Algorithmique et structures de données
Rapport de projet

Mohammed EL ARABI Faical EL GOUIJ

April 12, 2025

1 Introduction

Le projet s'aritcule autour de l'algorithmique, de la théorie des graphes et de l'optimisation. Il s'intéresse à proprement parler au calcul des tailles des *composantes connexes* d'un graphe $G = (V, E)$ construit à partir d'un nuage de points p_i de la manière suivante :

- Chaque sommet v_i dans V correspond à un point p_i dans le nuage de points.
- Chaque arête e_i dans E correspond à un couple de points (p_i, p_j) tel que $dist(p_i, p_j) \leq d_s$ où d_s est une distance seuil.

Dans ce rapport, nous présentons un algorithme qui prend en entrée un nuage de points et une distance seuil, construit le graphe $G = (V, E)$ comme préalablement décrit, et termine en affichant sur la sortie standard la taille de ses *composantes connexes*.
De plus, nous discutons sa performance et sa complexité en analysant les résultats empiriques issus de son exécution sur différents nuages de points et distances seuil.
De surcroit, nous en donnons plusieurs versions optimisées, qui ont réduit significativement le temps d'exécution, en particulier pour les nuages de points volumineux.
Enfin, nous comparons les performances des différents algorithmes conçus et concluons.

2 Version primitive de l'algorithme

La version naïve de notre algorithme consiste avant toute chose à construire le graphe $G = (V, E)$, puis à explorer, pour chaque sommet v dans V, la *composante connexe* à laquelle il appartient.
Une approche récursive étant impertinente, particulièrement pour les nuages de points volumineux, nous optons pour une démarche algorithmique itérative basée sur une pile.

2.1 Construction du graphe

Nous choisissons de matérialiser l'ensemble des sommets V par une *table de hachage*, dont les *clés* sont les sommets et les *valeurs* sont l'un des deux *tokens* suivants :

- **PAS ENCORE TRAITE** : pour marquer les points qui n'ont pas encore été identifiés comme faisant partie d'une *composante connexe*.
- **DANS COMPOSANTE CONNEXE** : pour marquer les points identifiés comme faisant partie d'une *composante connexe*.

Nous modélisons l'ensemble des arêtes E par une *table de hachage* également, dont les *clés* sont les sommets, et les *valeurs* sont les listes de leurs voisins.

Ainsi, à partir d'un nuage de points et d'une distance seuil, nous construisons le graphe $G = (V, E)$ de la manière suivante :

- Nous marquons tous les sommets avec le *token* : **PAS ENCORE TRAITE**.
- Pour chaque sommet v_i, nous calculons sa distance de tous les autres sommets v_j (avec $v_j \neq v_i$). Si cette distance est inférieure ou égale à la distance seuil, nous ajoutons v_j à la liste des voisins de v_i.

2.2 Exploration des composantes connexes

Nous présentons ici un algorithme intermédiaire explorant, pour un point donné p, la composante connexe de G à laquelle il appartient.

Algorithme d'exploration de composante connexe (Alg-ECC)
Entrée : un point p, un graphe $G = (V, E)$.

1. Initialiser la taille de la composante connexe à explorer à 0.
2. Créer une pile contenant p.
3. Marquer p dans V avec le token **DANS COMPOSANTE CONNEXE**.
4. Tant que la pile n'est pas vide faire :
   (a) Dépiler en stockant dans p'.
   (b) Incrémenter la taille de la composante connexe en cours d'exploration.
   (c) Pour chaque voisin v de p' marqué avec le token **PAS ENCORE TRAITE** faire :
      i. Marquer v avec le token **DANS COMPOSANTE CONNEXE**.
      ii. Empiler v dans la pile.
5. Renvoyer la taille de la composante connexe explorée.

2.3 Calcul des tailles des composantes connexes

Nous présentons dans cette sous-section notre algorithme principal.

Algorithme de parcours et marquage concomittants (Alg-PMC)
Entrée : un nuage de points et une distance seuil.

1. Construire le graphe $G = (V, E)$ selon la méthode décrite à la section 2.1.
2. Initialiser la liste des tailles des composantes connexes : $L = [\,]$
3. Pour chaque sommet v dans V faire :
   (a) Si v est marqué avec le token **DANS COMPOSANTE CONNEXE** : passer directement au sommet suivant. Sinon :
   (b) Calculer la taille de la composante connexe à laquelle appartient v selon **Alg-ECC**.
   (c) Ajouter cette taille à L.
4. Trier dans l'ordre décroissant puis afficher L.

