Conception et Pratique de l'Algorithmique

Binh-Minh Bui-Xuan

Problème du Cercle Minimum

Dilyara BABANAZAROVA
28709428

Année universitaire : 2023/2024

Sommaire

1 Introduction

Le "Problème du cercle minimum", également connu sous le nom de "problème du cercle englobant minimal" , est un problème classique en géométrie computationnelle. Il s'agit de trouver le cercle de plus petit rayon qui contient un ensemble donné de points dans son intérieur ou sur sa frontière. Ce cercle minimal est essentiel dans de nombreux domaines, tels que la cartographie, la robotique, la vision par ordinateur, et bien d'autres.

L'importance de ce problème réside dans sa simplicité apparente et sa complexité réelle. Bien que le concept de trouver le cercle englobant minimal semble intuitif, la recherche d'une solution optimale est un défi significatif. Dans ce rapport, nous abordons la résolution du problème du cercle minimum en explorant deux algorithmes distincts : l'algorithme naïf et l'algorithme de Welzl. L'objectif principal de notre étude est d'analyser et de comparer ces deux approches, en mettant en évidence leurs forces, leurs faiblesses et leurs performances respectives. En outre, nous présenterons également l'algorithme de Megiddo, un algorithme probabiliste qui offre une solution efficace en temps linéaire.

L'algorithme naïf adopte une approche intuitive en se basant sur des principes géométriques simples. Il repose sur des lemmes et des théorèmes spécifiques pour assurer sa correction et évaluer sa complexité. En revanche, l'algorithme de Welzl est une méthode plus sophistiquée qui s'appuie sur des concepts avancés de géométrie algorithmique. Cette approche utilise une récursivité pour déterminer le cercle minimum couvrant à partir d'un ensemble de points donné.

2 Description formelle du problème du cercle minimum

Le problème du cercle minimum consiste à trouver le cercle de rayon minimum qui couvre un ensemble donné de points dans le plan. Formellement, étant donné un ensemble de points $P = \{p_1, p_2, ..., p_n\}$, le cercle minimum couvrant est défini comme le cercle C tel que :

- Tous les points de P sont à l'intérieur ou sur le bord de C.
- Le rayon de C est minimum parmi tous les cercles satisfaisant la première condition.

Ce problème est largement étudié en informatique géométrique en raison de son importance pratique dans divers domaines.

2.1 Structure de données utilisée

La résolution du problème du cercle minimum nécessite une représentation appropriée des points et des cercles dans le plan. Pour cela, nous utilisons deux classes principales : **Point** et **Circle**.

2.1.1 Classe Point

La classe Point représente un point dans le plan cartésien en deux dimensions. Elle utilise les coordonnées entières (x,y) pour définir la position du point.

- **Attributs** :
  - `x` : coordonnée horizontale du point.
  - `y` : coordonnée verticale du point.
- **Constructeurs** : La classe Point dispose de plusieurs constructeurs permettant d'initialiser un point avec différentes valeurs de coordonnées.
- **Méthodes utilitaires** : La classe propose des méthodes pour obtenir et définir les coordonnées x et y du point, ainsi que d'autres méthodes pour effectuer des opérations sur le point.

2.1.2 Classe Circle

La classe Circle représente un cercle avec un centre et un rayon. Le centre est représenté par un objet java.awt.Point, tandis que le rayon est un entier. De plus, la classe Circle peut également stocker une couleur pour la représentation graphique.

- **Attributs** :
  - `center`
  - `radius`
  - `color`

— **Constructeurs** : La classe Circle dispose de constructeurs pour initialiser un cercle avec un centre, un rayon et éventuellement une couleur.
— **Méthodes utilitaires** : La classe fournit des méthodes permettant d'obtenir les attributs, ainsi que d'autres méthodes pour modifier le centre ou le rayon du cercle.

3 Analyse et présentation théorique des algorithmes

Dans cette section, nous examinerons trois algorithmes bien connus pour résoudre le problème du cercle minimum : l'algorithme naïf, l'algorithme de Welzl et l'algorithme de Megiddo. Nous explorerons les principes théoriques sous-jacents à chaque approche, ainsi que les lemmes et théorèmes qui en garantissent la validité et la complexité.

