L'algorithme Welzl résolvant le problème du cercle minimum

résumé :

soit P un ensemble de points dans le plans , soit C le cercle minimum de rayon r englobant tout les points de P; l'algorithme de welzl est une approche incrémental permettant de calculer le cercle C en un temps linéaire , son principe est de séparer les points de P en deux , chaque point soit elle est a l'intérieur de C soit elle appartient au cercle C .

Mots_clés:

cercle minimum, welzl, naïf, complexité linéaire, récursive .

1.Introduction :

Il existe plusieurs problèmes dans de différents domaines qui se réduisent au calcul d'un cercle minimum qui couvre un ensemble donnée de n points dans le plan , comme par exemple la détection de la collision d'objets dans des jeux vidéo en informatique , déterminer le point d'impact et le rayon d'action d'une bombe pour toucher des objectifs déterminés [1] (dans le domaine militaire), ou encore la détermination des villes couvertes par le réseaux dans le monde de la télécommunication .

Pour résoudre ce problème , plusieurs approches ont était proposées , en peut citer l'algorithme de ritter , l'Algorithme de Shamos , Algorithme Welzl ...etc .

On peut déterminer le cercle minimum en utilisant un algorithme naïf qui parcourt tous couple de points et vérifier que le cercle induit contient tous les points de l'ensemble de départ, mais sont plus grand inconvénient et sa complexité qui est pire cas en O(n4) [3] .

parmi les autres algorithmes proposées on va étudier l'approche de Welzl pour résoudre ce problème , et on va comparé ses résultats a celles l'algorithme naïf.

L'algorithme de Welzl est un algorithmes récursives qui donne un cercle minimum incrémental a chaque itération c-a-d il augmente le diamètre du cercle jusqu'à se qu'il couvre tous les points .

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

2.Définitions et Notations Utilisées :

1. cercle minimum couvrant un ensemble de point:

c'est le plus petit cercle ou le cercle de plus petit Diamètre contenant tous les points, il est définit par un centre c et un rayon r .

2. les cas de bases :

- le cercle qui passe par un ensemble réduit à un seul point x, est le cercle dont le centre est le point x et de rayon nul .

- le cercle qui passe par deux points p et q , est le cercle dont le centre est c et de rayon r définie comme suit :

c = point [(xp+xq)/2 , (yp+yq)/2] .

r = distance(p , q) . avec distance(p,q) la distance euclidienne entre deux points .

-le cercle passant par 3 point a, b , c 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 :

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 determiner ce cercle en appui sur les deux lemmes suivant [3]:

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:

A partir de ces deux lemme on déduit un algorithme simple et facile a implanté qui détermine le cercle minimum c d'un ensemble de points noté points , son principe est le suivant :

-parcourir tous les points de l'ensemble de départ points deux a deux soit p et q et vérifie que le cercle induit par ces deux points couvre tout les points de l'ensemble , si un tel cercle existe alors c'est le résultat retourné .

- sinon parcourir tous les points de l'ensemble de départ points trois a trois soit p, q et r non colinéaire et vérifie que le cercle circonscrit par le triangle pqr couvre tous les points de l'ensemble

```
calculnaif CercleMin(points) :
        pour tout p dans points
            pour tout q dans points
                    c ← cercle de centre P+q/2 de diamètre |pq|
                    si c couvre tout les points alors retourner c
        pour tout p dans points
            pour tout q dans points
                    pour tout points r dans points
```

```
                                    c ← cercle circonscrit de p, q et r
                                    si c couvre tout les points  alors retourner c
```

Algorithme 1 : calculnaif CercleMin(points)[3]

pour vérifier que le cercle couvre tous les points , il faut parcourir l'ensemble de points et tester si un point p est a l'intérieur ou a l'extérieur du cercle , on s'arrête au premier points qui sort du périmètre du cercle .

```
fonction containAll(cercle , points):
    pour tout p dans points
        si distance(p, centre du cercle) > radius du cercle alors
            retourner false
    return true;
```