2.4 Analyse des résultats d'exécution de Alg-PMC

Notre algorithme **Alg-PMC** termine en temps polynomial en affichant sur la sortie standard la liste des tailles des *composantes connexes* triée dans l'ordre décroissant.
Afin d'analyser sa performance et mettre en lumière ses limites, nous l'avons implémenté sous Python et l'avons exécuté sur plusieurs nuages de points et distances seuil.
Nous avons constaté que, même en variant la *distance seuil*, notre algorithme termine toujours pour les nuages de points de taille inférieure à 10000. Au-delà, il prend un temps d'exécution plutôt lent.
De plus, nous avons également remarqué que la *distance seuil* influe sur la taille des *composantes connexes* de sorte que, plus elle augmente, plus le graphe G a tendance à avoir des sommets réunis en *clusters*, engendrant une *composante connexe* de taille grande.

Nous présentons ci-dessous trois courbes de performance décrivant l'évolution du temps d'exécution $T(n)$ de notre algorithme en fonction de la taille des entrées n (ie le nombre de points) pour différentes *distances seuil* d_s.

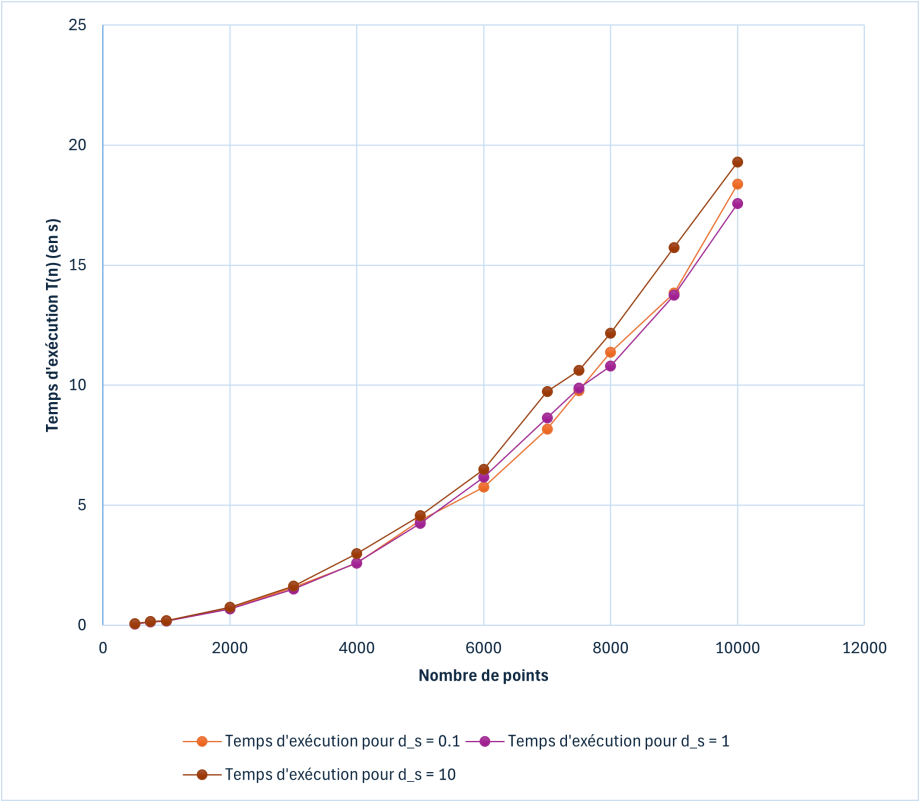

Figure 1. Évolution du temps d'exécution $T(n)$ de **Alg-PMC** en fonction du nombre de points n, pour différentes distances seuil.

L'analyse de la courbe ci-dessus suggère fortement que **Alg-PMC** présente une complexité quadratique en $O(n^2)$ où n correspond au nombre de points.
En effet, le temps d'exécution croît de façon non linéaire à mesure que le nombre de points augmente. De plus, le ratio $T(n)/n^2$ est quasi-constant comme l'illustre la figure ci-dessous.
Par ailleurs, les différentes courbes correspondant à des valeurs variées de la *distance seuil* conservent une évolution similaire, indiquant que la complexité est principalement liée à la taille des données plutôt qu'à ce paramètre.

