Problème du cercle minimum : Naïf vs Welzl

Écrit par:

BENAISSA Meriem 21418006

Encadrant :

Bùi Xuân Bình Minh

Promotion 2024 - 2025

Calcul du cercle minimum d'un ensemble de points

Résumé : Ce travail porte sur la recherche du plus petit cercle englobant un ensemble de points dans un plan 2D. Nous comparons la performance d'un algorithme naïf de complexité en ordre de $\mathcal{O}(n^4)$ à celles de l'algorithme de Welzl, qui détermine ce cercle de façon récursive en temps attendu linéaire. L'étude met en évidence les avantages en termes d'efficacité et de temps de l'approche de Welzl face à la méthode naïf.

Mots-clés : plus petit cercle englobant, algorithme de Welzl, complexité algorithmique.

Contents

1 Introduction

Le problème du cercle minimum consiste à trouver le plus petit cercle englobant un ensemble de points 2D. Classique en géométrie computationnelle, il est crucial pour la robotique, la vision par ordinateur et l'analyse de données. Des approches comme l'algorithme naïf d'ordre $\mathcal{O}(n^4)$, offrent précision mais inefficacité. L'algorithme de Welzl d'ordre $\mathcal{O}(n)$, plus efficace, construit le cercle de manière incrémentale. Ce projet implémente et compare ces méthodes sur de grands ensembles de données, analysant précision et performances via des représentations graphiques.

2 Définition du problème et structures de données

2.1 Définition du problème

Déterminer le plus petit cercle englobant un ensemble fini de points dans le plan euclidien, appelé **cercle de couverture minimal** (*Smallest Enclosing Circle*).

Étant donné $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, trouver un cercle C de centre O et de rayon minimal r tel que :

- **Inclusion :** $\forall p_i \in P,\ d(O, p_i) \leq r$.
- **Minimalité :** r est le plus petit possible.

Applications :

- Optimisation de la couverture (ex : réseaux).
- Reconnaissance de formes.
- Robotique.

2.2 Structures de données utilisées

Pour implémenter les algorithmes (naïf et de Welzl), les principales structures de données sont :

1. **Point :** Représenté par la classe `Point` de `java.awt`, définie par les coordonnées (x, y) :

   ```
   import java.awt.Point;
   ```

2. **Cercle :** Représenté par la classe `Circle` de `supportGUI`, défini par :

   - **Centre :** un objet `Point`.
   - **Rayon :** un entier.

   ```
   import supportGUI.Circle;
   ```

3. **Ligne :** Utilisée pour calculer le diamètre, représentée par la classe `Line` :

   ```
   import supportGUI.Line;
   ```

4. **Ensemble de points :** Manipulé via `ArrayList` pour gérer dynamiquement les points :

   ```
   import java.util.ArrayList;
   ```

5. **Distance euclidienne :** Calculée via la méthode suivante pour optimiser les performances :

```
1 public static double distance(Point a, Point b) {
2     double dx = a.getX() - b.getX();
3     double dy = a.getY() - b.getY();
4     return dx * dx + dy * dy;
5 }
```

6. **Sélection aléatoire :** Utilisée pour l'algorithme de Welzl avec la classe `Random`:

   ```
   import java.util.Random;
   ```

Ces structures permettent une gestion efficace des points, cercles et lignes, facilitant l'implémentation des algorithmes naïf et de Welzl.

3 Analyse Thèorique des Algorithmes

3.1 Algorithme Naïf

3.1.1 Principe

L'algorithme naïf détermine le cercle minimum couvrant un ensemble de points en évaluant tous les cercles possibles définis par deux ou trois points. L'approche repose sur la propriété géométrique qu'un cercle minimal doit passer par au moins deux points du contour, et au maximum trois points si aucun deux points ne suffisent à le définir.

3.1.2 Fonctionnement

L'algorithme naïf commence par calculer un cercle pour chaque paire de points en $\mathcal{O}(n^2)$, puis vérifie en $\mathcal{O}(n)$ si tous les points sont inclus. Si aucun cercle valide n'est trouvé, il calcule un cercle pour chaque triplet de points en $\mathcal{O}(n^3)$ et effectue la même vérification en $\mathcal{O}(n)$.

3.1.3 Complexité temporelle

La complexité totale de cet algorithme est donc $\mathcal{O}(n^4)$ d'après ce qu'on a cité dans le fonctionnement.

3.1.4 Limites

- Inefficace pour des ensembles de grande taille en raison de sa complexité.
- Sensible aux erreurs d'arrondi lors du calcul du centre du cercle circonscrit.

