Sciences Sorbonne Université
Master STL - Année Universitaire 2024-2025

Petit Projet CPA

Calcul du Cercle Minimum Englobant

Nom et Prénom : AHMED Youra
N°etudiant : 21416343
Enseignant : Binh-Minh Bui-Xuan

9 mars 2025

Table des matières

Table des matières **1**

1 Introduction **2**

2 Présentation théorique des algorithmes **3**
2.1 Définition du problème 3
2.2 Algorithme Naïf 3
2.3 Algorithme de Welzl 3
2.4 Comparaison des algorithmes 4

3 Implémentation **4**

4 Expérimentation **5**
4.1 Base de test 5
4.2 Méthodologie de test 5
4.3 Conditions expérimentales 5

5 Analyse des résultats **5**
5.1 Comparaison des temps d'exécution 5
5.2 Comparaison des temps moyens d'exécution 6
5.2.1 Calcul des temps moyens 6
5.2.2 Facteur d'accélération moyen 7
5.2.3 Interprétation et conséquences 7
5.3 Analyse du gain de performance 8
5.4 Résultats visuels de l'interface graphique 8

6 Conclusion et perspectives **9**
6.1 Perspectives et améliorations possibles 10

Références **10**

1 Introduction

La géométrie algorithmique est un domaine fondamental en informatique qui trouve des applications dans des domaines aussi variés que la robotique, la vision par ordinateur et l'optimisation. L'un des problèmes classiques de ce domaine est la détermination du plus petit cercle couvrant un ensemble de points dans un plan, également connu sous le nom de *Minimum Enclosing Circle* (MEC) [1]. Ce problème consiste à trouver le plus petit cercle possible contenant un ensemble donné de points.

L'enjeu de ce problème réside dans l'optimisation du temps de calcul, en particulier lorsque l'ensemble de points est de grande taille. Deux approches sont souvent utilisées :

— Un algorithme naïf, basé sur une approche exhaustive qui teste toutes les combinaisons possibles pour trouver le cercle optimal. Mais bien que correct, cet algorithme devient rapidement inefficace lorsque le nombre de points augmente.

— L'algorithme de Welzl, une méthode probabiliste efficace qui permet de résoudre le problème en temps quasi-linéaire en moyenne, ce qui le rend adapté aux grands ensembles de données.

Dans ce projet, j'ai pour objectif d'implémenter ces deux algorithmes, de les comparer expérimentalement et d'analyser leurs performances sur une base de test de grande taille. Je m'intéresse aussi notamment à la précision des résultats et aux gains de temps obtenus par la méthode de Welzl par rapport à l'algorithme naïf.

Le rapport est structuré comme suit :
— La section 2 présente les deux algorithmes étudiés ainsi que leurs fondements théoriques.
— La section 3 décrit brièvement l'implémentation choisie ainsi que les structures de données utilisées.
— La section 4 détaille la méthodologie adoptée pour l'expérimentation, en précisant la provenance des données et les outils utilisés.
— La section 5 expose et analyse les résultats expérimentaux obtenus sous forme de graphiques et d'indicateurs pertinents.
— Enfin, la section 6 discute les résultats obtenus et propose des perspectives d'amélioration.

2 Présentation théorique des algorithmes

2.1 Définition du problème

Le *Minimum Enclosing Circle* (MEC) est le plus petit cercle englobant un ensemble fini de points dans le plan. Il est unique et peut être défini de manière géométrique comme le cercle ayant un rayon minimal tout en contenant tous les points de l'ensemble.

2.2 Algorithme Naïf

L'algorithme naïf consiste à tester toutes les paires et triplets de points possibles afin de trouver le plus petit cercle couvrant l'ensemble des points. Cet algorithme repose sur l'idée que :

— Un cercle couvrant minimal est soit défini par deux points sur son diamètre.

— Soit défini par trois points formant un triangle dont il est le cercle circonscrit.

Sa **complexité** est de $O(n^4)$, ce qui le rend inefficace pour de grandes bases de test.