3.1 Algorithme naïf

L'algorithme naïf repose sur les principes suivants :

— **Lemme 1 :** Un cercle dont le diamètre est égal à la distance entre deux points de l'ensemble et qui couvre tous les autres points est un cercle couvrant de rayon minimum.
— **Lemme 2 :** En 2D, il existe un unique cercle passant par trois points non-colinéaires.
— **Théorème :** Le problème du cercle minimum peut être résolu en temps $O(n^4)$.

3.1.1 Description de l'algorithme

L'algorithme naïf procède comme suit :

1. Si l'ensemble de points est vide, retourner `null`.
2. Examiner chaque paire de points dans l'ensemble de points pour trouver un cercle dont le diamètre est égal à la distance entre ces deux points.
3. Vérifier si ce cercle couvre tous les points de l'ensemble. Si oui, le retourner comme étant le cercle couvrant de rayon minimum.
4. Si aucun cercle couvrant n'est trouvé parmi les paires de points, passer à l'étape suivante.
5. Examiner chaque triplet de points dans l'ensemble de points pour trouver un cercle circonscrit à ces trois points.
6. Vérifier si ce cercle couvre tous les points de l'ensemble et s'il a un rayon plus petit que le cercle couvrant précédemment trouvé. Si oui, le considérer comme le cercle couvrant de rayon minimum.
7. Retourner le cercle couvrant de rayon minimum.

3.1.2 Pseudo-code de l'algorithme

Data: Ensemble de points *Points*

for *chaque point p dans Points* **do**
  for *chaque point q dans Points* **do**
    Construire un cercle c ayant pour centre $(p+q)/2$ et pour diamètre $|pq|/2$;
    if *c couvre tous les points de Points* **then**
      Retourner c;
    end
  end
end
Initialiser *resultat* à un cercle de rayon infini;
for *chaque point p dans Points* **do**
  for *chaque point q dans Points* **do**
    for *chaque point r dans Points* **do**
      Construire un cercle c circonscrit à p, q et r;
      if *c couvre tous les points de Points et que c a un rayon plus petit que celui de resultat* **then**
        Mettre à jour *resultat* avec c;
      end
    end
  end
end
Retourner *resultat*;

3.1.3 Complexité de l'algorithme

La complexité de cet algorithme est $O(n^4)$, où n est le nombre de points dans l'ensemble. Cela est dû aux quatre boucles imbriquées utilisées pour parcourir toutes les combinaisons possibles de points.

Exemple de sortie de l'algorithme naïf :

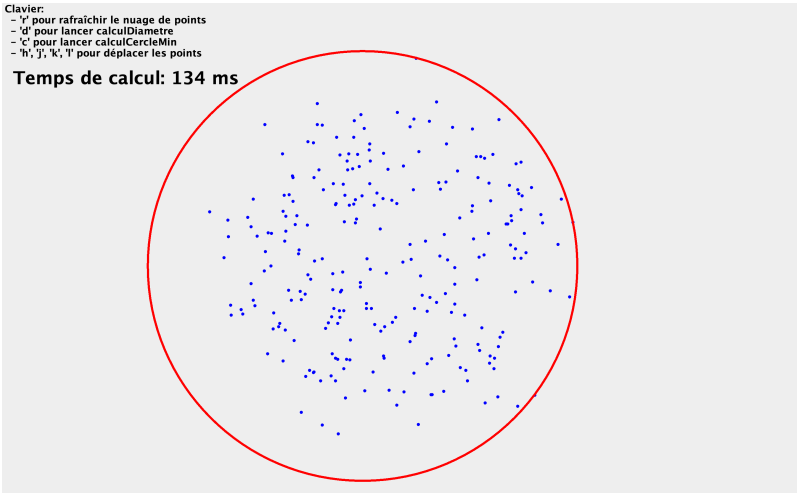

FIGURE 1 – Nombre de points d'entrée = 256, rayon du cercle résultant = 389

3.2 Algorithme de Welzl

L'algorithme de Welzl repose sur le principe fondamental du plus petit cercle englobant (PCE), qui est le cercle de rayon minimum contenant tous les points donnés dans un ensemble fini de points du plan. L'idée principale derrière l'algorithme est de réduire le problème de trouver le

cercle minimum englobant un ensemble de points à un problème de plus petite taille en retirant de manière récursive certains points de l'ensemble.