*fonction 1 : fonct*ion containAll(cercle , points):

3.1.3. Implémentation:

```
public Circle calculCercleMinNaif(ArrayList<Point> points) {

    if (points.isEmpty()) { return null; }

    Circle c = null;        Circle cc = null;
    int diam = Integer.MAX_VALUE;
    for (Point p : points) {
        for (Point q : points) {
            if (p != null && q != null && !p.equals(q)) {
            cc = new Circle(new Point(((p.x + q.x) / 2),((p.y + q.y) /
        2)), ((int(p.distance(q))/2));
                    if (containsAll(cc, points) && cc.getRadius() < diam) {
                            diam = cc.getRadius();
                            c = cc;
                    }
            }
        }
    }
```

```
for (Point p : points) {
        for (Point q : points) {
                for (Point r : points) {
                        if ((!p.equals(q) && !q.equals(r) && !r.equals(p))) {
                                cc = circle3point(p, q, r);
                        if (containsAll(cc, points) && cc.getRadius() < diam) {
                                        diam = cc.getRadius(); c = cc;
                        }
                        }
                }
                }
        }
        return c;
}
```

L'algorithme naïf étudié dans cette section parcours l'ensemble de points P , et comme P est un ensemble finie donc l'algorithme naïf se termine . Il commence par parcourir toutes les paires de point et si il trouve pas un cercle couvrant tous l'ensemble P alors il compare des triplet de points pour déterminer un cercle circonscrit au trois points , donc a la fin on est sûr qui va retourner un cercle minimum.

Donc l'algorithme naïf se termine et retourne un cercle minimum .

3.1.4. Complexité:

Les deux premier boucle imbriquée de l'algorithme sont en $O(n^2)$ et ,les trois boucle imbriqué qui parcours tous les points sont en $O(n^3)$ auquel en ajoute le test containAll qui a une complexité en O(n) ; donc la complexité de l'algorithme naïf au pire cas est $O(n^4)$.

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 a 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 es unique.

2. si p n'est pas a l'intérieur du cercle b_md(P-{p},R) alors p appartient au contours du cercle a 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).[2]

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: [1]

- 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, ϕ)=md(P) et b_md(P,R) est toujours définit et différent de nul ; donc a l'état initial il faut s'arranger a 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.

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

fonction 1 : la fonction qui appel l'algorithme de Welzl

```
function procédure B_MINIDISK(P,R); comment: return b_md(P,R)
        if P = ϕ [or |R|=3] then
                D := b_md(ϕ, R)
        else
        choose random p ϵ P;
        D := B_MINIDISK(P -{p}, R);
        if [D defined and] p ϵ/D then
           D := B_MINIDISK(P -{p}, R U {p});
   return D;
```

Algorithme 2 : Algorithme originale de Welzl

3.2.4. Implémentation:

```
private Circle bMinDisk(ArrayList<Point> Ps, ArrayList<Point> R) {
      ArrayList<Point> P = new ArrayList<Point>(Ps);
      Random r = new Random(); Circle D = null;

      if (P.isEmpty() || R.size() == 3) {
            D = bmd(new ArrayList<Point>(), R);
      } else {
            Point p = P.get((r.nextInt(P.size())));
            P.remove(p);       D = bMinDisk(P, R);
            if (D != null && !contains(D, p)) {
                  R.add(p);    D = bMinDisk(P, R);     R.remove(p);
            }
      }

      return D;
}
```

