Département Sciences du Numérique

Science des réseaux - Algorithmes de clustering

Capucine DAVID, Ambre LIABAT, Mathis RIGAUD

Année 2024 - 2025
Semestre 8

Table des matières

Table des figures

1 Introduction

L'objectif de ce projet est de chercher à intégrer des utilisateurs dans des clusters afin qu'ils soient desservis par les faisceaux de satellites en orbite basse. Chaque satellite peut générer un cluster de 90 km de diamètre et fournir un débit maximal de 1, 2 ou 4 Gbps. Notre but est de minimiser le nombre de clusters nécessaires pour desservir chaque utilisateur d'une base de données fournie. Les utilisateurs sont répartis autour du globe et ont des besoins en termes de débits différents.

Nous allons tester deux algorithmes différents pour créer ces clusters, puis analyser leurs résultats. Nous remettrons enfin en question l'objectif du projet, et nous attarderons sur les autres pistes que nous aurions pu explorer.

2 Algorithmes de clustering

Nous utilisons deux algorithmes différents : un algorithme basé sur les Ball Trees et l'algorithme DBSCAN (Density-Based Spatial Clustering for Applications with Noise).

2.1 Algorithme Ball-Tree

L'algorithme Ball Tree sert à diviser des données en boules imbriquées les unes dans les autres. On peut donc définir un arbre à boules comme un arbre binaire dont chaque noeud définit une boule de dimension donnée et contenant ainsi un sous-ensemble de points. Notons trois caractéristiques :

Les nœuds internes divisent les points en deux groupes disjoints, chacun assigné à une bille, selon leur distance au centre des billes.

De plus, bien que les billes puissent se chevaucher, chaque point appartient à une seule bille lors du partitionnement.

Et les feuilles de l'arbre correspondent à des billes finales contenant explicitement les points qu'elles englobent.

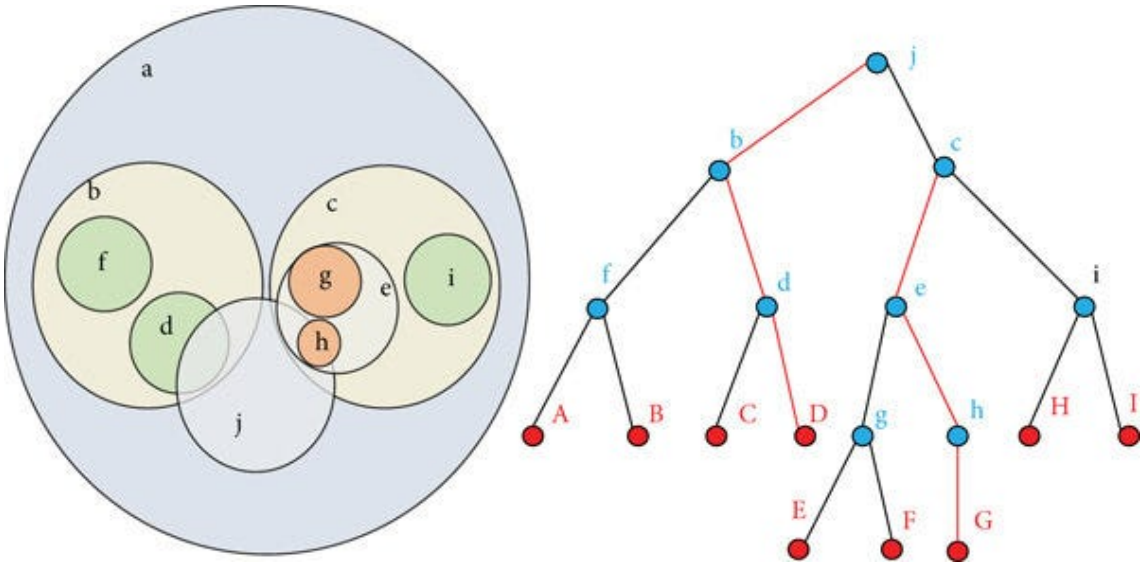


FIGURE 1 – Algorithme de Ball Tree

2.2 Algorithme DBSCAN

DBSCAN, density-based spatial clustering of application with noise, est un algorithme fondé sur la densité pour partitionner les données. Pour fonctionner, l'algorithme utilise deux paramètres ϵ qui définissent le voisinnage d'un point à étudier et $MinPoints$, le nombre minimal de points qui doivent être dans le cluster.