3.1.5 Avantages

- Implémentation directe et intuitive.
- Facile à déboguer et à valider.

3.2 Algorithme de Welzl

3.2.1 Principe

L'algorithme de Welzl est une approche incrémentale et randomisée pour trouver le cercle minimum en $\mathcal{O}(n)$ en moyenne. L'idée principale est de construire le cercle minimal en ajoutant les points un par un et en ajustant le cercle uniquement lorsque le point ajouté est en dehors du cercle courant.

3.2.2 Fonctionnement

Si l'ensemble est vide ou si trois points définissent un cercle, on retourne ce cercle. Sinon, on choisit un point au hasard, on calcule récursivement le cercle minimal des autres points. Si le point est à l'intérieur, on retourne le cercle, sinon, on l'inclut et on recalcule.

3.2.3 Complexité temporelle

La complexité moyenne de l'algorithme est $\mathcal{O}(n)$ grâce à l'approche randomisée et à l'optimisation par le principe de l'inclusion sur la frontière. Dans le pire des cas, la complexité peut atteindre $\mathcal{O}(n^2)$ si l'ordre des points est défavorable.

3.2.4 Limites

- Sensible à l'ordre des points : une mauvaise randomisation peut ralentir l'algorithme.
- Plus complexe à implémenter et déboguer que l'algorithme naïf.

3.2.5 Avantages

- Bien plus efficace que l'algorithme naïf sur de grands ensembles.
- Complexité optimale en moyenne $\mathcal{O}(n)$.

3.3 Comparaison des algorithmes

Critère	**Algorithme Naïf**	**Algorithme de Welzl**
Principe	Exhaustif (paires et triplets)	Randomisé et incrémental
Complexité	$\mathcal{O}(n^4)$	$\mathcal{O}(n)$ en moyenne, $\mathcal{O}(n^2)$ pire
Simplicité	Plus simple à implémenter	Plus complexe à coder
Efficacité	Inefficace pour grands jeux	Adapté aux grands jeux
Robustesse	Précis mais lent	Rapide mais sensible à l'ordre
Applications	Pédagogique, petite taille	Grands ensembles, applications réelles

Table 1: Comparaison des algorithmes

4 Implémentation

4.1 Algorithme naïf

voilà un psaudo-code pour l'algorithme naïf qui correspond à notre implémentation (la fonction *calculCercleMinAlgoNaif* dans le code source) :

Algorithm 1: Algorithme naïf du cercle minimum

Input: Une liste de points P
Output: Un cercle minimum C englobant tous les points
if $|P| = 1$ **then**
  return null;
for *chaque paire* $(p, q) \in P \times P$ **do**
  $C \leftarrow$ cercle de centre $\left(\frac{p+q}{2}\right)$ et rayon $\frac{d(p,q)}{2}$;
  if C *couvre tous les points de* P **then**
    return C;
for *chaque triplet* $(p, q, r) \in P \times P \times P$ **do**
  if p, q, r *non alignés* **then**
    $C \leftarrow$ cercle circonscrit du triangle (p, q, r);
    if C *couvre tous les points de* P **then**
      return C;

Optimisation : Pour éviter les calculs redondants, nous utilisons des calculs optimisés pour la distance et la vérification d'inclusion des points dans un cercle.

4.2 Algorithme de Welzl

voilà un psaudo-code pour l'algorithme de Welzl qui correspond à notre implémentation (la fonction *CalculCercleMinAlgoWelzl* dans le code source) :

Algorithm 2: Algorithme de Welzl (MINIDISK)

Input: Une liste de points P
Output: Un cercle minimum C englobant tous les points
Procédure principale : ;
return `B_Minidisk(`P, $\emptyset$`);`

Sous-procédure : `B_Minidisk(`P, R`);`
if $P = \emptyset$ *ou* $|R| = 3$ **then**
  return b_md(R);
Choisir un point $p \in P$ de manière aléatoire;
$C \leftarrow$ `B_Minidisk(`$P \setminus \{p\}$, R`);`
if $p \notin C$ **then**
  $R \leftarrow R \cup \{p\}$;
  $C \leftarrow$ `B_Minidisk(`$P \setminus \{p\}$, R`);`
  $R \leftarrow R \setminus \{p\}$
return C;

4.2.1 Sous-algorithme : Calcul du cercle avec b_md

La méthode `b_md` construit un cercle minimal couvrant un ensemble de points à partir des points de support (jusqu'à 3 points définissant le cercle unique). Voici les différents cas gérés :