Voici l'algorithme utilisé

Algorithm 1 Algorithme naïf du cercle minimum englobant

Require: Un ensemble de points P dans le plan
Ensure: Le plus petit cercle englobant tous les points de P

$C \leftarrow$ cercle nul (centre à l'origine, rayon 0)
for all $(A, B) \in P \times P$ et $A \neq B$ **do**
    $C' \leftarrow$ Cercle passant par A et B
    if C' couvre tous les points de P et $C'.rayon < C.rayon$ **then**
        $C \leftarrow C'$
    end if
end for
for all $(A, B, C) \in P \times P \times P$ et $A \neq B \neq C$ **do**
    $C' \leftarrow$ Cercle circonscrit à A, B, C
    if C' couvre tous les points de P et $C'.rayon < C.rayon$ **then**
        $C \leftarrow C'$
    end if
end for
return C

2.3 Algorithme de Welzl

L'algorithme de **Welzl** est une méthode **récursive** efficace pour calculer le plus petit cercle englobant (*Minimum Enclosing Circle*). Contrairement à l'approche naïve, il fonctionne en construisant le cercle de manière incrémentale, en ajoutant les points un par un et en mettant à jour le cercle si nécessaire.

Principe :

— On sélectionne un point aléatoire et on tente de construire le cercle sans ce point.
— Si ce point est en dehors du cercle calculé, il doit faire partie de la solution, et on recommence en incluant ce point dans l'ensemble de contrainte.
— La récursivité s'arrête lorsqu'on a un cercle défini par au plus trois points.

Cet algorithme a une **complexité** espérée de $O(n)$, ce qui le rend extrêmement efficace pour traiter de grands ensembles de points.

L'algorithme de Welzl est défini par le pseudo-code suivant :

Algorithm 2 Algorithme de Welzl pour le cercle minimum englobant

Require: Ensemble de points P, ensemble de points R (initialement vide)
Ensure: Le plus petit cercle englobant tous les points de P

```
function WELZL(P, R)
    if P est vide ou |R| = 3 then
        return CERCLEMINIMAL(R)
    end if
    Sélectionner un point p aléatoire dans P
    P ← P − {p}
    C ← WELZL(P, R)
    if p ∈ C then
        return C
    end if
    return WELZL(P, R ∪ {p})
end function
function CERCLEMINIMAL(R)
    if |R| = 0 then
        return Cercle de rayon 0
    else if |R| = 1 then
        return Cercle centré sur R[0], rayon 0
    else if |R| = 2 then
        return Cercle passant par R[0] et R[1]
    else
        return Cercle circonscrit à R[0], R[1], R[2]
    end if
end function
```

2.4 Comparaison des algorithmes

- L'algorithme naïf est simple à implémenter mais devient rapidement inutilisable pour un grand nombre de points.
- L'algorithme de Welzl est plus complexe mais offre une solution bien plus efficace en pratique.

3 Implémentation

Le projet est structuré pour séparer clairement les responsabilités entre les modules principaux :

- `algorithms/` : Contient l'implémentation des méthodes de calcul.
- `Experimentation/` : Gère l'exécution des tests et la collecte des résultats.
- `resultatY/` : Regroupe les fichiers de sortie, y compris les résultats et graphiques.
- `Visualisation.py` : Script Python pour l'analyse et la visualisation des performances.

Le projet est développé en **Java SE 22**, avec l'utilisation de bibliothèques comme `supportGUI.jar` et `jfreechart` pour la gestion graphique et les visualisations. L'analyse des résultats est effectuée avec **Python** (`Matplotlib, Pandas, NumPy`).

4 Expérimentation

L'objectif de cette phase est d'évaluer les performances des algorithmes en mesurant leur **temps d'exécution** sur un large ensemble de données. Pour cela, j'ai utilisé un programme qui automatise le processus de test sur plusieurs fichiers d'entrée et enregistre les résultats obtenus.