Cet algorithme permet de différencier trois types de points :

— core points
— border points
— noise points

FIGURE 2 – Algorithme de DBSCAN

3 Implémentation

3.1 Fonctions et scritps

Pour pouvoir lancer nos simulations, il faut exécuter le script **run_simulations.py**, puis rentrer dans la ligne de commande l'algorithme que nous souhaitons étudier : *ball_tree* ou *dbscan*. Afin de fluidifier les simulations, plusieurs fonctions on été créées :

- **color_csv** dans colorateur.py
- **calcul_nb_clusters_ball_tree** dans calculs_clusters_stats.py
- **nb_users_par_cluster_ball_tree** dans calculs_clusters_stats.py

Notons que les deux dernières ne servent que pour l'algorithme de ball tree. En effet, concernant DBSCAN, le nombre de clusters et le nombre moyen d'utilisateurs par clusters sont déjà calculés dans la fonction **run_dbscan** dans dbscan.py.

3.2 Résultats

Une fois les simulations terminées, nous pouvons retrouver divers fichiers de résultats :

- dans le dossier **res**/<**nom_algorithme**>, 2 fichiers pour chaque débit testé
  - un contenant les clusters créés
  - un où l'on a rajouté une couleur aux clusters pour la visualisation des résultats
- dans le dossier **stats**, 2 fichiers contenant pour chacun des algorithmes testés un CSV avec débit, nombre de clusters créés et nombre moyen d'utilisateurs par clusters

4 Résultats des algorithmes

Nous utilisons kepler.gl pour observer les données du tableau CSV sur une carte du monde. Chaque cluster possède une couleur, dans la mesure du possible différente de celles des clusters voisins. Cela permet de différentier les clusters sur la carte. On importe simplement les données CSV sur le site, puis on peut observer les résultats après quelques ajustements sur la légende.

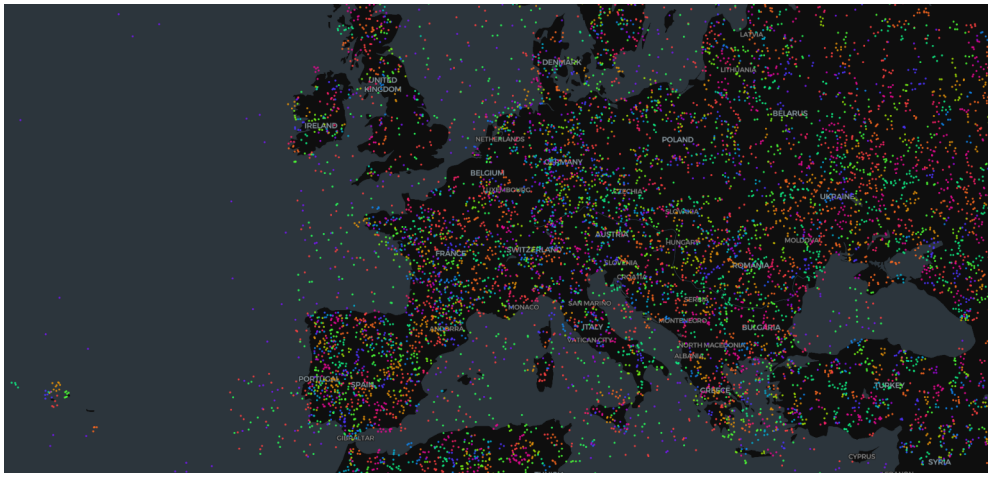
4.1 Algorithme Ball-Tree

L'algorithme de Ball-Tree semble avoir des résultats corrects. En effet, on observe bien logiquement une diminution du nombre de clusters créés quand le débit maximal par cluster augmente (ce qui est logique car on peut ajouter plus d'utilisateurs dans le cluster). On observe les résultats de l'algorithme dans le tableau suivant :

