avec GloutonFas, un algorithme utilisé pour extraire un ordre spécifique à utiliser pour l'algorithme de Bellman-Ford dans l'idée d'améliorer ces performances. L'idée est d'explorer différentes combinaisons d'arbres de plus courts chemins provenant de ces graphes d'entraînement afin de trouver un ordre qui améliore les performances de l'algorithme de Bellman-Ford.

L'union des arbres de plus courts chemins des graphes d'entraînement permet de créer un nouveau graphe qui représente une combinaison des chemins optimaux de ces graphes. Cette approche vise à exploiter les informations contenues dans les différents graphes d'entraînement pour obtenir

Algorithm 3 generate_weighted_graph

```
1: function GENERATE_WEIGHTED_GRAPH(self, bound)
2:     Variables : have_neg_cycle ← VRAI
3:     while have_neg_cycle do
4:         new_graph ← Graph(self.vertices)
5:         for u, v, w dans self.edges do
6:             weight ← random.randint(−bound, bound)
7:             new_graph.add_edge(u, v, weight)
8:         EndFor
9:         source ← utils.get_best_src_node(self)
10:        dist, _, paths ← utils.relaxation(new_graph, source)
11:        utils.correct_negative_cycles(new_graph, dist, paths)
12:        dist, _, paths ← utils.relaxation(new_graph, source)
13:        have_neg_cycle ← utils.has_negative_cycle(new_graph.edges, dist)
14:    endWhile
15:    Retourner new_graph
```

un ordre qui peut être utilisé de manière efficace dans d'autres graphes de même taille.

En résumé les étapes à suivre sont :

- Créer les graphes d'entraînement et de tests (dans notre cas G1, G2, G3, H)
- Créer le graphe d'unification des arbres de plus court chemins des graphes d'entraînement
- appliquer l'algorithme glouton et obtenir l'ordre spécifique comme heuristique pour l'algorithme BF
- appliquer l'algorithme BF avec l'ordre et comparé les resultats

3.3 Comparaison des résultats obtenu pour l'algorithme BF avec ordre

après préparer les pré-traitement pour l'algorithme BF nous comparerons donc un ordre tiré aléatoirement (il a été prouvé que le nombre d'itérations attendu dans le pire des cas est $|V|/2$) avec un ordre tiré par l'algorithme gloutonfas nous obtiendront les résultats présents dans la figure ci-dessous

On constate donc une nette amélioration des itérations effectué avec l'ordre obtenu grâce à l'algorithme gloutonFas en convergeant avec une seul itération, alors qu'avec un ordre tiré aléatoirement l'algorithme converge en 3 itérations.

Sur la figure ci dessous représente le nombre d'itération evaluer par l'algorithme BF avec ordre gloutonfas en rouge et ordre tiré aléatoire en bleu, on peut remarquer qu'il y'a nettement plus d'itérations avec ordre tiré aléatoire, et cela est claire car parfois l'ordre tiré aléatoire peut donnerr de bon résultat mais en majorité du temps ce n'est pas le cas, alors que BF avec l'ordre traité donne trés souvent de bons résultat avec des itérations moindre

