CFA INSTA
Institut National Supérieur des Technologies Avancées

L'algorithme Welzl résolvant le problème du cercle minimum

Table de matière

1. Introduction

Il existe divers problèmes dans plusieurs domaines qui peuvent être ramenés à la détermination d'un cercle minimal englobant un ensemble donné de n points dans le plan. Ces problèmes comprennent, par exemple, la détection de collisions d'objets dans les jeux vidéo, le calcul du point d'impact et du rayon d'action d'une bombe pour atteindre des objectifs spécifiques (dans le domaine militaire), ou encore la détermination des villes couvertes par un réseau dans le domaine des télécommunications.

Pour résoudre ce problème, plusieurs approches ont été proposées, parmi lesquelles on peut citer l'algorithme de Ritter, l'algorithme de Shamos, l'algorithme de Welzl, etc. Une approche naïve consisterait à parcourir tous les couples de points et à vérifier si le cercle engendré contient tous les points de l'ensemble initial. Cependant, cette méthode présente un inconvénient majeur en raison de sa complexité dans le pire des cas, qui est en O(n^4).

Parmi les algorithmes proposés, nous examinerons l'approche de Welzl pour résoudre ce problème et comparerons ses résultats à ceux de l'algorithme naïf. L'algorithme de Welzl est un algorithme récursif qui fournit un cercle minimal incrémental à chaque itération, c'est-à-dire qu'il augmente le diamètre du cercle jusqu'à ce qu'il englobe tous les points.

L'objectif principal de ce projet est de réaliser une étude théorique et expérimentale de l'algorithme de Welzl ainsi que de l'algorithme naïf.

2. Définitions et Notations Utilisées

1. Cercle minimum couvrant un ensemble de points

Le cercle minimum couvrant un ensemble de points est défini comme le cercle de plus petit diamètre ou le plus petit cercle contenant tous les points. Il est caractérisé par un centre, noté c, et un rayon, noté r.

2. Cas de base

- Pour un ensemble réduit à un seul point x, le cercle minimum est celui dont le centre est le point x et dont le rayon est nul.
- Pour un ensemble de deux points p et q, le cercle minimum est défini par un centre c et un rayon r :
  - Centre c : (xp + xq)/2 , (yp + yq)/2
  - Rayon r : distance(p, q), où distance(p, q) représente la distance euclidienne entre les points p et q.
- Pour un ensemble de trois points a, b, et c, le cercle minimum est le cercle circonscrit au triangle abc.

3. la distance entre deux point p et q

la distance en p et q se calcul par la formule suivante :

$$\text{distance(p,q)} = \sqrt{(xb-xa)^2+(yb-ya)^2}$$

4. le cercle circonscrit

le cercle circonscrit a un triangle abc est définit par les formules suivantes :

soit d = (a.x*(b.y-c.y)+ b.x*(c.y-a.y)+c.x*(a.y-b.y))*2 ;

et norm(Point a) = (a.x*a.x)+(a.y*a.y) ;

les coordonnées du centre du cercle qui passe par les 3 points a,b et c seront calculer par les deux expressions suivantes:

xc = ((norm(a)*(b.y-c.y)) + (norm(b)*(c.y-a.y)) + (norm(c)*(a.y-b.y)))/d ;

yc = ((norm(a)*(c.x-b.x)) + (norm(b)*(a.x-c.x)) + (norm(c)*(b.x-a.x)))/d ;

le cercle circonscrit au triangle abc est définit par :

le centre c= point[xc , yc] et de rayon r = distance(a,c) ;

3. Problème de cercle minimum

3.1. Algorithme Naïf

3.1.1. Principe

Soit P un ensemble de points dans le plan , soit C le cercle minimum de rayon r contenant tous les points de P , pour déterminer ce cercle en appui sur les deux lemmes suivant :

Lemme 1 : si un cercle de diamètre égale à la distance de deux points de la liste couvre tout autre point de la liste, alors ce cercle est un cercle couvrant de rayon minimum.

Lemme 2 : en 2D, il existe un et un seul cercle passant par 3 points non-colinéaires.

3.1.2. Algorithme

À partir de ces deux lemmes, nous créons une fonction `naif` cherche à déterminer le cercle minimum couvrant un ensemble de points en utilisant une approche naïve. Elle explore toutes les combinaisons de deux points pour vérifier si le cercle défini par ces deux points couvre tous les autres points. Si tel est le cas, elle retourne ce cercle. Sinon, elle explore toutes les combinaisons de trois points non-colinéaires pour déterminer le cercle circonscrit au triangle formé par ces points. Si ce cercle couvre tous les points, elle le compare avec le cercle minimum courant et le met à jour si nécessaire. En fin de compte, elle retourne le cercle minimum ainsi obtenu.