fonction 2 : la fonction bMinDisk de l'algorithme de Welzl

```
private Circle bmd(ArrayList<Point> P, ArrayList<Point> R) {
      if (P.isEmpty() && R.size() == 0)
            return new Circle(new Point(0, 0), 10);

      Random r = new Random();      Circle D = null;
      if (R.size() == 1) {    D = new Circle(R.get(0), 0); }
      if (R.size() == 2) {
            double cx = (R.get(0).x + R.get(1).x) / 2;
            double cy = (R.get(0).y + R.get(1).y) / 2;
            double d = R.get(0).distance(R.get(1)) / 2;
            Point p = new Point((int) cx, (int) cy);
            D = new Circle(p, (int) Math.ceil(d));
      } else {
            if (R.size() == 3)
                  D = circle3point(R.get(0), R.get(1), R.get(2));
      }
      return D;
}
```

fonction 3 : la fonction bmd de l'algorithme de Welzl

L'algorithme de welzl est un algorithme récursive, et comme a chaque itération il enlève le point de l'ensemble de départ , on est sûr que les appels récursives se terminent quand il ne reste aucune point dans P , de plus en a supposer que le b_md est toujours définie , donc l'algorithme de Welzl se termine et retourne un cercle minimum .

3.2.3. Complexité:

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

4. Résultats expérimentales :

Dans cette partie on a réaliser une étude expérimentale comparative des temps d'exécution entre l'algorithme Naïf qui a une complexité en $O(n^4)$ et l'algorithme de Welzl qui a une complexité en O(n) .

4.1. Contexte :

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 milliseconde .

4.2. Comparaison du temps de calcul en fonction du nombre de points:

On s'interresse dans cette section a l'évolution du temps en fonction du nombre de points a l'entre des deux algorithme

nombre de points	Algo Welzl (ms)	Algo Naif(ms)
10	0,034789	0,068591
50	0,198688	6,931977
100	0,20038	55,191853
200	0,237441	532,937385
500	0,40139	6378,331465
1000	6,059525	47577,044588