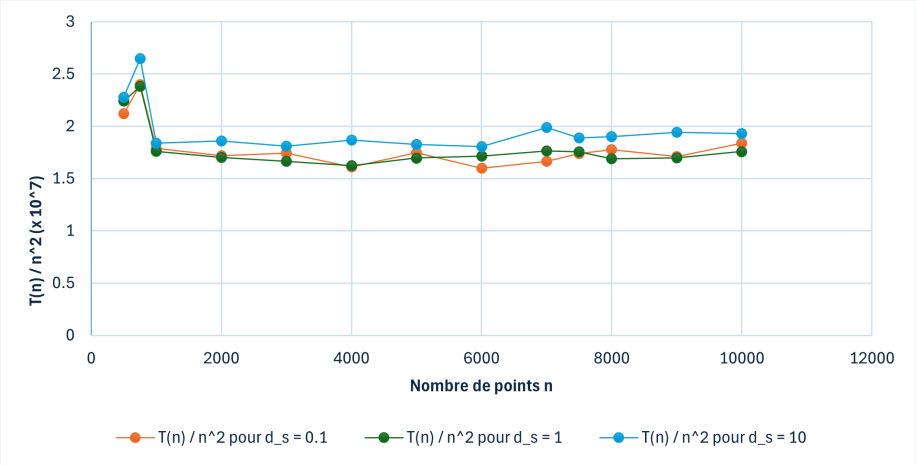

Figure 2. Ratio $T(n)/n^2$ pour différentes distances seuil

3 Versions optimisées de Alg-PMC

3.1 Optimisation de la construction du graphe

Dans l'objectif d'optimiser la construction du graphe $G = (V, E)$, étape la plus coûteuse de notre algorithme, nous subdivisons l'espace des points en une grille consistant en cellules carrées de côté égale à la distance seuil.
Nous faisons correspondre chaque point p_i à une cellule c_j de sorte que les points proches au regard de la distance seuil s'associent ou bien à la même cellule, ou bien à des cellules contiguës.
Ainsi, pour chaque point p_i associé à la cellule c_j, nous nous contentons de calculer sa distance des points correspondants à la même cellule et aux cellules avoisinantes uniquement, limitant ainsi le nombre de comparaisons.

3.2 Optimisation au niveau de l'implémentation

Les dernières optimisations portent sur l'implémentation Python de notre algorithme.
Afin de comparer les distances entre points par rapport à la *distance seuil*, nous choisissons de comparer les carrés des distances plutôt que d'utiliser la fonction racine carrée, celle-ci étant coûteuse en temps de calcul.
Par ailleurs, nous recourons à certaines méthodes Python permettant d'éviter des opérations potentiellement coûteuses, comme l'utilisation de la méthode *get()* des dictionnaires.
De plus, nous changeons la structure de données matérialisant les sommets en *set*.
Enfin, plutôt que de marquer les sommets avec des tokens, nous décidons de supprimer un sommet dès qu'il a été identifié comme appartenant à une *com-*

posante connexe.
Il est à noter que nous avons pensé à utiliser une file à double queue afin d'optimiser les accès aux sommets à traiter au cours de l'exploration d'une *composante connexe*, en utilisant les méthodes *popleft() et pop()*, mais nous nous sommes rendus compte que cela n'améliorait pas la performance de l'implémentation.

3.3 Analyse des résultats obtenus après optimisation

L'exécution de **Alg-PMC** ainsi optimisé a mis en évidence un comportement étroitement lié à la valeur de la *distance seuil*.
En effet, les trois courbes de la *figure* 3, matérialisant l'évolution du temps d'exécution $T(n)$ en fonction de n, montrent que la *distance seuil* conditionne la complexité de l'algorithme optimisé.