3.1.3. Implémentation

Pseudo-code

```
Fonction naif(points):
   Si la taille de points est inférieure à 1, retourner null

   cX, cY, cRadius, cRadiusSquared ← 0
   Pour chaque point p dans points:
      Pour chaque point q dans points:
         cX ← 0.5 * (p.x + q.x)
         cY ← 0.5 * (p.y + q.y)
         cRadiusSquared ← 0.25 * ((p.x - q.x) * (p.x - q.x) + (p.y - q.y) * (p.y - q.y))
         allHit ← true
         Pour chaque point s dans points:
            Si (s.x - cX) * (s.x - cX) + (s.y - cY) * (s.y - cY) > cRadiusSquared:
```

```
            allHit ← false
            Sortir de la boucle
        Si allHit:
            Retourner Cercle(Point(int(cX), int(cY)), int(sqrt(cRadiusSquared)))

resX, resY, resRadiusSquared ← 0, 0, MAX_VALUE
Pour i de 0 à taille de points - 1:
    Pour j de i+1 à taille de points - 1:
        Pour k de j+1 à taille de points - 1:
            p ← points[i]
            q ← points[j]
            r ← points[k]
            Si (q.x - p.x) * (r.y - p.y) - (q.y - p.y) * (r.x - p.x) == 0:
                Continuer à la prochaine itération (les points sont colinéaires)
            Si p.y == q.y ou p.y == r.y:
                Si p.y == q.y:
                    p ← points[k]
                    r ← points[i]
                Sinon:
                    p ← points[j]
                    q ← points[i]
            mX ← 0.5 * (p.x + q.x)
            mY ← 0.5 * (p.y + q.y)
            nX ← 0.5 * (p.x + r.x)
            nY ← 0.5 * (p.y + r.y)
            alpha1 ← (q.x - p.x) / (p.y - q.y)
            beta1 ← mY - alpha1 * mX
            alpha2 ← (r.x - p.x) / (p.y - r.y)
            beta2 ← nY - alpha2 * nX
            cX ← (beta2 - beta1) / (alpha1 - alpha2)
            cY ← alpha1 * cX + beta1
            cRadiusSquared ← (p.x - cX) * (p.x - cX) + (p.y - cY) * (p.y - cY)
            Si cRadiusSquared >= resRadiusSquared:
                Continuer à la prochaine itération
            allHit ← true
            Pour chaque point s dans points:
                Si (s.x - cX) * (s.x - cX) + (s.y - cY) * (s.y - cY) > cRadiusSquared:
                    allHit ← false
                    Sortir de la boucle
            Si allHit:
                Afficher "R =", sqrt(cRadiusSquared)
                resX, resY, resRadiusSquared ← cX, cY, cRadiusSquared

Retourner Cercle(Point(int(resX), int(resY)), int(sqrt(resRadiusSquared)))
```

Algorithme 1 : Algorithme naif

3.1.4. Complexité

La complexité de l'algorithme naïf pour trouver le cercle minimum couvrant un ensemble de points est en O(n^4), où \(n\) est le nombre de points dans l'ensemble. Cela est dû aux boucles imbriquées explorant toutes les combinaisons de deux et trois points, ce qui conduit à une complexité quadratique dans le pire des cas.

3.2. Algorithme Welzl

3.2.1. Principe

L'algorithme de welzl est un algorithme récursive qui , pour un certain ensemble de points P , sert à déterminer le cercle minimum contenant tous les points de cet ensemble , la démarche de cet algorithme est incrémental.

Lemme 3 :

Soit P et R deux ensembles finis de points dans le plan ,telle que P est non vide , soit un point p de P et soit bmd(P,R) le cercle minimum.

1. s'il existe un cercle contenant tous les points de P avec les points de R appartenant au cercle , alors bmd(P,R) existe est unique

2. si p n'est pas à l'intérieur du cercle b_md(P-{p},R) alors p appartient au contours du cercle à condition qu'elle existe donc b_md(P,R)= b_md(P-{p},RU{p}).

3. si bmd(P,R) existe , alors il existe un sous ensemble S de P d'au plus Max{0,3-|R|} point dans P tel-que bmd(P,R)=bmd(S,R).

3.2.2. Algorithme

A partir du lemme 3 on déduit le fonctionnement de l'algorithme de welzl qu'on peut résumer dans les étapes suivantes :

on considère 3 ensembles:

- Q est l'ensemble des points non testés, contient tous les points au départ.
- R est l'ensemble des points testés situés sur le cercle minimum, initialement vide.
- P est l'ensemble des points situés strictement à l'intérieur du cercle, initialement vide.