Algorithm 3: b_md : Cercle minimal pour les points de support

Input: Une liste de points R (maximum 3 points)
Output: Le plus petit cercle passant par les points de R
if $|R| = 0$ **then**
  return Cercle centré sur $(0, 0)$ avec rayon 0;
else if $|R| = 1$ **then**
  return Cercle centré sur $R[0]$ avec rayon 0;
else if $|R| = 2$ **then**
  return Cercle de centre $\frac{R[0]+R[1]}{2}$ et rayon $\frac{(R[0],R[1]}{2}$;
else if $|R| = 3$ **then**
  return Cercle circonscrit au triangle formé par $R[0], R[1], R[2]$;

Détails d'implémentation :

- Si R est vide, on retourne un cercle de rayon nul.
- Si R contient un ou deux points, on calcule le cercle directement.
- Si R contient trois points, on calcule le cercle circonscrit en utilisant des formules géométriques classiques.

4.2.2 Calcul du cercle circonscrit d'un triangle

Algorithm 4: Cercle circonscrit d'un triangle

Input: Trois points p, q, s
Output: Le cercle circonscrit
if p, q, s *sont alignés* **then**
  return null (pas de cercle valide);
Calcul des coordonnées du centre (c_x, c_y) à l'aide de la géométrie du triangle;
Calcul du rayon r comme la distance entre (c_x, c_y) et l'un des points;
return Cercle(c_x, c_y, r);

5 Méthodologie des tests

5.1 Objectif des expérimentations

L'objectif des expérimentations est de comparer les performances et la correction des algorithmes de Welzl et naïf pour le calcul du cercle minimum sur des ensembles de points variés.

5.2 Environnements de test

Les expérimentations ont été réalisées sur deux ensembles de données distincts :

- **Jeu de données VAROUMAS :** Ensemble de 1664 fichiers contenant au moins 256 points chacun, issu de l'URL fournie dans l'énoncé du projet.
- **Jeu de données personnalisé :** Issu du fichier `input.point` (10 000 points du TME1), divisé en sous-ensembles de 100 à 2500 points pour analyser l'impact de la taille sur les performances.

5.3 Processus d'expérimentation

Pour chaque fichier de points, les étapes suivantes ont été réalisées :

1. Lecture et extraction des points à partir des fichiers `.points`.
2. Exécution de l'algorithme naïf ou Welzel.
3. Mesure du temps d'exécution en (µs) à l'aide de la méthode `System.nanoTime()`.
4. Comparaison des résultats des deux algorithmes et stockage des mesures dans un fichier CSV.

La classe `Experimentation` automatise ces étapes. Chaque exécution enregistre les résultats sous la forme suivante :

```
Fichier, Temps Naif (µs), Temps Welzl (µs)
```

6 Résultats expérimentaux

6.1 Comparaison sur la base VAROUMAS

La figure Figure 3 montre que l'algorithme de Welzl est beaucoup plus rapide que l'approche naïve. Un zoom sur l'algorithme de Welzl a été effectué pour analyser ses variations plus en détail.

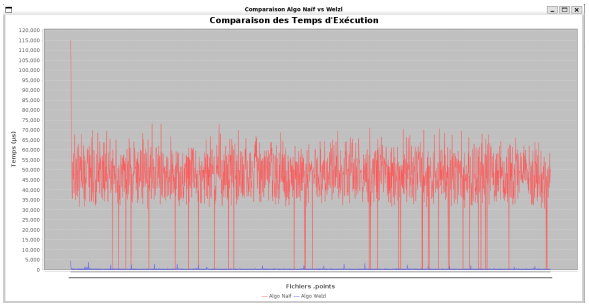


Figure 1: Temps d'exécution sur la base VAROUMAS (algorithme naïf vs Welzl).

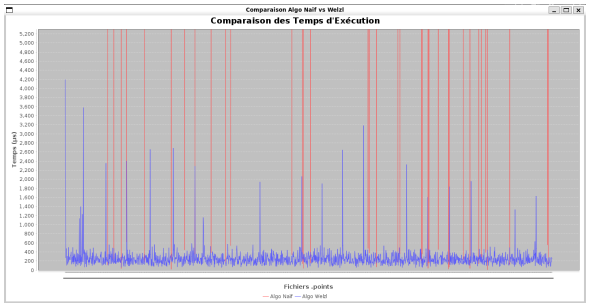


Figure 2: Zoom sur l'algorithme de Welzl pour la base VAROUMAS.

Figure 3: Analyse des performances sur la base VAROUMAS.