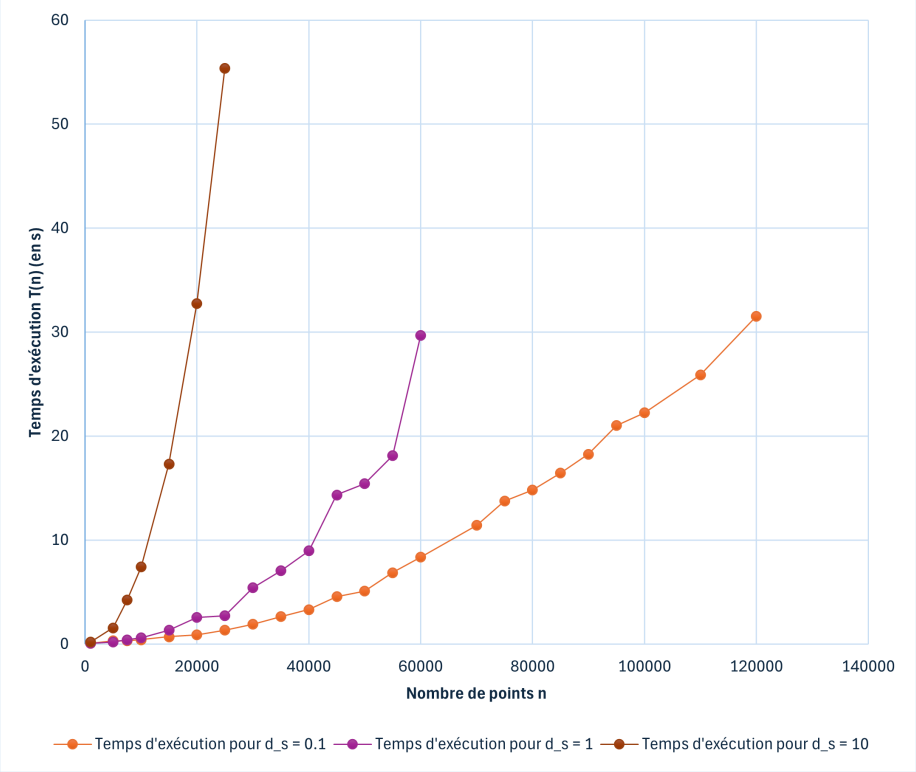

Figure 3. Résultats d'exécution de **Alg-PMC** optimisé sur différents nuages de points et différentes distances seuil.

Nous observons que pour une *distance seuil* suffisamment grande $d_s = 10$, le comportement de l'algorithme optimisé tend à ressembler fortement à sa version primitive, avec une complexité quadratique en $O(n^2)$.
Cela s'explique par le fait qu'une grande valeur de d_s connecte un nombre important de points, concentrés dans des cellules contiguës. Par conséquent, le

filtrage spatial devient inefficace et le nombre de comparaisons de distances explose, ce qui dégrade les performances.
À l'inverse, on remarque que l'efficacité de l'algorithme s'améliore nettement lorsque la *distance seuil* est réduite, en particulier sur des nuages de points volumineux.
En comparant les courbes correspondant à $d_s = 0.1$ (orange) et $d_s = 1$ (violet), on observe une croissance similaire jusqu'à environ 10 000 points. Au-delà, la courbe violette croît plus rapidement ; la capacité de discrimination spatiale de la grille devient moins efficace, car $d_s = 1$ augmente le nombre de points à comparer dans chaque cellule et ses voisines.
La courbe orange, en revanche, conserve une croissance plus lente et régulière, ce qui témoigne d'une complexité proche de $O(n\,log(n))$. La courbe violette tend, quant à elle, vers un comportement quasi-linéarithmique voire super-linéarithmique entre $O(n\,log(n))$ et $O(n\,log^{\alpha}(n))$ (avec $\alpha > 1$), reflétant un impact progressif de la *distance seuil* sur la performance de l'algorithme.

Remarque :
Le $O(n\,log(n))$ pour les petites distances seuil est en partie dû à la méthode *sort* de Python, qui domine le comportement linéaire de **Alg-PMC** optimisé.

4 Conclusion

En définitive, nous avons conçu un algorithme capable, à partir d'un nuage de points et d'une distance seuil donnée, de calculer et d'afficher la taille de l'ensemble des composantes connexes du graphe G.
De plus, nous avons considérablement amélioré ses performances en introduisant une structure de grille spatiale et en adoptant des choix d'implémentation efficaces, permettant de faire passer la complexité de l'algorithme de quadratique à linéarithmique.

L'analyse expérimentale nous a permis de dégager une stratégie d'utilisation optimale :

- Pour une distance seuil très faible, la version optimisée est à privilégier, en particulier pour des nuages de points volumineux.
- Pour une distance seuil importante, la version primitive peut suffire, à condition que le nombre de points reste modéré (typiquement inférieur à 10000).

Par ailleurs, ce projet nous a permis de prendre conscience des limites de certaines conceptions naïves, mais aussi d'explorer des pistes d'optimisation algorithmique. Nous avons consolidé nos connaissances en algorithmique, en théorie des graphes, ainsi qu'en programmation efficace sous Python, tout en développant une meilleure capacité d'analyse critique de la performance.