On supposera pour tout l'algorithme que b-md(P,)=md(P) et b_md(P,R) ϕ est toujours définie et différent de nul ; donc a l'état initial il faut s'arranger à définir un cercle b_md0 pour passer au autre itération sinon À une étape donnée, on détermine le cercle minimum de R , en garde alors les points sur le cercle dans

R, et les autres sont transférés dans P. soit un point r choisi aléatoirement dans Q et lui en est retiré : si r se trouve à l'intérieur du cercle actuel, alors il est inclus dans P; sinon, le cercle est remplacé par un appel récursif aux ensembles P et R + r.

```java
public Circle welzl(ArrayList<Point> points) {
 return CercleMin(points, new ArrayList<Point>());
}
```

la fonction qui appel l'algorithme de Welzl

3.2.3. Implémentation

```
CercleMin(P, R):
    P_clone = copie profonde de P
    random = objet Random
    cercle = null

    Si P_clone est vide ou |R| == 3 alors
        cercle = bmd(ensemble vide, R)
    Sinon
        p = choisir un point aléatoire de P_clone
        Supprimer p de P_clone

        cercle = CercleMin(P_clone, R)

        Si cercle n'est pas nul et non estInterieur(cercle, p) alors
            Ajouter p à R
            cercle = CercleMin(P_clone, R)
            Supprimer p de R

    Retourner cercle
```

la fonction CercleMin de l'algorithme de Welzl

```
bmd(P, R):
    Si P est vide et |R| == 0 alors
        Retourner un cercle avec le centre à (0, 0) et le rayon à 10

    r = objet Random
    cercle = null

    Si |R| == 1 alors
        cercle = Cercle avec le centre à R[0] et le rayon à 0

    Si |R| == 2 alors
        cx = (R[0].x + R[1].x) / 2
        cy = (R[0].y + R[1].y) / 2
        d = distance entre R[0] et R[1] / 2
        p = Point avec les coordonnées (cx, cy)
        cercle = Cercle avec le centre à p et le rayon à ceil(d)

    Si |R| == 3 alors
        cercle = cercle_avec_3point(R[0], R[1], R[2])

    Retourner cercle
```

la fonction bmd de l'algorithme de Welzl

3.2.3. Complexité

L'algorithme de welzl a une complexité linéaire en O(n) au pire cas.

4. Résultats expérimentales

4.1. Contexte

- Equipe de projet : M. DAM Thai Long & M. PHAM Quang Minh
- Temps d'exécution: microseconde
- Les tests réalisés au cours de ce projet sont essentiellement basés sur la mesure du temps d'exécution , il ont étaient effectuées sur une machine de 4 gigabits en utilisant la base de test «Varoumas_benchmark » ; les temps d'exécution sont en nanoseconde pour préserver la précision puis convertis en microseconde.