Tableau 1: comparaison des temps d'exécution par rapport au nombre de points

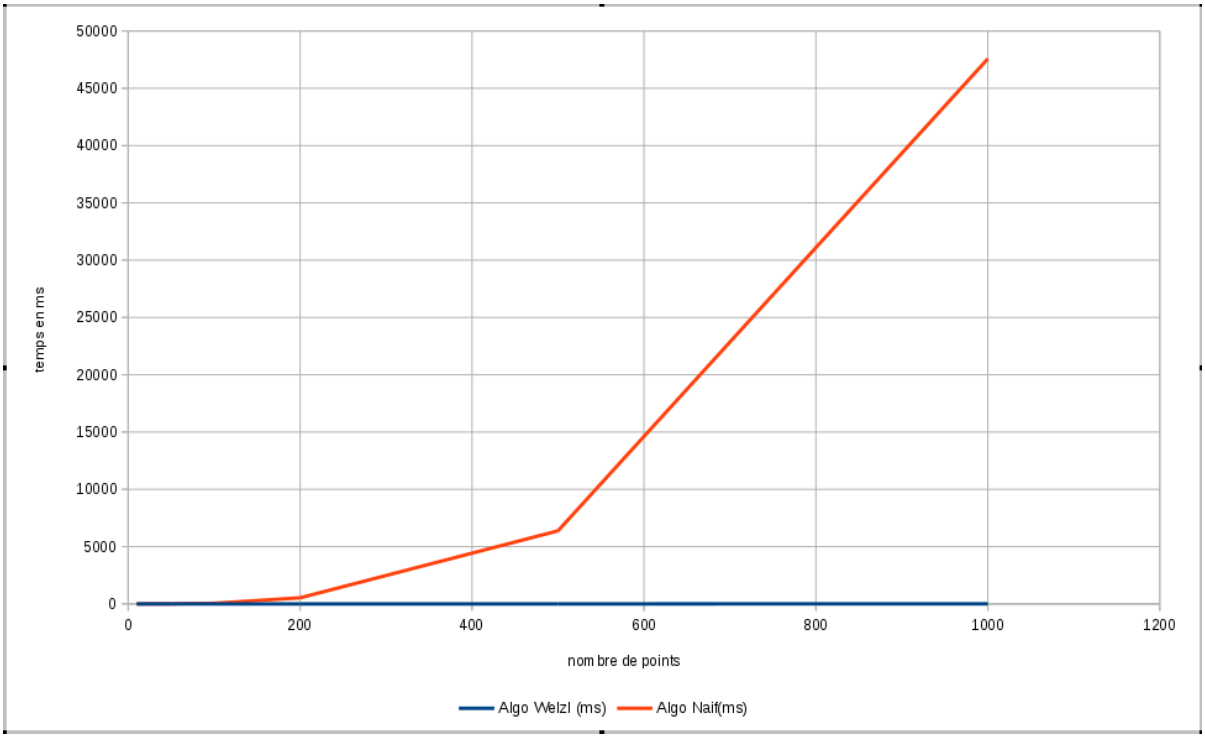


figure 1: comparaison des temps d'exécution par rapport au nombre de points

Le tableau 1 montre que le temps exécution de l'algorithme naïf croit d'une manière polynomiale en fonction de nombre de points, ce qui confirme sa complexité qui est de l'ordre de n^4 . Tandis que l'algorithme de Welzl se comporte d'une manière linéaire (de complexité en O(n)).

Les résultats obtenues montre que pour un petit nombre de points les deux algorithmes se comporte d'une manière quasi linéaire , mais l'écart du temps s'élargit de plus en plus que le nombre de points est grands .

4.3. temps de calcul en fonction du nombre de points pour l'algorithme de Welzl :