3.2.1 Description de l'algorithme

Voici comment fonctionne l'algorithme de Welzl :

1. Si l'ensemble de points est vide ou si la bordure contient trois points, on retourne le cercle englobant correspondant.
2. Sinon, on choisit un point aléatoire dans l'ensemble de points.
3. On appelle récursivement la fonction sur l'ensemble de points restants sans inclure le point choisi précédemment.
4. Si le cercle englobant retourné ne contient pas le point choisi, on l'ajoute à la bordure et on rappelle récursivement la fonction sur l'ensemble de points restants et la nouvelle bordure.
5. On retourne le cercle englobant final.

3.2.2 Pseudo-code de l'algorithme

Data: Ensemble de points *points*, Bordure de points *bordure*

if *La taille de points est égale à 0 ou la taille de bordure est égale à 3* **then**
  if *La taille de bordure est égale à 3* **then**
    Retourner le cercle englobant de trois points dans *bordure*;
  else
    Retourner un cercle de rayon 0;
  end
else
  Choisir un point aléatoire p dans *points*;
  points ← *points* sans p;
  disk ← Appeler récursivement la fonction avec les nouveaux *points* et la même *bordure*;
  if *disk n'est pas* `null` *et p est à l'extérieur de disk* **then**
    Ajouter p à la *bordure*;
    disk ← Appeler récursivement la fonction avec les nouveaux *points* et la nouvelle *bordure*;
  else
    Ne rien faire;
  end
  Retourner *disk*;
end

3.2.3 Complexité de l'algorithme

La complexité de l'algorithme de Welzl est généralement en $O(n)$, où n est le nombre de points dans l'ensemble. La raison en est que l'algorithme utilise une méthode de retour arrière récursive pour explorer les combinaisons de points, ce qui peut être réalisé en temps linéaire dans de nombreux cas.

Exemple de sortie de l'algorithme de Welzl :

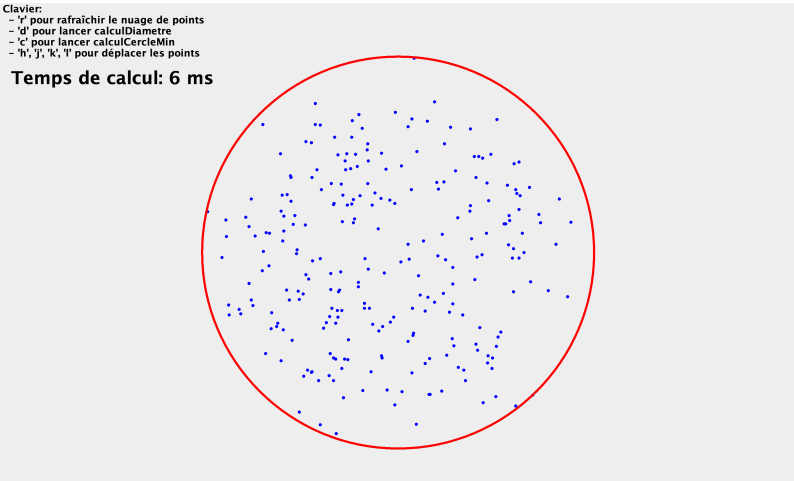

FIGURE 2 – Nombre de points d'entrée = 256, rayon du cercle résultant = 355

3.3 Algorithme de Megiddo

L'algorithme de Megiddo est un algorithme probabiliste pour trouver le cercle minimum d'un ensemble de points. Il est basé sur l'idée que le cercle minimum passe par deux ou trois points expérimentaux, ce qui permet de réduire le nombre de points à considérer. L'algorithme utilise une technique appelée "prune and search" pour réduire la taille du problème en éliminant un nombre fractionnaire de points à chaque itération.

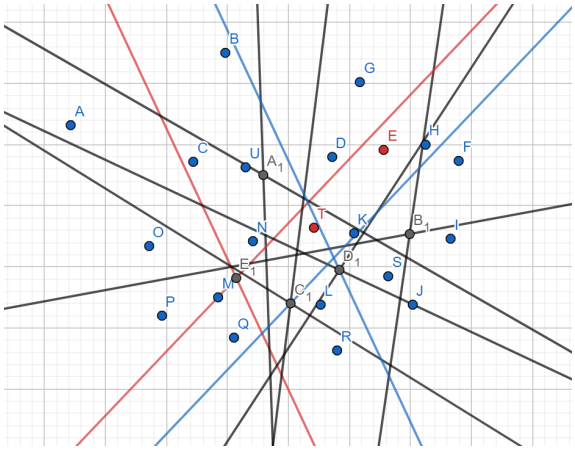

FIGURE 3 – Illustration de l'algorithme de Megiddo. [1]