4.1 Base de test

Les tests ont été effectués sur la **base de données de Varoumas** [1], qui contient **1664 instances** avec un minimum de **256 points par fichier**. Cette base est particulièrement bien adaptée à l'évaluation des performances, car elle couvre une grande variété de configurations de points.

4.2 Méthodologie de test

Chaque fichier de test contient une liste de points définis par leurs coordonnées (x, y). L'expérimentation a suivi les étapes suivantes :

1. **Lecture des fichiers de test** : Les fichiers sont chargés et leurs points extraits.
2. **Tri et traitement** : Les fichiers sont traités dans un ordre déterminé pour garantir une cohérence dans l'analyse des performances.
3. **Mesure des temps d'exécution** : Chaque algorithme est exécuté sur l'ensemble des points du fichier, et son temps d'exécution est mesuré en microsecondes.
4. **Enregistrement des résultats** : Les données collectées sont enregistrées dans un fichier CSV sous la forme : `Fichier, Temps Naïf (µs), Temps Welzl (µs)`

4.3 Conditions expérimentales

Les principales conditions expérimentales sont :
— Exécution sur une machine avec **Java SE 22**
— Mesure du temps d'exécution en utilisant `System.nanoTime()` pour assurer une haute précision.
— Comparaison des résultats obtenus pour analyser l'efficacité relative des algorithmes.
Les résultats de ces expérimentations seront analysés dans la section suivante.

5 Analyse des résultats

5.1 Comparaison des temps d'exécution

L'image 1 représente la comparaison des temps d'exécution des algorithmes Naïf et Welzl sur l'ensemble des fichiers de test de la base VAROUMAS. L'axe des abscisses correspond aux fichiers de test, tandis que l'axe des ordonnées indique le temps d'exécution

1. `http://www-npa.lip6.fr/~buixuan/files/cpa2024/Varoumas_benchmark.zip`

en secondes, affiché avec une **échelle logarithmique** pour mieux visualiser les écarts entre les deux algorithmes.

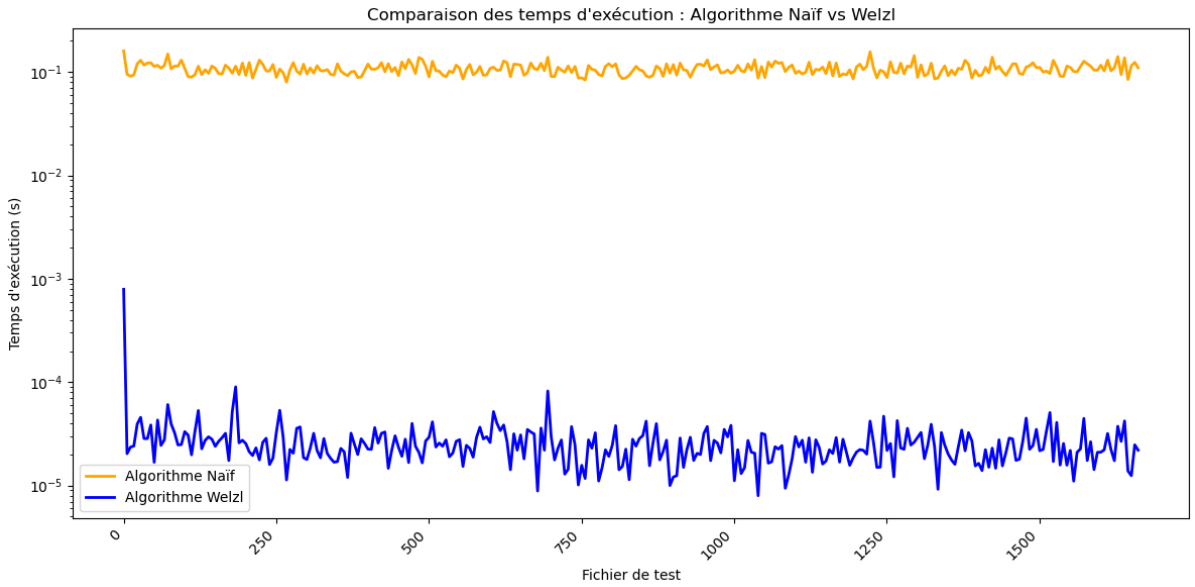

FIGURE 1 – Comparaison des temps d'exécution des algorithmes Naïf et Welzl

Sur ce graphique :

— La courbe en **orange** correspond à l'algorithme **Naïf**, et on observe qu'il présente des variations importantes, mais reste globalement beaucoup plus long en temps d'exécution.