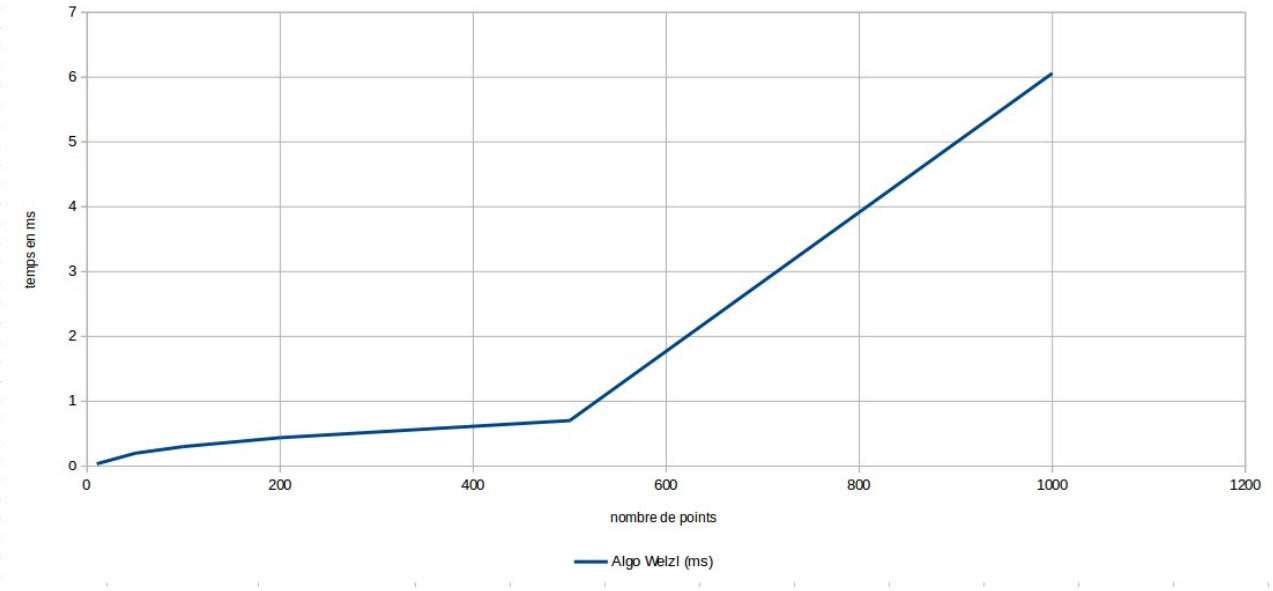


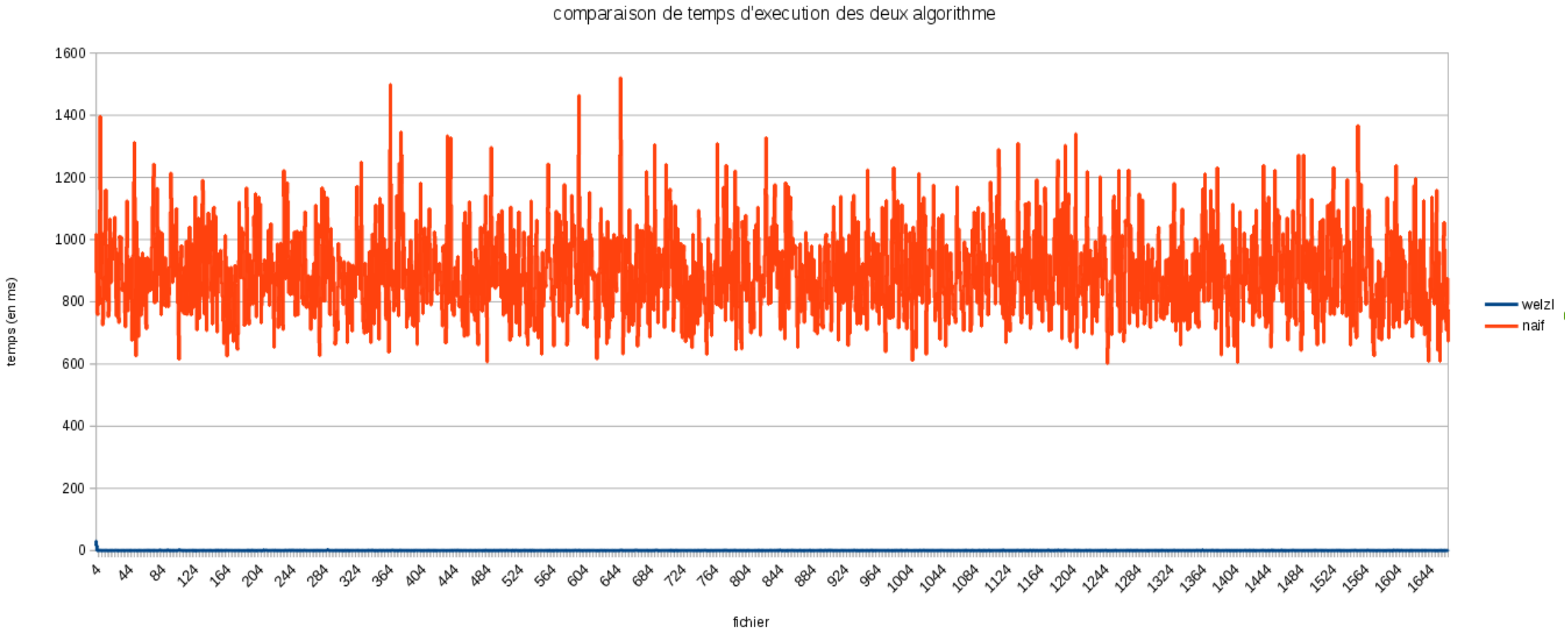
Figure 2: temps de calcul en fonction du nombre de points pour l'algorithme de Welzl

la figure 2 montre représente la variation du temps en fonction du nombre de points ; l'intervalle du temps d'exécution est en milliseconde , donc l'algorithme donne de très bonnes temps de calcul .

en constate que la courbe n'est pas complètement linaire (comparant avec sa complexité qui est en o(n)) , cela s'explique par le fait que les points sont sélectionnées d'un manière aléatoire ce qui fait que l'algorithme n'est pas toujours dans le pire cas mais il fait encore des temps nettement meilleur .