3.3.1 Description de l'algorithme

L'algorithme de Megiddo fonctionne de la manière suivante :

1. Il détermine d'abord le quart de plan Q où se trouve le centre du plus petit cercle englobant, basé sur les propriétés du problème. Ensuite, il initialise un ensemble D vide qui contiendra les points sélectionnés pour former le cercle englobant, et il calcule la distance maximale r entre deux points de l'ensemble P.
2. L'algorithme utilise une boucle principale qui s'exécute tant que le nombre de points non colinéaires dans l'ensemble D est inférieur à 3.
3. Pour chaque paire de points p et q dans l'ensemble P, l'algorithme vérifie si ces points appartiennent au quart de plan Q. Si c'est le cas, il calcule l'intersection m des médiatrices de $\overline{p,q}$ pour déterminer le point médian. Si le point médian m n'appartient pas au quart de plan Q, l'algorithme sélectionne soit le point p soit le point q, en fonction de leur distance

respective par rapport au quart de plan Q. Ces points sont ajoutés à l'ensemble D pour former le cercle englobant.

4. À chaque itération de la boucle principale, le rayon r du cercle englobant est réduit de moitié ($r \leftarrow r/2$).
5. Une fois que suffisamment de points ont été sélectionnés pour former le cercle minimum englobant, l'algorithme retourne le cercle circonscrit à l'ensemble D.

3.3.2 Pseudo-code de l'algorithme

Data: Ensemble de points P

Déterminer le quart de plan Q où se trouve le centre du SED. *(Propriétés du problème)*;
$D \leftarrow \emptyset$;
$r \leftarrow \max_{p,q \in P} \text{dist}(p, q)$;
while $\|D\| < 3$ **do**
  for *chaque paire de points* $p, q \in P$ **do**
    if $p, q \in Q$ **then**
      $m \leftarrow$ intersection des médiatrices de $\overline{p, q}$;
      if $m \notin Q$ **then**
        $D \leftarrow D \cup \{p\}$ si $\text{dist}(p, Q) < \text{dist}(q, Q)$;
        $D \leftarrow D \cup \{q\}$ si $\text{dist}(p, Q) \geq \text{dist}(q, Q)$;
      end
    end
  end
  $r \leftarrow r/2$;
end
Retourner le cercle circonscrit à D;

3.3.3 Complexité de l'algorithme

La complexité temporelle de l'algorithme de Megiddo est linéaire, c'est-à-dire $O(n)$, où n est le nombre de points dans l'ensemble. Cette complexité linéaire est obtenue grâce à la technique de "prune and search" qui réduit la taille du problème à chaque itération, éliminant (n/16) points inutiles. Cette élimination est réalisée en résolvant deux fois un problème similaire où le centre du cercle englobant recherché est contraint de se situer sur une ligne donnée.

La complexité spatiale de l'algorithme de Megiddo est également linéaire, c'est-à-dire $O(n)$.

4 Tests

4.1 Méthode d'obtention des jeux de tests

Nous avons obtenu notre jeu de tests en téléchargeant l'archive `Varoumas_benchmark.zip` depuis le lien suivant : http://www-npa.lip6.fr/~buixuan/files/cpa2023/Varoumas_benchmark.zip.

Dans le répertoire fourni, nous avons trouvé un total de 1664 fichiers points, nommés `test-1.points` à `test-1664.points`. Chaque fichier contient des coordonnées de 256 points test au format `x y`. Par exemple, voici un extrait de contenu d'un fichier de test :

```
508 160
436 265
403 211
...
```

Le fichier `test-1.points` contient tous les points des autres fichiers regroupés en un seul fichier.

Nous avons utilisé la fonction suivante pour lire les données des fichiers de test :

```
public static ArrayList<Point> readPointsFromFile(String fileName) {
    ArrayList<Point> points = new ArrayList<>();
    try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] parts = line.split("␣");
            int x = Integer.parseInt(parts[0]);
            int y = Integer.parseInt(parts[1]);
            points.add(new Point(x, y));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return points;
}
```

Cette fonction lit les coordonnées des points à partir du fichier spécifié et les stocke dans une liste d'objets de type `Point`.