— La courbe en **bleu** représente l'algorithme **Welzl**, qui est nettement plus rapide et relativement stable sur l'ensemble des fichiers de test.

— L'axe des ordonnées est en **échelle logarithmique**, ce qui permet de mieux visualiser les différences de plusieurs ordres de grandeur entre les deux algorithmes.

On observe que :

1. **L'algorithme naïf** présente une montée initiale importante avant de se stabiliser à un niveau constant. Cette augmentation soudaine pourrait être expliquée par des effets d'initialisation (chargement des bibliothèques, gestion mémoire) ou par la structure des premiers fichiers de test, qui pourraient contenir des configurations de points défavorables à cet algorithme.
2. Une fois cette montée passée, le temps d'exécution semble relativement stable, bien que toujours très élevé, ce qui est cohérent avec **la croissance exponentielle** de sa complexité.
3. **L'algorithme de Welzl**, en revanche, reste extrêmement rapide et stable, avec des temps d'exécution nettement inférieurs sur toute la base de test.

5.2 Comparaison des temps moyens d'exécution

L'efficacité d'un algorithme peut être évaluée en comparant son temps moyen d'exécution sur un large ensemble de tests. La figure ci-dessous illustre les temps moyens d'exécution des algorithmes **naïf** et **Welzl**.

5.2.1 Calcul des temps moyens

Le temps moyen d'exécution d'un algorithme est défini par :