4.4. temps de calcul sur tous les fichiers de test :

dans cette section on étudier le temps d'exécution sur tout les fichiers de la base de test « Varoumas_benchmark » , pour les deux algorithme Naïf et welzl .


comparaison de temps d'execution des deux algorithme
temps (en ms)
1600
1400
1200
1000
800
600
400
200
0
4
44
84
124
164
204
244
284
324
364
404
444
484
524
564
604
644
684
724
764
804
844
884
924
964
1004
1044
1084
1124
1164
1204
1244
1284
1324
1364
1404
1444
1484
1524
1564
1604
1644
fichier
welzl
naif

La figure 3 représente le temps d’exécution en milliseconde de chaque fichiers de la base de test «Varoumas_benchmark» sur les deux algorithme naïf (en rouge) et Welzl (en bleue) .

On constate que les temps d’exécution des fichiers test sur l'algorithme naïf varie dans l'intervalle [600 , 1600] milliseconde , les valeurs proches de 600 milliseconde sont des temps d’exécution meilleurs cas ou l'algorithme ne parcours pas tout les points dans le test appartenance au cercle (containsAll) , par-contre au pire cas il peut atteindre 1600 milliseconde soit 1,6 seconde qui est un temps énorme vue le nombre de points des fichiers tests (256 points) .

Les résultats obtenues sur les fichiers tests par l'algorithme de welzl donnent des variation dans l'intervalle [1 , 40] milliseconde, cela s'explique sur le fait que chaque point est traité une seul fois c-a-d après traitement de chaque point en la supprime de la liste des points .

L’écart de temps entre l'algorithme de welzl et le naïf sont très grand qui va de 600 a 1600 milliseconde , donc l'algorithme proposé par Welzl est un bon algorithme par rapport au temps d’exécution .

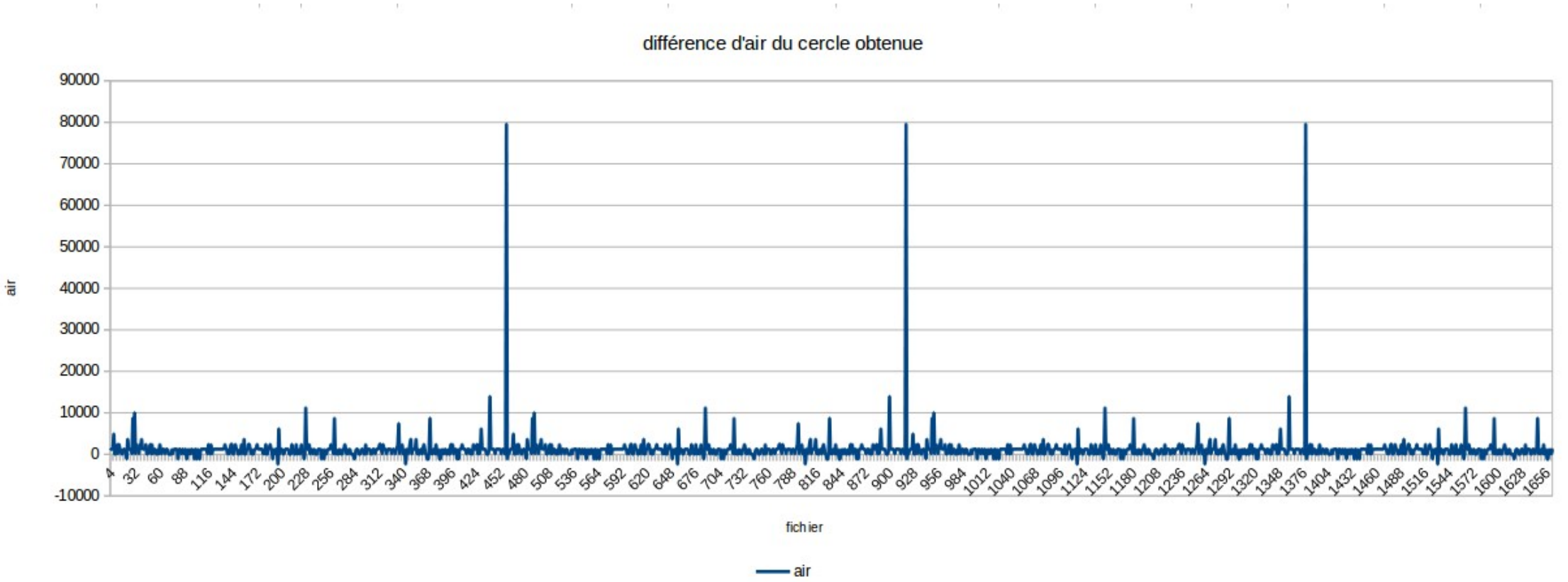
4.4. Analyse des cercles obtenus par les deux algorithmes :