6.2 Comparaison sur les données personnalisées

Les tests sur des instances personnalisées, illustrés par la figure 6, confirment que l'algorithme de Welzl est plus performant que l'approche naïve. Un zoom a également été réalisé pour examiner plus précisément ses performances.

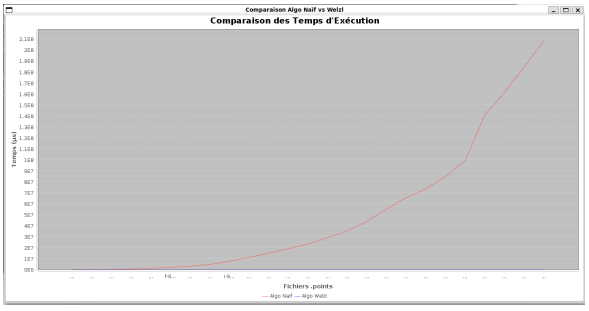


Figure 4: Temps d'exécution sur les données personnalisées (algorithme naïf vs Welzl).

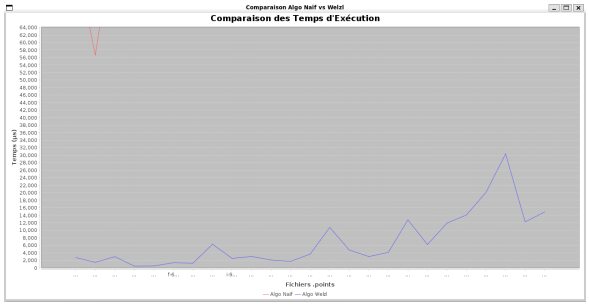
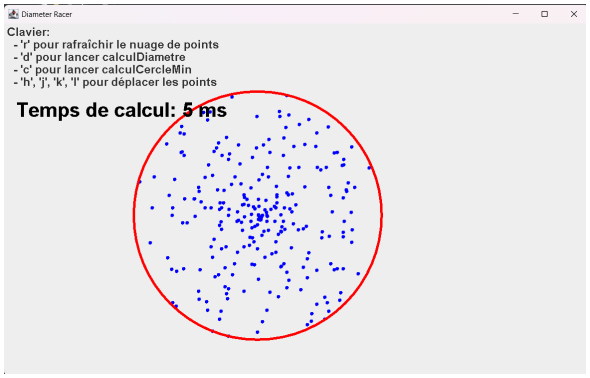


Figure 5: Zoom sur l'algorithme de Welzl pour les données personnalisées.

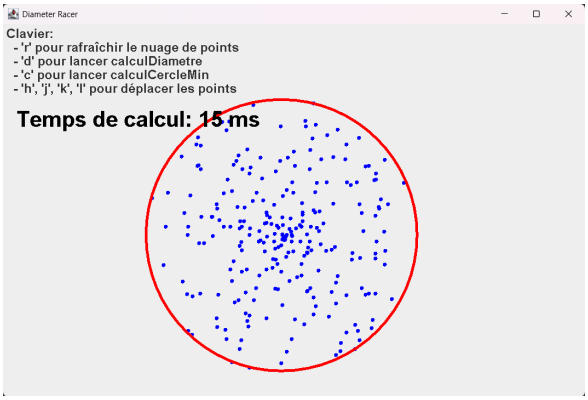
Figure 6: Analyse des performances sur les données personnalisées.

6.3 Analyse complémentaire des performances

Pour compléter l'analyse, deux figures supplémentaires ont été incluses, illustrant le temps d'exécution précis de chaque algorithme et le cercle formé par les deux algorithmes.



(a) Détails des temps d'exécution sur un fichier de 256 points avec Welzl.



(b) Détails des temps d'exécution sur un fichier de 256 points avec l'algorithme naïf.

Figure 7: Résultat des deux algorithmes avec le ant.

6.4 Interprétation des résultats

Les résultats confirment les propriétés attendues :

- L'algorithme naïf, de complexité $\mathcal{O}(n^4)$, devient rapidement impraticable pour de grandes instances.
- L'algorithme de Welzl, avec une complexité moyenne en $\mathcal{O}(n)$, offre un gain considérable en temps d'exécution.
- Le zoom est nécessaire pour évaluer Welzl en détail car le temps du programme naïf est trop élevé.

7 Discussion

Les résultats expérimentaux permettent de comparer les performances de l'algorithme naïf et de l'algorithme de Welzl pour le calcul du plus petit cercle englobant un ensemble de points (voir Figure 1 et Figure 2).