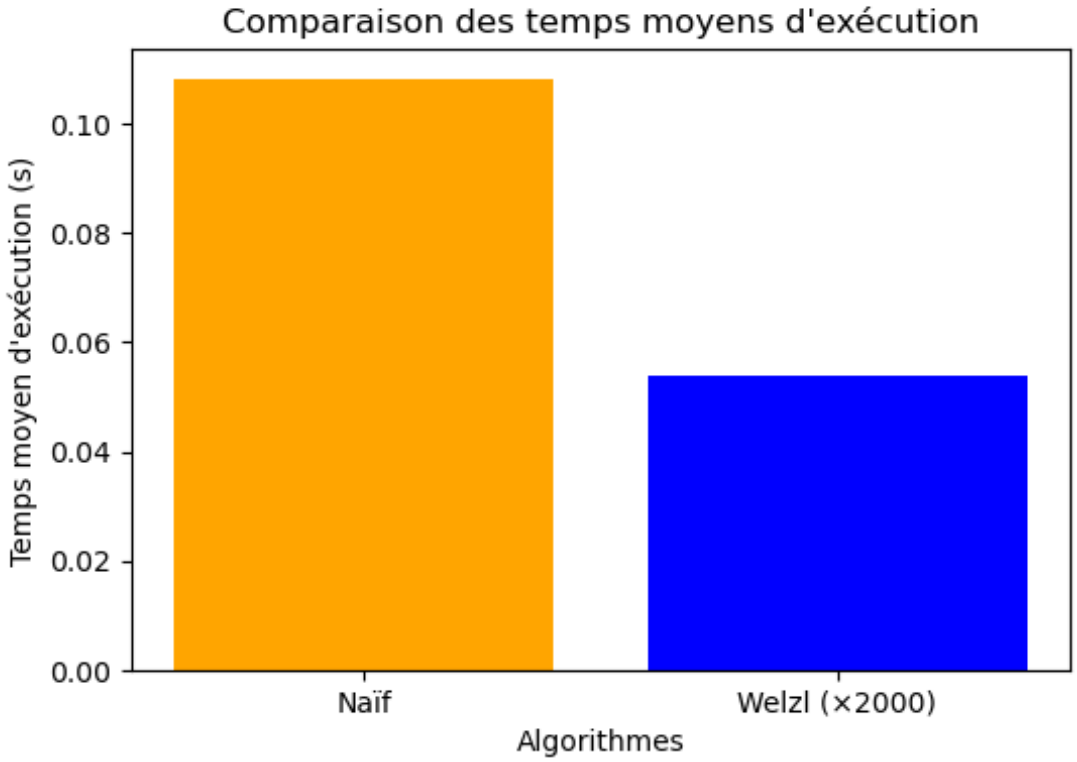

FIGURE 2 – Comparaison des temps moyens d'exécution des algorithmes

$$T_{\text{moy, naïf}} = \frac{1}{N} \sum_{i=1}^{N} T_{\text{naïf},i} \tag{1}$$

$$T_{\text{moy, Welzl}} = \frac{1}{N} \sum_{i=1}^{N} T_{\text{Welzl},i} \tag{2}$$

où :
— N est le nombre total de fichiers de test,
— $T_{\text{naïf},i}$ et $T_{\text{Welzl},i}$ sont les temps d'exécution mesurés pour chaque fichier i.
L'algorithme de Welzl étant extrêmement rapide, son temps moyen a été multiplié par un facteur de 2000 dans la figure 2 pour le rendre visible.

5.2.2 Facteur d'accélération moyen

Le facteur d'accélération moyen entre les deux algorithmes est donné par la relation suivante :

$$\text{Speedup} = \frac{T_{\text{moy, naïf}}}{T_{\text{moy, Welzl}}} \tag{3}$$

Les résultats montrent que l'algorithme de **Welzl** est en moyenne plus de **2000 fois plus rapide** que l'algorithme **naïf**.

5.2.3 Interprétation et conséquences

Les observations suivantes peuvent être faites :
— **Scalabilité :** L'algorithme naïf devient impraticable pour des ensembles de points de grande taille, tandis que l'algorithme de Welzl reste performant.

— **Impact pratique :** Si un problème prend **10 minutes** avec l'algorithme naïf, il prendrait environ **0.3 secondes** avec Welzl, rendant ce dernier bien plus adapté aux applications réelles.

— **Efficacité en temps réel :** Cette différence d'ordre de grandeur signifie que l'algorithme de Welzl pourrait être utilisé dans des applications en temps réel ou pour traiter de grands ensembles de données.

L'analyse des moyennes confirme donc que Welzl est bien supérieur en pratique, et que l'algorithme naïf est inadapté aux grandes instances.

5.3 Analyse du gain de performance

Le graphique ci-dessous représente le facteur d'accélération de l'algorithme de Welzl par rapport à l'algorithme naïf :

$$\text{Facteur d'accélération} = \frac{\text{Temps d'exécution de l'algorithme Naïf}}{\text{Temps d'exécution de l'algorithme Welzl}} \tag{4}$$

