CSI2510 - Structures des Données et Algorithmes

Projet de programmation
Partie 2

Chloé Al-Frenn
300211508

Université d'Ottawa
Le 3 novembre 2022

1. Expérience 1: Validation

Suite à l'implémentation de l'expérience 1 j'ai réussi à obtenir douze fichiers de texte, deux pour chacun des six points, le premier utilisant la méthode des arbres k-d et l'autre utilisant la méthode linéaire. De plus, à chaque fois qu'une méthode est appelée celle-ci va imprimer tous les points voisins et le nombre de points voisins sur la console.

Afin de vérifier si mes deux classes fonctionnent correctement je dois m'assurer que les deux méthodes produisent la même liste de voisins sans qu'ils ne soient nécessairement dans le même ordre. Pour ce faire, j'ai vérifié que le nombre de voisins trouvés par les deux méthodes était le même. En effet pour le premier point j'obtiens 5 voisins pour les deux méthodes et les autres points de 2 à 6 donnent respectivement 2, 1, 17, 2 et 2 pour chacune des deux méthodes. Puisque le nombre de voisin reste constant peu importe la méthode je peux déterminer que les deux méthodes donnent les même points.

2. Expérience 2: Temps d'exécution

2.1 Point Cloud 1

Pour le premier fichier Point_Cloud_1.csv j'ai fait un test en utilisant un epsilon de 0.5 et un paramètre de saut de 10. Pour le test avec la méthode kd j'ai obtenu un temps moyen de 0.03219473371582851 millisecondes En revanche, pour le test avec la méthode linéaire avec les mêmes paramètres j'ai obtenu un temps de 0.1321156037799526 millisecondes. Dans ce cas, la méthode kd est environ 75% plus rapide.

2.2 Point Cloud 2

Pour le deuxième fichier Point_Cloud_2.csv j'ai fait un test en utilisant un epsilon de 0.5 et un paramètre de saut de 10. Pour le test avec la méthode kd j'ai obtenu un temps moyen de 0.03937174970484057 millisecondes En revanche, pour le test avec la méthode linéaire avec les mêmes paramètres j'ai obtenu un temps de 0.17684208048012606 millisecondes. Dans ce cas, la méthode kd est environ 77% plus rapide.

2.3 Point Cloud 3

Finalement, pour le troisième fichier Point_Cloud_3.csv j'ai continuer à faire un test en utilisant un epsilon de 0.5 et un paramètre de saut de 10. Pour le test avec la méthode kd j'ai obtenu un temps moyen de 0.029097129596809927 millisecondes En revanche, pour le test avec la méthode linéaire avec les mêmes paramètres j'ai obtenu un temps de 0.17921512295081957 millisecondes. Dans ce cas, la méthode kd est environ 83% plus rapide.

2.4 Conclusion

En conclusion, lorsque je compare les résultats du temps pris par les deux méthodes je peux observer que pour les trois fichiers de points utilisés la méthode kd est au moins 75% plus rapide que la méthode linéaire. Je m'attendais à ce que celle-ci soit plus efficace d'environ 40% à 50%. Alors le fait qu'elle soit plus efficace de plus de 70% est assez surprenant.

3. Expérience 3: Intégration à DBScan

3.1 Point Cloud 1

J'ai fait 5 tests pour chacune des classes avec les paramètres de 0.5 pour epsilon et de 15 pour le minimum de point puisque c'est ce qui avait fonctionné pour moi pour le premier projet. Pour l'algorithme DBScan utilisant NearestNeighbors j'ai obtenu des temps de 5484, 5516, 3878, 5515 et 3616 millisecondes. Ce qui donne un temps moyen de 4801,18 millisecondes . Pour l'algorithme DBScan utilisant NearestNeighborsKD j'ai obtenu des temps de 1205, 1177, 1234, 1167 et 1135 millisecondes. Ce qui donne un temps moyen de 1183,6 millisecondes. Dans ce cas, l' utilisation de NearestNeighborsKD rend le programme environ 75% plus rapide.

3.2 Point Cloud 2

J’ai fait 5 tests pour chacune des classes avec les paramètres de 1.0 pour epsilon et de 10 pour le minimum de point puisque c'est ce qui avait fonctionné pour moi pour le premier projet. Pour l’algorithme DBScan utilisant NearestNeighbors j’ai obtenu des temps de 11740, 11554, 11827, 18217 et 12011 millisecondes. Ce qui donne un temps moyen de 13069,8 millisecondes . Pour l’algorithme DBScan utilisant NearestNeighborsKD j’ai obtenu des temps de 5921, 5816, 6147, 6007 et 5902 millisecondes. Ce qui donne un temps moyen de 5958,6 millisecondes. Dans ce cas, l’ utilisation de NearestNeighborsKD rend le programme environ 54% plus rapide.

3.3 Point Cloud 3

J’ai fait 5 tests pour chacune des classes avec les paramètres de 0.5 pour epsilon et de 15 pour le minimum de point puisque c'est ce qui avait fonctionné pour moi pour le premier projet. Pour l’algorithme DBScan utilisant NearestNeighbors j’ai obtenu des temps de 7224, 7337, 12607, 12877 et 12806 millisecondes. Ce qui donne un temps moyen de 10570,2 millisecondes. Pour l’algorithme DBScan utilisant NearestNeighborsKD j’ai obtenu des temps de 1480, 1480, 1422, 1458 et 1430 millisecondes. Ce qui donne un temps moyen de 1454. Dans ce cas, l’ utilisation de NearestNeighborsKD rend le programme environ 86% plus rapide.

3.4 Conclusion

Suite aux résultats de l'expérience 2 je m’attendais à ce que l’utilisation de NearestNeighborsKD rende l'exécution du programme beaucoup plus rapide. J’ai pu observer que c'était effectivement le cas puisque c’est au minimum 2 fois plus rapide que l'exécution de DBScan avec la classe NearestNeighbors.