# Analyse des Implémentations de l'Algorithme DBScan

Jordan Lau

300240600

École de science d'informatique et de génie électrique, Université d'Ottawa

CSI 2510: Structures de Données et Algorithmes

Professeur Robert Laganière

5 décembre 2022

## Introduction

Une partie clé du logiciel d'un véhicule autonome, c'est l'algorithme DBScan (Density-Based Scanning). Cet algorithme analyse les nuages de point détectés par les senseurs LIDAR et forme les groupes des points qui représente les objets autour d'un véhicule autonome (Laganière et al., 2022). Premièrement, une méthode qui s'appelle *rangeQuery* détermine le nombre d'autres points dans le voisinage d'un point donné. Si ce voisinage est assez grand, l'algorithme étiquette ces points-là comme un groupe. Autrement, ils sont étiquetés comme du bruit. Ce petit rapport présentera trois expériences qui analysent les différences entre deux implémentations du méthode *rangeQuery*, une partie de l'algorithme DBScan. La première implémentation de *rangeQuery* processus les points linéairement. Celui-ci a un temps d'exécution au pire case de O(n), où n est égal au nombre total des points (Laganière et al., 2022). La deuxième implémentation de *rangeQuery* processus les points avec un arbre binaire k-d. Cet arbre est trois-dimensionnel.

## Expérience 1 - Validation

Dans la première expérience, l'implémentation k-d était comparée avec l'implémentation linéaire afin de confirmer qu'ils produisent les mêmes résultats. Le méthode *rangeQuery* a produit le même ensemble des points voisins pour cinq points aléatoires. Ça confirme que les deux implémentations fonctionnent bien.

**Expérience 2 - Temps d'exécution**

Dans la deuxième expérience, le temps d'exécution de chaque implémentation a été testé. Le temps a été pris avant et après l'appel du *rangeQuery* sur chaque 10ème point dans trois fichiers *Point Cloud*.

Tableau 1. Le temps moyen, en nanosecondes, dans l'expérience 2.

| Implementation | Point Cloud 1 | Point Cloud 2 | Point Cloud 3 |
|---|---|---|---|
| lineare | 146230 | 204645 | 229553 |
| arbre k-d | 19800 | 28160 | 23179 |

Comme prévu, l'implémentation arbre k-d était significativement plus rapide. Ça, c'est parce que la structure d'un arbre binaire k-d peut éliminer les sous-arbres grands avec juste une comparaison pendant la recherche. Même si cet arbre-ci n'est pas un arbre AVL, il a une complexité temporelle au moyen cas de O(logn), tellement plus rapide que l'implémentation linéaire, qui a une complexité temporelle de O(n) au pire.

**Expérience 3 - Intégration à DBScan**

Dans la dernière expérience, le temps d'exécution de l'algorithme DBScan était testé avec les deux implémentations de *rangeQuery*. Le temps a été pris avant et après la lecture de trois fichiers *Point Cloud* et la recherche des groupes.

Tableau 2. Le temps moyen, en nanosecondes, dans l'expérience 2.

| implémentation | Point Cloud 1 | Point Cloud 2 | Point Cloud 3 |
|---|---|---|---|
| lineare | 3933374400 | 12089901700 | 8508229000 |
| arbre k-d tree | 1170327100 | 3084733699 | 1490790401 |

Dans cette expérience, l'implémentation k-d était plus rapide, mais pas avec telle différence. Ca, c’est parce que, malgré les avantages des arbres k-d pour la recherche, l’arbre k-d ne peut pas être construit de façon efficace. Il faut plusieurs comparaisons afin d'insérer un point dans la bonne feuille.

# Références

Laganière, R., Lee, W., & Moura, L. (2022). Détection d'objets avec l'algorithme DBScan. *CSI 2510: Structures de Données et Algorithmes.* Université d'Ottawa.