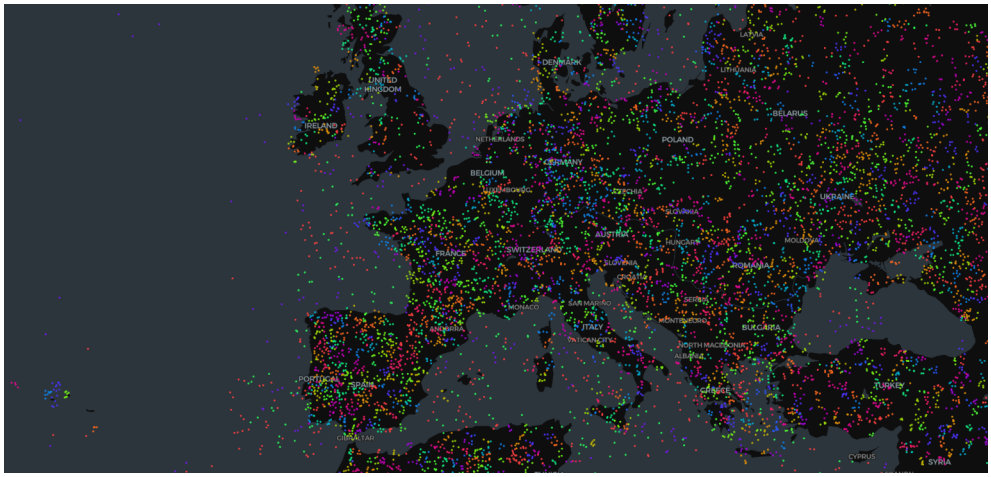
Débit	Nombre de clusters	Nombre moyen d'utilisateurs
1 Gbps	40408	2.18
2 Gbps	24482	3.59
4 Gbps	18788	4.68

TABLE 1 – Résultats de l'algorithme Ball Tree

Sur l'Europe, on observe grâce à kepler.gl la répartition suivante :



(a) 1 Gbps



(b) 2 Gbps

(c) 4 Gbps

FIGURE 3 – Observation des clusters créés par l'algorithme Ball-Tree (Europe)

On observe bien une "clusterisation" plus élevée lors de l'augmentation du débit maximal permis par chaque cluster : les clusters contiennent plus d'utilisateurs, et sont donc moins nombreux !

4.2 Algorithme DBSCAN

L'algorithme DBSCAN semble avoir des résultats plus mitigés que l'algorithme de Ball-Tree. En effet, on se rend compte que peu importe le débit, le nombre de clusters / nombre d'utilisateurs par cluster varie peu :

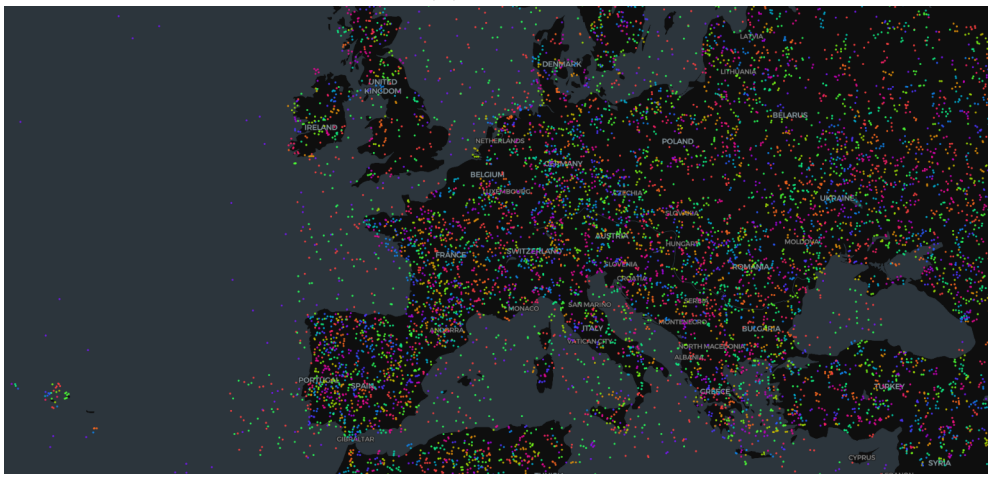
Débit	Nombre de clusters	Nombre moyen d'utilisateurs
1 Gbps	46979	1.87
2 Gbps	44802	1.96
4 Gbps	44652	41.97

TABLE 2 – Résultats de l'algorithme DBSCAN

On observe cette répartition sur la carte de l'Europe :

(a) 1 Gbps

(b) 2 Gbps



(c) 4 Gbps

FIGURE 4 – Observation des clusters créés par l'algorithme DBSCAN (Europe)

Contrairement à l'algorithme Ball-Tree, les clusters restent limités et peu étendus même lorsque le débit maximal possible par cluster augmente. Cela est dû à une faiblesse de l'algorithme DBSCAN, qui considère qu'un point "bruit" i.e. un point

solitaire ou particulier sera considéré comme un cluster à lui seul, plutôt que d'être intégré dans un cluster voisin disponible.