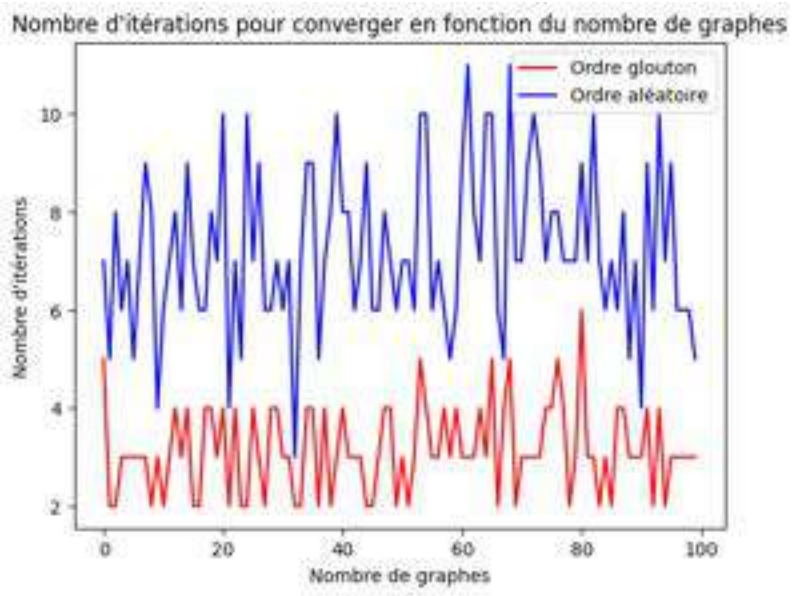

FIGURE 1 – Nombre d'itérations pour converger en fonction du nombre de graphes

4 Influence du nombre de Graphes d'Entraînements sur le nombre d'Itérations.

Nous avons d'abord généré 20 instances de graphes aléatoires afin de tester l'amélioration de la méthode avec l'ordre glouton par rapport à un ordre aléatoire. Les résultats obtenus sont les suivants :

- Nombre d'itérations pour converger avec l'ordre glouton : 3
- Nombre d'itérations pour converger avec l'ordre aléatoire : 4
- Amélioration de la méthode avec l'ordre glouton par rapport à un ordre aléatoire : 25.0 pourcent

Les résultats indiquent que l'algorithme a convergé en 3 itérations en utilisant l'ordre glouton, tandis qu'il a fallu 4 itérations avec un ordre aléatoire pour atteindre la convergence. Cela démontre que l'approche gloutonne permet d'obtenir une convergence plus rapide par rapport à l'approche aléatoire. L'amélioration de la méthode avec l'ordre glouton par rapport à l'ordre aléatoire est de 25.0 pourcent, ce qui correspond à la réduction du nombre d'itérations nécessaires pour atteindre la convergence. Ces résultats mettent en évidence l'efficacité de l'approche gloutonne dans la résolution du problème considéré, en réduisant le nombre d'itérations nécessaires pour obtenir la solution souhaitée.

Nous avons réalisé ensuite une série de tests pour évaluer les performances de l'ordre obtenu à partir de l'algorithme GloutonFas en fonction du nombre de graphes d'entraînement. Nous avons calculé le nombre d'itérations obtenu pour différentes valeurs du nombre de graphes d'entraînement.

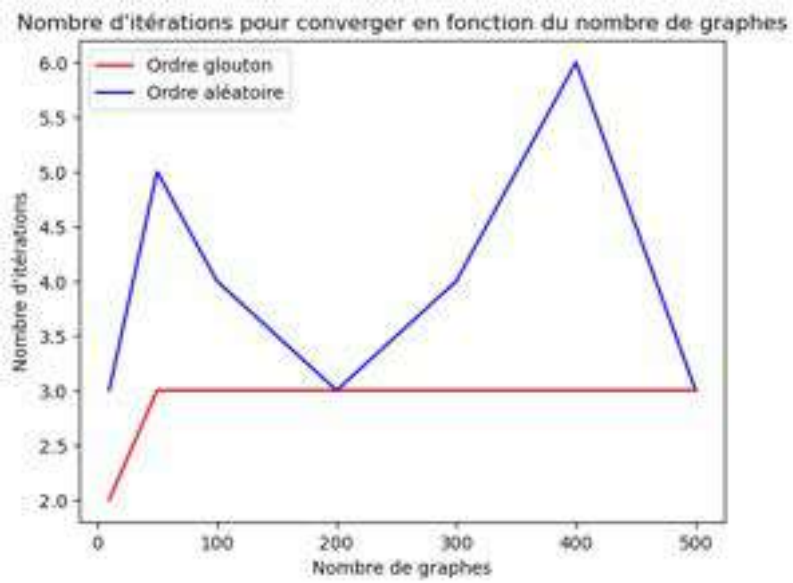

FIGURE 2 – Nombre d'itérations pour converger en fonction du nombre de graphes

Les nombres de graphes que nous avons testés sont les suivants : 10, 50, 100, 200, 300, 400 et 500.

Les résultats obtenus sont représentés dans le graphe suivant :

D'après l'analyse du graphe du nombre d'itérations pour converger en fonction du nombre de graphes, nous observons des comportements différents pour les stratégies d'ordre aléatoire et glouton.