dans cette partie on s'interresse au cercle résultant de chaque fichier test par les deux algorithmes naïf et Welzl , la figure suivante représente la différence de l'air du cercle obtenue par l'algorithme naïf et celui de l'algorithme Welzl .

Le calcule de l'aire du cercle est obtenues en essayant de préserver la plus grande précision possible , pour cela on utiliser le type double du langage java .

On constate qu'on générale les deux cercle obtenues par les deux algorithme ont des Aire différents et donc des diamètres différents , et cela s'explique par les approches différentes par les deux algorithmes pour résoudre le problème .

Sachant que l'algorithme naïf donne un résultant exacte , on peut conclure que l’algorithme Welzl ne donne pas toujours un résultat exacte, mais dans la majorité des cas il donne un cercle très proche du cercle résultant de l'algorithme naïf , ou même le même cercle .


différence d'air du cercle obtenue
air
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
-10000
4
32
60
88
116
144
172
200
228
256
284
312
340
368
396
424
452
480
508
536
564
592
620
648
676
704
732
760
788
816
844
872
900
928
956
984
1012
1040
1068
1096
1124
1152
1180
1208
1236
1264
1292
1320
1348
1376
1404
1432
1460
1488
1516
1544
1572
1600
1628
1656
fichier
air

5. Conclusion:

L'algorithme naïf est un algorithme qui sert essentiellement de référence pour évaluer l'algorithme de Welzl qui fait l'objet de cette étude, il permet notamment de simplifier le principe du problème du cercle minimum ,mains sont plus grand inconvénient est sa complexité au pire cas qui de l'ordre de $O(n^4)$ se qui donne de très mauvais temps de calcul .

L'algorithme proposées par Welzl est un algorithme incrémental , il se base sur le principe que chaque point de l'ensemble est soit a l'intérieur du cercle soit elle appartient au cercle minimum englobant tout les points ; sa complexité pire cas est linéaire (en O(n)) .

Les résultats expérimentaux obtenues sur la fichiers de la base de test «Varoumas_benchmark» montrent que l'algorithme Welzl est un bon algorithme vis-a-vis des temps de calcul qui sont de l'ordre du milliseconde; mais il faut prendre en compte le fait que c'est un algorithme récursif qui génère trop d'appel de fonctions , par conséquence peut saturer la pile d'appel de fonction , et ne pas fonctionner sur des trop grand fichiers .

References :

[1] : https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_cercle_minimum

*[2] : Welzl, Emo (1991), "Smallest enclosing disks (balls and ellipsoids)", in Maurer, H., New Results and New Trends in Computer Science, Lecture Notes in Computer Science **555**, Springer-Verlag, pp. 359–370,*

[3] : http://www-apr.lip6.fr/ buixuan/cpa2016