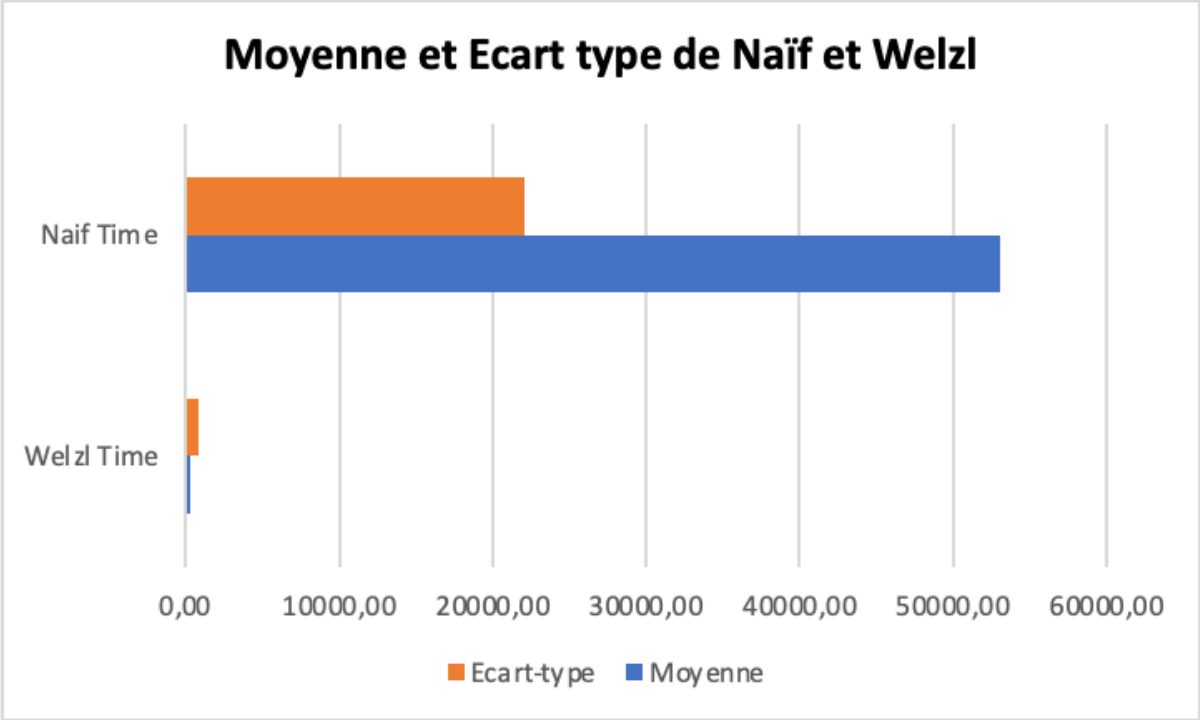
4.2. Résultat

Les résultats ont été enregistrés dans le fichier CSV, puis transformés et visualisés sous forme de diagrammes à l'aide d'Excel. Les données présentées dans le fichier "output.csv" ont été générées après l'exécution de 1664 cas de test sur ma machine. Les résultats obtenus sont presque identiques à ceux obtenus sur la machine de mon collègue. Par conséquent, nous avons décidé d'utiliser les chiffres collectés sur sa machine pour créer les graphiques dans Excel. Même si on utilise 808 cas de test mais ils sont largement suffisant pour analyser

Voici la moyenne et écart-type de ces deux solutions :

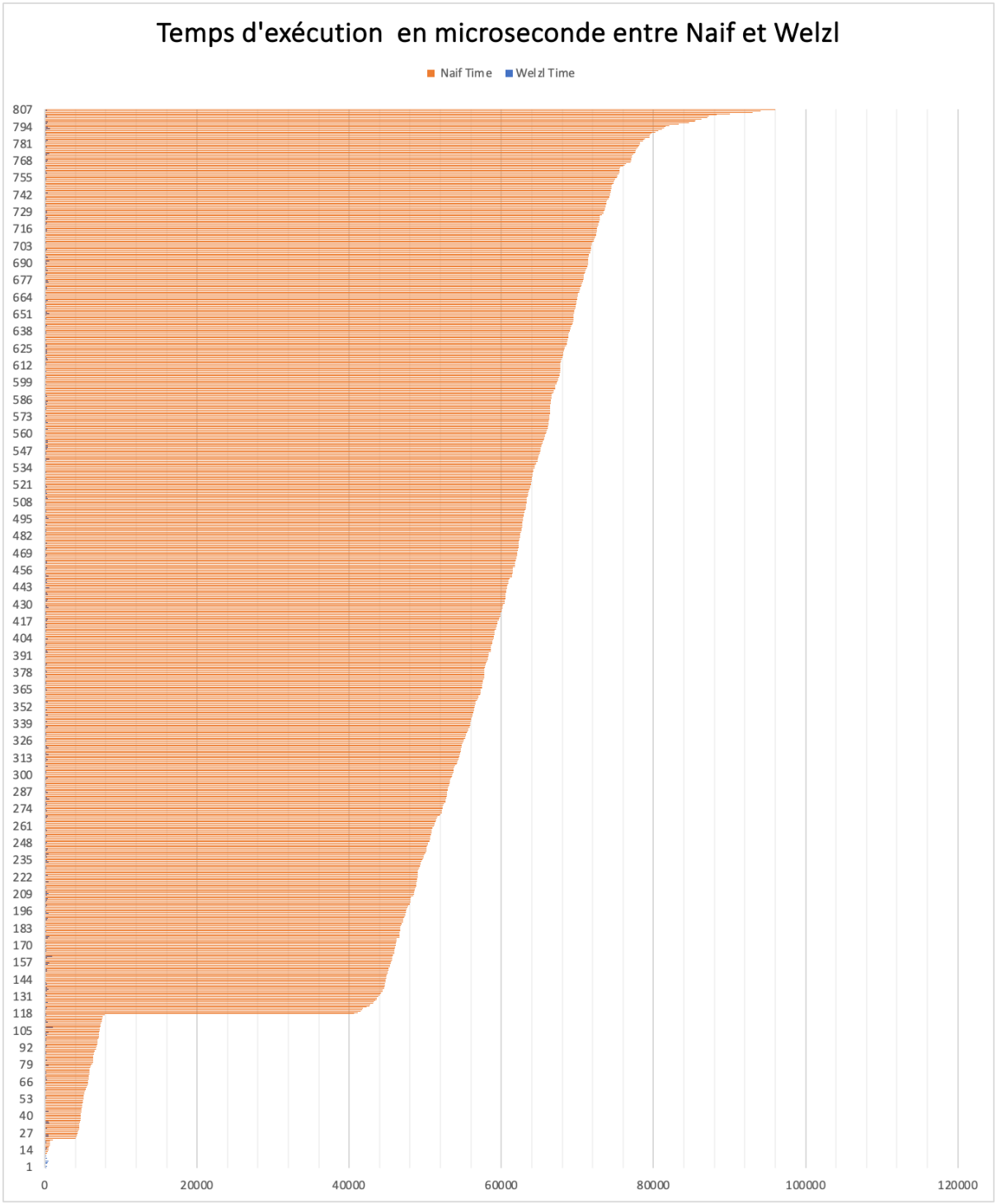
	Welzl Time	**Naif Time**
Moyenne	270,73	53063,44
Ecart-type	814,27	22053,81