4.2 Test de performance

Dans cette section, nous présentons les résultats des tests de performance réalisés pour évaluer les algorithmes de cercle minimum naïf et de Welzl. Pour chaque test, nous expliquons le contexte du test, présentons le diagramme correspondant et analysons les résultats obtenus.

4.2.1 Comparaison du temps CPU moyen

Nous avons effectué des tests pour mesurer le temps CPU moyen requis par l'algorithme naïf et l'algorithme de Welzl sur un ensemble de données de tailles 256. Pour chaque test, nous avons calculé le temps CPU nécessaire à l'exécution de l'algorithme sur un jeu de données spécifique.

Les fonctions `writeCPUTimeBMDToFile` et `writeCPUTimeNaifToFile` ont été utilisées pour enregistrer le temps CPU dans des fichiers de sortie.

Ces fonctions ont été utilisées pour chaque jeu de données de VAROUMAS, à l'exception du fichier test-1.points, dans notre ensemble de tests.

Enfin, pour calculer la moyenne du temps CPU moyen pour chaque algorithme, nous avons utilisé la fonction `calculerMoyenneEcartFrequenceTempsCPU`. Cette fonction lit les temps CPU enregistrés dans les fichiers de sortie, calcule la moyenne des temps CPU et enregistre cette moyenne dans un fichier de sortie.

Voici les résultats obtenus pour la moyenne du temps CPU pour chaque algorithme :

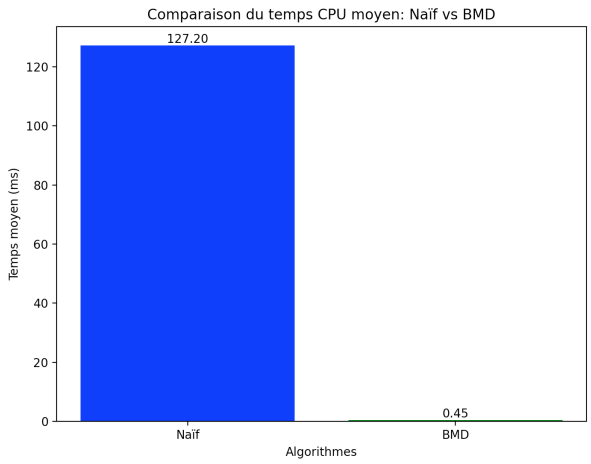

FIGURE 4 – Diagramme de comparaison du temps CPU moyen

Nous pouvons constater une différence significative entre les temps CPU moyens des deux algorithmes. L'algorithme de Welzl est beaucoup plus efficace en termes de temps CPU que l'algorithme naïf, comme en témoigne la moyenne nettement inférieure obtenue pour l'algorithme de Welzl.

4.2.2 Comparaison de l'écart-type du temps CPU moyen

De même, nous avons calculé l'écart-type du temps CPU moyen pour chaque algorithme sur différentes tailles de jeu de données. Pour calculer l'écart-type, nous utilisons la formule suivante :

$$\text{Écart-type} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

Où :

— x_i représente chaque valeur individuelle des temps CPU mesurés.
— $\bar{x}$ est la moyenne des temps CPU.
— n est le nombre total de valeurs des temps CPU.

Le diagramme en bâton ci-dessous compare les écarts-types entre les deux algorithmes :

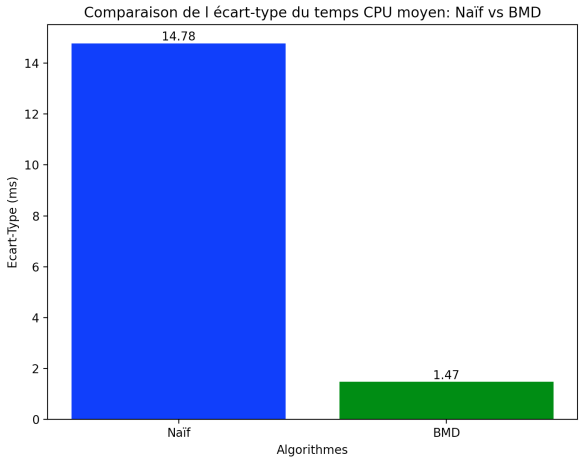

FIGURE 5 – Diagramme de comparaison de l'écart-type

L'écart-type mesure la dispersion des valeurs autour de la moyenne. Dans notre contexte, cela signifie que plus l'écart-type est élevé, plus les temps CPU individuels varient autour de la moyenne, ce qui peut indiquer une instabilité dans les performances de l'algorithme.