Pour l'ordre aléatoire, au début, avec environ 0 à 50 graphes, le nombre d'itérations augmente. Cela peut être dû à l'incertitude et à la variabilité introduites par l'ordre aléatoire. Ensuite, le nombre d'itérations diminue à environ 200, puis augmente à nouveau jusqu'à atteindre environ 400. En fin de compte, le nombre d'itérations semble fluctuer, indiquant une instabilité due à la nature aléatoire de la stratégie.

En revanche, pour l'ordre glouton, le nombre d'itérations augmente initialement avec l'augmentation du nombre de graphes. Cependant, une fois que le nombre de graphes augmente davantage, le nombre d'itérations se stabilise. Cela suggère que la stratégie gloutonne est plus efficace pour résoudre les problèmes avec un grand nombre de graphes.

Une conclusion importante est que le nombre d'itérations de la stratégie gloutonne est toujours inférieur à celui de la stratégie aléatoire. Cela indique que l'ordre glouton permet d'atteindre plus rapidement la convergence que l'ordre aléatoire, ce qui peut être bénéfique en termes de temps de calcul et d'efficacité.

En résumé, le graphe montre que l'ordre glouton a tendance à nécessiter moins d'itérations pour converger par rapport à l'ordre aléatoire. Cependant, l'ordre aléatoire peut présenter une certaine

instabilité et des fluctuations dans le nombre d'itérations en raison de sa nature aléatoire. Ces résultats soulignent l'importance de choisir la bonne stratégie d'ordre en fonction des caractéristiques du problème à résoudre.

5 Étude pour le cas des graphes par niveau

Nous avons créé un graphe par niveau comprenant 4 sommets par niveau et 2500 niveaux. Dans ce graphe, les sommets du niveau j sont tous reliés aux sommets du niveau j + 1, et les poids des arcs sont choisis de manière aléatoire et uniforme parmi les entiers de l'intervalle [-10, 10].

Les résultats obtenus ont montré que l'amélioration des performances de l'algorithme glouton était d'environ 0 pourcent pour chaque nombre de graphes testé. Cela indique que l'approche gloutonne n'a pas apporté d'avantages significatifs dans la résolution de ce problème spécifique.

L'approche gloutonne est généralement considérée comme efficace dans de nombreux problèmes, mais dans le cas spécifique du graphe généré avec des niveaux, cette approche ne semble pas être aussi performante. Les résultats obtenus lors de nos tests ont révélé qu'il n'y avait pas d'amélioration significative des performances de l'algorithme glouton pour ce type de graphe.

Le graphe généré avec des niveaux, où les sommets d'un niveau précèdent tous les sommets du niveau suivant, présente une structure particulière qui peut rendre l'approche gloutonne moins efficace.

En conclusion, bien que l'approche gloutonne soit généralement efficace, elle ne semble pas être adaptée à la résolution du graphe généré avec des niveaux. D'autres approches ou méthodes pourraient être explorées pour améliorer les performances dans ce contexte spécifique.

6 Conclusion

Dans cette étude, nous avons examiné l'algorithme de Bellman-Ford et son comportement en fonction de l'ordre des sommets dans le graphe. Notre objectif était d'affiner l'estimation du nombre d'itérations nécessaires pour que l'algorithme converge.

Nous avons constaté que l'ordre des sommets a un impact significatif sur les performances de l'algorithme. En modifiant l'ordre, nous avons pu observer des variations dans la propagation de l'information sur les chemins les plus courts, ce qui a influencé le nombre d'itérations nécessaires pour atteindre la convergence.

De plus, nous avons utilisé des graphes d'entraînement pour créer un graphe d'unification des arbres de plus courts chemins. Cette approche nous a permis d'exploiter les informations contenues dans les graphes d'entraînement pour obtenir un ordre plus efficace dans d'autres graphes de même taille.

Nos expérimentations ont montré que l'utilisation d'un bon ordre des sommets peut considérablement améliorer les performances de l'algorithme de Bellman-Ford. Cela peut être particulièrement utile dans des contextes où la convergence rapide est cruciale.

En conclusion, cette étude nous a permis de mieux comprendre les facteurs qui influencent l'efficacité de l'algorithme de Bellman-Ford. Elle ouvre également des perspectives intéressantes pour l'amélioration des performances de cet algorithme dans différents contextes.