Par ailleurs, l'exécution de l'algorithme Ball-Tree est plus rapide que celle de l'algorithme DBSCAN.

5 Remise en question

Rapellons que l'objectif initial du projet était de réduire le nombre de clusters de satellites. Cependant, après de résultats tant mitigés, nous pouvons en venir à nous demander si cette décision de réduire le nombre de cluster était réellement la meilleure.

En effet, bien que le réduction du nombre de satellites soit nécessaire pour une utilisation plus efficace des bandes passantes, nous pouvons remarquer que des pertes de redondance, d'adaptabilité et de ressources sont inévitablement entrainées.

6 Autres pistes

6.1 Sweep Line

Nous avons durant ce projet également commencé à implémenter l'algorithme Sweep Line, cependant, faute de temps, nous n'avons pas pu aboutir à une version fonctionnelle.

Plus précisement, nous avons cherché à implémenter l'algorithme de Fortune. Cet algorithme repose sur une droite qui balaye l'espace de gauche à droite. À gauche de cette droite se forme une beach line, composée d'arcs de paraboles, chacun associé à un germe déjà rencontré.

Voici les grandes étapes de cet algorithme ;

— Lorsqu'un nouveau germe est atteint, un arc de parabole est ajouté à la berge.
— Les intersections entre arcs se trouvent sur les bissectrices des germes concernés.
— Lorsqu'un troisième germe est ajouté, s'il existe un cercle passant par les trois germes et tangent à la droite de balayage, le centre de ce cercle devient un nœud du diagramme de Voronoï (car il est équidistant des trois germes)
— Cela provoque la disparition d'un arc de la beach line

Ainsi, la beach line est modélisée par un arbre binaire de recherche, permettant des opérations en temps $O(nlog(n))$.

6.2 K-means

Enfin, nous avons également essayé d'implémenter l'algorithme des k-means. Cet algorithme est un algorithme de clustering qui cherche à minimiser la somme des distances intra-cluster, soit la somme suivante : $\sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$.

Cet algorithme fonctionne de la façon suivante : lors de l'initialisation, on choisit le nombre de clusters K ainsi que les centres parmi les points du fichier. Pour converger plus rapidement, on peut utiliser la méthode K-means++ afin de choisir les centres

initiaux. Ensuite, les points sont assignés aux clusters disponibles. On met ensuite à jour les centres on prenant les centres de gravité des clusters formés, puis on réitère ces deux dernières opérations jusqu'à convergence. On voit donc qu'il faut le nombre de clusters, soit exactement ce que l'on cherche dans notre projet, afin de faire tourner cet algorithme. Pour contourner ce problème, nous avons donc souhaité implémenter une boucle pour tester le clustering selon un nombre de clusters variant entre un minimum (calculé en fonction du débit max par cluster) et un maximum (le nombre de points du fichier). Nous avons ensuite observé que cette version de k-means, bien que prometteuse, était trop gourmande en calcul et ne pouvait donc pas tourner sur nos ordinateurs qui ne sont pas assez puissants.

Cette piste a donc été abandonnée.

Références

[1] A parallel algorithm for approximating the silhouette using a ball tree, *Ivan Šimeček and Ivan Šimeček*, sciencedirect.com, 4 novembre 2022.

[2] Fast and explainable clustering based on sorting, *Xinye Chen and Stefan Güttel*, sciencedirect.com, 31 janvier 2024.

[3] A density-based algorithm for discovering clusters in large spatial databases with noise, *Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu*, dl.acm.org, 2 août 1996.

[4] Efficient incremental density-based algorithm for clustering large datasets, *Ahmad M. Bakr, Naia M. Ghanem, Mohamed A. Ismail*, sciencedirect.com, 20 juin 2015.

[5] A sweep-line algorithm for spatial clustering, *Krista Rizman Žalik, Borut Žalik*, sciencedirect.com, 6 juin 2008.

[6] Sweep-Hyperplane Clustering Algorithm Using Dynamic Model, *Krista Rizman Žalik, Borut Žalik, Niko Lukac*, researchgate.net, décembre 2014.