Nous observons une différence significative entre les écart-types des deux algorithmes. L'écart-type du temps CPU moyen pour l'algorithme de Welzl est bien inférieur à celui de l'algorithme naïf. Cela suggère que les temps CPU pour l'algorithme de Welzl sont plus cohérents et moins dispersés que ceux de l'algorithme naïf. En d'autres termes, l'algorithme de Welzl présente une plus grande stabilité dans ses performances.

4.2.3 Comparaison de la consommation de mémoire moyenne

Un autre aspect important à évaluer est la consommation de mémoire des algorithmes. Pour cela, nous avons utilisé les fonctions `writeMemoryUsageNaifToFile` et `writeMemoryUsageBMDToFile`. Ces fonctions ont permis de mesurer la consommation de mémoire de chaque algorithme en enregistrant la différence de mémoire avant et après leur exécution.

Pour appliquer ces fonctions à chaque jeu de données dans notre ensemble de tests, nous avons utilisé la fonction `calculateCPUFilesFromDirectory`, qui a enregistré les résultats dans des fichiers de sortie. Ensuite, pour obtenir la moyenne de la consommation de mémoire, nous avons utilisé la fonction `calculerMoyenneMemoire`.

Après avoir exécuté ces opérations, nous avons obtenu les résultats suivants :

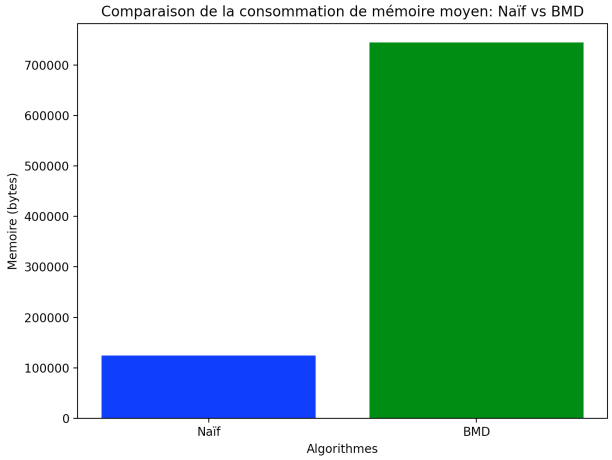

FIGURE 6 – Diagramme de comparaison de la consommation de mémoire

Nous pouvons observer que l'algorithme naïf présente une consommation moyenne de mémoire de 124418.58 bytes, tandis que l'algorithme de Welzl présente une consommation moyenne de mémoire bien plus élevée, soit 744481.36 bytes.

Cette différence significative dans la consommation de mémoire entre les deux algorithmes peut être expliquée par les différences dans leur approche algorithmique. L'algorithme naïf utilise généralement moins de mémoire car il n'a pas besoin de stocker les données intermédiaires telles que les cercles minimums. En revanche, l'algorithme de Welzl nécessite souvent plus de mémoire car il doit conserver des informations supplémentaires sur les cercles minimums trouvés jusqu'à présent pour prendre des décisions lors de la recherche de nouveaux cercles minimums.

4.2.4 Diagramme de fréquence des temps d'exécution pour l'algorithme naïf

Pour mieux comprendre la répartition des temps d'exécution des algorithmes, nous avons généré un diagramme de fréquence des temps d'exécution.

Voici le diagramme de fréquene des temps d'execution pour l'algorithme naif :

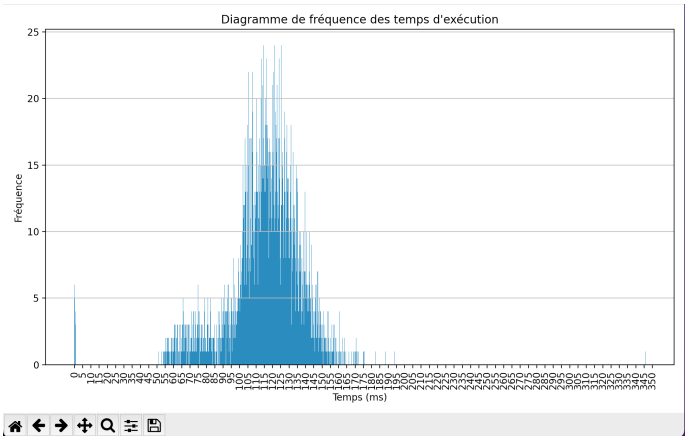

FIGURE 7 – Diagramme de fréquence pour l'algorithme naif

Le diagramme de fréquence des temps d'exécution révèle une distribution asymétrique, avec une majorité de temps d'exécution compris entre 100 et 145 ms. Cette observation est cohérente avec la complexité dans le pire cas de l'algorithme naif, qui est de $O(n^4)$. Cette complexité exponentielle implique que le temps d'exécution augmente de manière significative avec le nombre de points.

