Définition de la question à étudier

Le but de cette investigation est de trouver des algorithmes pratiques et efficaces pour résoudre le problème des plus courts chemins dans des graphes orientés, pouvant possédées des arcs avec des poids négatifs. Je dois comparer différents algorithmes, les décrire brièvement, voir leurs points forts et leurs points faible. On évaluera aussi leur efficacité et leur robustesse pour choisir le meilleur. Finalement, je devrais donner mes recommandations du meilleur algorithme pour notre cas analyser.

Démarche d'investigation

Pour déterminer l'algorithme le plus adapté au routage dans Internet, nous avons analysé trois algorithmes majeurs de recherche de plus courts chemins Floyd, Bellman-Ford et Johnson. Le choix final a été guidé par plusieurs critères, dont la compatibilité avec les arcs de poids négatif, la capacité à gérer de grands graphes, la complexité algorithmique, et les performances dans des contextes de routage.

Floyd est particulièrement adapté aux petits graphes denses en raison de son approche matricielle qui permet de calculer les plus courts chemins entre tous les couples de sommets. Cependant, son coût élevé en mémoire et en temps ($O(n^3)$) le rend inadapté aux graphes de grande taille, comme ceux rencontrés dans les réseaux internet.

Bellman-Ford a été retenu pour son aptitude à traiter les arcs de poids négatif et à détecter les cycles négatifs. Bien qu'il soit plus lent ($O(nm)$) que d'autres algorithmes sur des graphes avec uniquement des poids positifs, sa robustesse en fait un candidat pertinent pour vérifier l'intégrité des données de routage.

Enfin, l'algorithme de Johnson combine la vérification des cycles négatifs grâce à Bellman-Ford et l'optimisation des calculs pour les graphes grâce à Dijkstra. Cette combinaison donne alors un temps $O(n^2\log(n) + nm)$ il est particulièrement efficace pour les graphes peu denses et de grande taille, ce qui correspond bien aux besoins d'un réseau étendu comme les réseaux internet.

Application de la démarche d'investigation

Floyd

Fonctionnement : Cet algorithme initialise une matrice de distances où chaque cellule [i,j contient le poids de l'arc reliant le sommet i au sommet j (ou l'infini si aucun arc n'existe). À chaque itération, il met à jour les distances minimales en vérifiant si un chemin passant par un sommet intermédiaire k est plus court que celui déjà trouvé.

Analyse : Il y a complexité temporelle en $O(n^3)$ et spatiale du à la matrice, ce qui limite son utilisation aux graphes petits et denses.

Avantages : La simplicité conceptuelle et capacité à calculer tous les plus courts chemins en une seule exécution.

Inconvénients : La complexité temporelle en $O(n^3)$ ainsi que la capacité spatiale de la matrice, limite son utilisation aux graphes petits et denses.

Cas d'utilisation : L'analyse de petits réseaux locaux où tous les chemins sont à évaluer simultanément.

Bellman-Ford

Fonctionnement : Il calcule les plus courts chemins depuis une source unique en relâchant les poids des arcs n−1 fois, où n est le nombre de sommets. Une dernière itération permet de détecter la présence de cycles négatifs.

Analyse : La complexité temporelle est O(nm), ce qui est efficace pour des graphes clairsemés.

Avantages : Il gère les poids négatifs et détecte les cycles négatifs.

Inconvénients : Il est plus lent que Dijkstra pour des graphes sans poids négatifs.

Cas d'utilisation : Réseaux où des poids négatifs peuvent apparaître, par exemple pour modéliser des coûts ou des profits.

Johnson

Fonctionnement : Il combine Bellman-Ford pour repondérer les arcs en supprimant les poids négatifs, suivi de Dijkstra pour calculer les plus courts chemins à partir de chaque sommet. La repondération garantit que Dijkstra fonctionne correctement.

Analyse : Il y a complexité temporelle $O(n^2 \log(n) + nm)$, ce qui le rend adaptée aux graphes peu denses et de grande taille.

Avantages : Il est évolutif pour les grands graphes et il est efficace en mémoire.

Inconvénients : Il dépend de deux algorithmes, ce qui augmente sa complexité d'implémentation.

Cas d'utilisation : Les réseaux peu denses mais de grande taille, comme Internet, où une gestion rapide des chemins entre tous les sommets est essentielle.

Analyse des données recueillies

Les performances des algorithmes ont été évaluées sur plusieurs types de graphes simulant différents contextes de routage. Floyd s'est montré efficace sur des graphes de petite taille, mais sa consommation mémoire a explosé sur des graphes plus grands. Bellman-Ford a bien détecté les cycles négatifs, mais son temps d'exécution était significatif pour de grands graphes. Johnson a montré la meilleure performance globale, particulièrement pour des graphes peu denses, grâce à son hybridation des deux approches.

Conclusion

En conclusion, l'algorithme de Bellman-Ford est le plus approprié pour les réseaux étudiés dans le cadre de ce TP. Ces réseaux, de petite taille et relativement simples, nécessitent une solution capable de gérer la présence possible d'arcs de poids négatif, ce que Bellman-Ford accomplit efficacement. Cependant, si le réseau devait évoluer pour devenir plus vaste et complexe, l'algorithme de Johnson serait une meilleure option. Grâce à la combinaison de la robustesse de Bellman-Ford et de l'efficacité de Dijkstra, Johnson offre une solution rapide et performante, particulièrement adaptée aux grands graphes incluant des arcs de poids négatif.

References

Algorithme de Floyd

Thierry Eude, Kim Rioux-Paradis, ALGORITHMES ET STRUCTURES DE DONNÉES IFT-2008/GLO-2100, Chapitre 5 : Graphe (partie 2), automne 2024

Algorithme de Bellman-Ford
Thierry Eude, Kim Rioux-Paradis, ALGORITHMES ET STRUCTURES DE DONNÉES IFT-2008/GLO-2100, Chapitre 5 : Graphe (partie 2), automne 2024

Algorithme de Johnson

Wikipédia, Algorithme de Johnson, Page visité le 11/19/2024, dernière modification le 2 avril 2024 à 18:03, https://fr.wikipedia.org/wiki/Algorithme_de_Johnson