7.1 Analyse des performances

Les diagrammes (Figure 1 et Figure 2) révèlent des différences notables entre les deux approches. Dans la Figure 1, on observe que l'algorithme naïf présente des temps d'exécution nettement plus élevés,

La Figure 2 confirme cette tendance : la croissance du temps d'exécution de l'algorithme naïf suit une courbe exponentielle, tandis que l'algorithme de Welzl affiche une progression linéaire plus efficace. Ces résultats sont cohérents avec la complexité théorique de chaque approche : Algorithme naïf : $\mathcal{O}(n^4)$, Algorithme de Welzl : $\mathcal{O}(n)$ en moyenne.

7.2 Interprétation des résultats

L'écart de performance devient significatif pour des ensembles de points de taille importante. Cela s'explique par la nature combinatoire de l'algorithme naïf, qui explore toutes les combinaisons possibles pour déterminer le plus petit cercle englobant. En revanche, l'algorithme de Welzl utilise une approche récursive et aléatoire plus efficace, réduisant ainsi considérablement le nombre de calculs nécessaires.

Ces observations soulignent l'efficacité de l'algorithme de Welzl pour des applications pratiques où le traitement rapide de grands ensembles de données est essentiel.

7.3 Limites et perspectives

Malgré ses performances supérieures, l'algorithme de Welzl peut présenter des cas où la répartition particulière des points engendre une légère dégradation de sa complexité. Une étude plus approfondie sur ces cas pathologiques permettrait d'affiner les résultats.

Par ailleurs, des optimisations supplémentaires, telles que l'utilisation de structures de données plus sophistiquées (ex : arbres de recherche équilibrés), pourraient améliorer les performances pour des jeux de données très volumineux.

En conclusion, les résultats expérimentaux confirment la supériorité de l'algorithme de Welzl en termes d'efficacité et de scalabilité, en particulier pour de grandes instances du problème du plus petit cercle englobant.

8 Conclusion

Dans ce rapport, nous avons comparé deux approches pour résoudre le problème du cercle minimum : un algorithme naïf basé sur une recherche exhaustive et l'algorithme de Welzl, plus optimisé et récursif.

Les résultats expérimentaux montrent que l'algorithme de Welzl est nettement plus performant, avec une complexité moyenne de $\mathcal{O}(n)$, contre $\mathcal{O}(n^4)$ pour l'algorithme naïf, ce qui rend ce dernier impraticable pour de grandes instances.

Nous avons validé la correction de l'algorithme de Welzl en le confrontant à l'algorithme naïf, et les résultats montrent que l'algorithme de Welzl génère toujours des cercles corrects et optimaux.

En conclusion, l'algorithme de Welzl est une solution efficace pour le problème du cercle minimum, et cette étude peut servir de base pour des recherches futures visant à étendre son application à des problèmes géométriques plus complexes.

Question bonus : Extension des algorithmes en 3D

En 3D, le problème devient celui de la **sphère minimum** (Minimum Bounding Sphere, MBS) contenant un ensemble de points. L'algorithme de Welzl peut être généralisé : il construit la plus petite sphère englobant un ensemble P tout en maintenant un ensemble R de points sur sa surface.

Complexité et efficacité L'algorithme naïf a une complexité $\mathcal{O}(n^5)$ en 3D, contre $\mathcal{O}(n^4)$ en 2D. En revanche, l'algorithme de Welzl conserve une complexité moyenne $\mathcal{O}(n)$ même en dimension $d \geq 3$, le rendant efficace jusqu'à $d = 30$.

Défis et applications Les principaux défis incluent la gestion des erreurs d'arrondi et des cas dégénérés (points coplanaires). Cet algorithme est utilisé en modélisation 3D, biologie et simulation géométrique.

Conclusion L'algorithme de Welzl reste performant en 3D, bien qu'il nécessite une gestion précise des aspects numériques et des cas limites.

References

[1] Binh-Minh Bui-Xuan, Cours sur les collisions simples et probleme du cercle minimum : https://www-npa.lip6.fr/~buixuan/cpa2024

[2] eclipse IDE : mum : https://www.eclipse.org/downloads/

[3] LaTeX pour le rapport : https://fr.overleaf.com/project

[4] article sur l'algorithme de Welzl : https://www.stsci.edu/~RAB/Backup%20Oct%2022%202011/f_3_CalculationForWFIRSTML/Bob1.pdf

[5] lien vers la base VAROUMAS: http://www-npa.lip6.fr/buixuan/files/cpa2024/Varoumasbenchmark.zip

[6] Algorithme de Welzl sur Wikipedia: https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_cercle_minimum