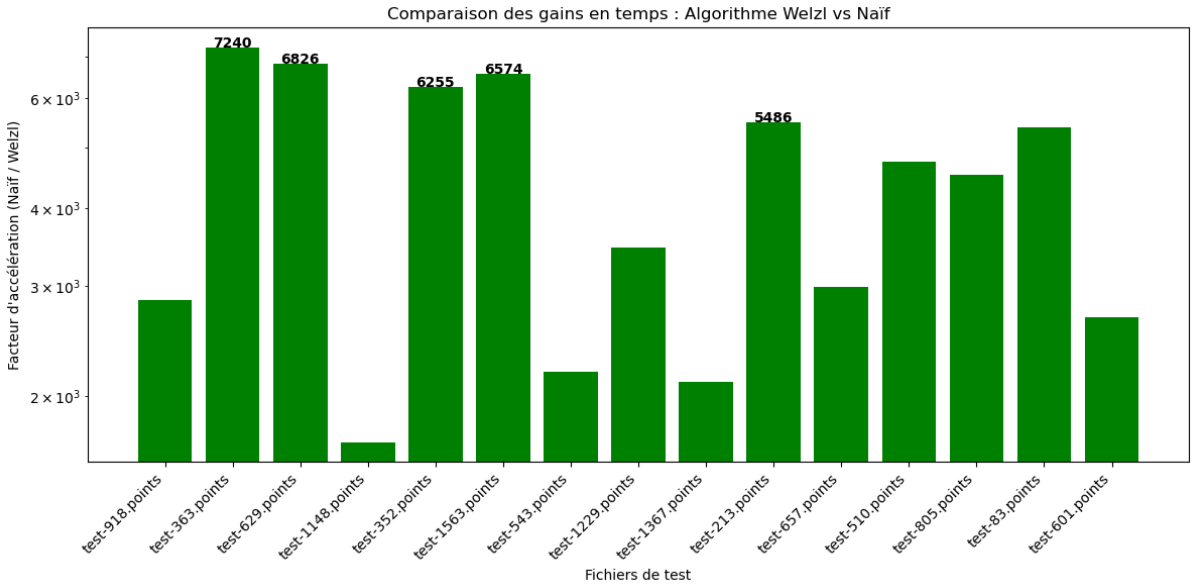

FIGURE 3 – Facteur d'accélération de l'algorithme de Welzl par rapport au naïf.

Sur cette image on observe plusieurs choses :

— L'algorithme de Welzl est en moyenne **5000 à 7000 fois plus rapide** que l'algorithme naïf.

— La variation du facteur d'accélération dépend de la structure des points. Certains fichiers, comme `test-1148.points`, montrent un gain plus faible.

— La tendance générale confirme la **scalabilité** de Welzl, contrairement au naïf qui devient inutilisable pour de grands ensembles de points.

Ainsi, cette comparaison met en évidence l'intérêt de l'algorithme de Welzl pour des applications nécessitant une grande efficacité en temps de calcul.

5.4 Résultats visuels de l'interface graphique

Pour compléter l'analyse expérimentale, j'ai testé l'exécution des deux algorithmes sur une instance spécifique de la base de test, **input.points**, contenant **256 points**.

L'objectif est d'observer directement les performances des deux méthodes sur un même jeu de données et de visualiser le temps d'exécution.