Tableau du temps d'exécution moyenne et l'écart-type entre deux solutions :



En examinant la courbe des résultats enregistrés, une disparité significative dans le temps d'exécution des deux algorithmes se dégage après le tri des données par ordre croissant du temps d'exécution de l'algorithme naïf. En effet, le temps d'exécution de l'algorithme naïf atteint jusqu'à ~ 100 000 microsecondes, tandis que le temps maximal de l'algorithme de Welzl est d'environ ~23 000 microsecondes (pour un seul cas). Notons que le temps moyen de l'algorithme de Welzl est extrêmement faible, presque 0 en microsecondes, comparé à environ 53 000 microsecondes pour l'algorithme naïf.

Pour expliquer cette différence, il convient de rappeler les complexités respectives de ces deux algorithmes. L'algorithme de Welzl a une complexité en O(n), tandis que l'algorithme naïf présente une complexité en O(n^4).


Temps d'exécution en microseconde entre Naif et Welzl
Naif Time
Welzl Time
807
794
781
768
755
742
729
716
703
690
677
664
651
638
625
612
599
586
573
560
547
534
521
508
495
482
469
456
443
430
417
404
391
378
365
352
339
326
313
300
287
274
261
248
235
222
209
196
183
170
157
144
131
118
105
92
79
66
53
40
27
14
1
0
20000
40000
60000
80000
100000
120000

4.3. Analyse des cercles obtenus par les deux algorithmes

Dans cette section, nous examinons les cercles résultants de chaque fichier de test généré par les algorithmes naïf et de Welzl. La figure suivante illustre la différence d'aire entre le cercle obtenu par l'algorithme naïf et celui obtenu par l'algorithme de Welzl.

Le calcul de l'aire du cercle est effectué en préservant la plus grande précision possible, en utilisant le type de données double du langage Java. On observe généralement que les deux cercles obtenus par les deux algorithmes présentent des aires différentes, traduisant ainsi des diamètres différents. Cette disparité s'explique par les approches distinctes utilisées par les deux algorithmes pour résoudre le problème.

Notons que l'algorithme naïf produit un résultat exact. Par conséquent, nous pouvons conclure que l'algorithme de Welzl ne fournit pas toujours un résultat exact. Cependant, dans la plupart des cas, il génère un cercle très proche de celui résultant de l'algorithme naïf, voire identique.

5. Conclusion

L'algorithme naïf, utilisé comme point de référence dans cette étude, simplifie le principe du problème du cercle minimum. Cependant, son principal inconvénient réside dans sa complexité au pire cas, qui est de l'ordre de O(n^4), entraînant des temps de calcul très médiocres.

En contraste, l'algorithme proposé par Welzl adopte une approche incrémentale. Il repose sur le principe que chaque point de l'ensemble est soit à l'intérieur du cercle minimum englobant tous les points, soit appartient à ce cercle. Sa complexité au pire cas est linéaire, soit en O(n).

Les résultats expérimentaux obtenus à partir des fichiers de la base de test "Varoumas_benchmark" démontrent que l'algorithme de Welzl est efficace en termes de temps de calcul, généralement de l'ordre de la milliseconde. Il est toutefois important de noter que cet algorithme récursif génère de nombreux appels de fonctions, ce qui peut saturer la pile d'appels de fonctions et entraîner un dysfonctionnement sur des ensembles de données trop volumineux.