La présence de temps d'exécution plus longs (supérieurs à 200 ms) est visible, mais moins fréquente. Ces temps d'exécution correspondent à des configurations de points particulièrement complexes qui ralentissent l'algorithme naif.

En conclusion, le diagramme de fréquence des temps d'exécution confirme que l'algorithme naif n'est pas adapté pour trouver le cercle minimum couvrant tous les points lorsqu'un grand nombre de points est considéré.

4.2.5 Diagramme de fréquence des temps d'exécution pour l'algorithme de Welzl

Enfin, voici le diagramme de fréquence des temps d'exécution pour l'algorithme de Welzl :

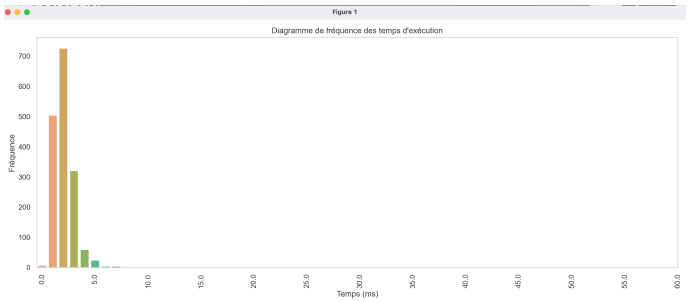

FIGURE 8 – Diagramme de fréquence pour l'algorithme de Welzl

Le diagramme de fréquence révèle une distribution symétrique des temps d'exécution pour les jeux de test de taille 256. Cette observation confirme la complexité linéaire de l'algorithme de Welzl.

La plupart des temps d'exécution se situent autour de 5 ms, indiquant une grande efficacité de l'algorithme pour cette taille de jeu de test. La distribution étroite souligne la stabilité de l'algorithme, assurant des performances prévisibles. L'absence de pics à des temps d'exécution élevés confirme la robustesse de l'algorithme face à des configurations de points inhabituelles.

5 Conclusion

La détermination du cercle minimum englobant tous les points d'un ensemble fini dans le plan est un défi fondamental aux multiples applications, stimulant ainsi la recherche et le développement d'algorithmes efficaces pour résoudre ce problème.Nous avons exploré quelques approches pour résoudre ce problème : l'algorithme naïf, l'algorithme de Welzl et l'algorithme de Megiddo.

L'algorithme de Welzl, reposant sur le concept du plus petit cercle englobant, se distingue par sa complexité en $O(n)$, ce qui le rend plus efficace que l'algorithme naïf. En utilisant une approche récursive et une méthode de retour arrière, cet algorithme offre des performances remarquables pour trouver le cercle minimum couvrant tous les points, avec des temps d'exécution courts et une stabilité démontrée sur différents jeux de test.

En revanche, l'algorithme naïf, bien que conceptuellement simple, présente une complexité exponentielle, ce qui le rend peu adapté pour des ensembles de données de grande taille.

L'algorithme de Megiddo, bien que probabiliste et susceptible de ne pas garantir la solution optimale dans tous les cas, offre une alternative intéressante avec une meilleure efficacité que l'algorithme naïf.

En conclusion, bien que l'algorithme de Welzl se distingue par sa performance et sa stabilité dans la résolution du problème du cercle minimum englobant tous les points, il convient de noter qu'il existe d'autres algorithmes pour aborder cette problématique. D'autres approches, telles que l'algorithme de Megiddo, offrent également des solutions efficaces, bien que probabilistes. Ces alternatives soulignent l'importance de la recherche continue dans ce domaine afin d'explorer de nouvelles méthodes et d'améliorer les performances de résolution du problème du cercle minimum englobant.

Références

[1] Wikipedia. https ://en.wikipedia.org/wiki/Smallest-circle$_p roblemcite_n ote - 9, 2023.