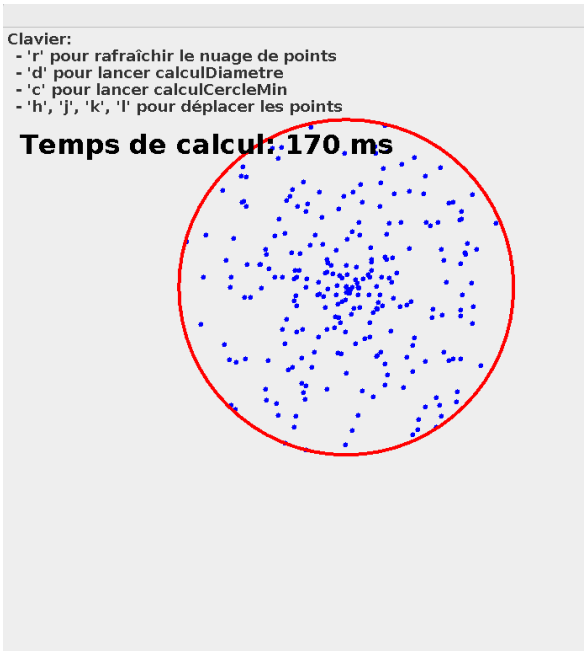

FIGURE 4 – Exécution de l'algorithme naïf - Temps : 174 ms

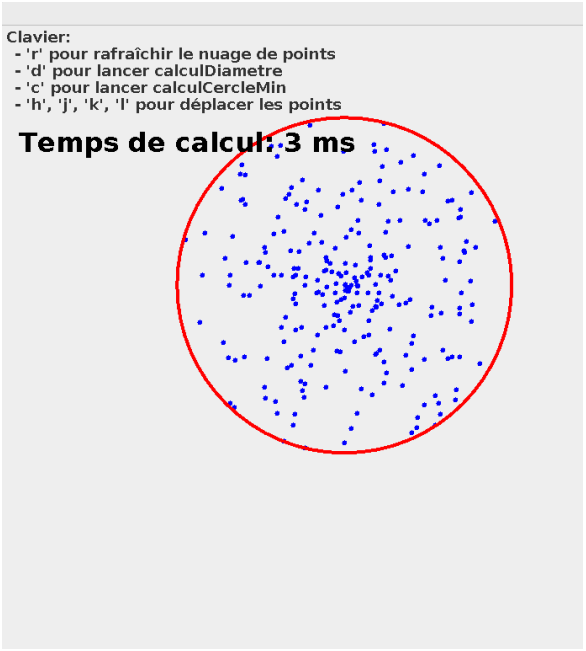

FIGURE 5 – Exécution de l'algorithme de Welzl - Temps : 3 ms

Ces images montrent clairement la différence d'efficacité entre les deux algorithmes. Sur le même ensemble de **256 points**, l'algorithme naïf prend ****170 ms****, tandis que l'algorithme de Welzl ne prend que ****3 ms****, soit une accélération d'un facteur d'environ ****58 fois****.

L'impact de cette différence de performance devient encore plus significatif lorsque l'on traite des ensembles de points plus grands comme on l'a observé plus en haut. L'algorithme naïf souffre de sa complexité exponentielle, alors que Welzl conserve une complexité quasi-linéaire en moyenne, ce qui explique cet écart considérable.

6 Conclusion et perspectives

Dans ce projet, j'ai étudié et comparé deux approches pour la résolution du problème du plus petit cercle englobant (*Minimum Enclosing Circle*). **L'algorithme naïf**, basé sur une recherche exhaustive, a montré ses limites en termes de performance avec une complexité exponentielle qui le rend rapidement inutilisable pour de grands ensembles de points.

À l'inverse, **l'algorithme de Welzl** s'est révélé extrêmement efficace, avec une complexité en moyenne **quasi-linéaire**, le rendant adapté aux grandes bases de test comme celui de **VAROUMAS** allant jusqu'à 1664 points.

Les expérimentations réalisées ont mis en évidence des écarts de performance significatifs. Les résultats obtenus montrent que Welzl est en moyenne plusieurs milliers de fois plus rapide que l'algorithme naïf, comme illustré dans les différentes analyses graphiques.

6.1 Perspectives et améliorations possibles

Bien que l'algorithme de Welzl soit performant, certaines améliorations et pistes de réflexion méritent d'être explorées :

- **Optimisation de l'implémentation** : L'implémentation actuelle repose sur une approche récursive. Une version itérative pourrait être envisagée pour améliorer la gestion de la mémoire et éviter les dépassements de pile pour des ensembles très volumineux.
- **Utilisation de structures de données avancées** : L'emploi d'arbres de recherche (KD-Tree) ou d'algorithmes de pré-traitement permettrait d'accélérer davantage la construction du cercle en réduisant le nombre de points à traiter [2].

En conclusion, ce projet a permis de mieux comprendre les enjeux liés au problème du cercle minimum englobant et d'explorer différentes approches algorithmiques. L'algorithme de Welzl, grâce à sa rapidité et son efficacité, constitue une solution optimale dans la plupart des cas pratiques. Toutefois, certaines optimisations et adaptations restent possibles pour encore améliorer ses performances, notamment dans des contextes où la rapidité d'exécution est un facteur critique.

Références

[1] GeeksforGeeks. Minimum enclosing circle using welzl's algorithm, 2024.

[2] Andrew W. Moore. A tutorial on kd-trees. *Carnegie Mellon University*, 2000.