UPMC - MASTER I SPÉCIALITÉ STL

Problème du cercle minimum

Projet de Conception et Pratique Algorithmique

Victor NEA

25/02/2016

Introduction

Le problème du cercle minimum a été pour la toute première fois soulevée par Sylvester en 1857[1]. Etant donné un ensemble de points, le but est de trouver le cercle de rayon minimum, appelé cercle minimum, qui recouvre l'ensemble de tous ces points.

C'est donc à la fois un problème de géométrie mais également d'algorithmique. L'un des principaux usages de trouver ce cercle minimum est de pouvoir déterminer la boîte englobante d'une forme géométrique pour déterminer de façon simple s'il y a collision entre deux objets. Il suffit alors de vérifier si les deux centres sont ont une distance supérieure ou à la somme des deux rayons pour qu'il n'y ait pas de collision.

Il faut donc trouver un algorithme qui calcule de façon (très) rapide le cercle minimum. Imaginez qu' on joue à un jeu vidéo et qu'on doit 1 seconde à chaque fois que l'on déplace notre personnage car l'algorithme qui détecte les collisions doit à chaque fois calculer les cercles minimum, cela poserait problème. On va donc étudier deux algorithmes très différents qui permettent de calculer le cercle minimum d'un ensemble de points.

On étudiera ici que les algorithmes dans un plan à deux dimensions.

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

Notations et définitions

Soit P un ensemble de $n \in N$ points de R^2.

On notera $p(x, y)$ le point de coordonnée $(x, y) \in R^2$.

On notera $C(x, y, r)$ le cercle de centre $(x, y) \in R^2$ et de rayon $r \in R$.

On notera $C(p, q)$ (p et q deux points) le cercle de centre $m\left(\frac{p.x + q.x}{2}, \frac{p.y + q.y}{2}\right)$ et de rayon $||\overrightarrow{mp}||$.

On notera $C(p, q, r)$ (p , q et r trois points) le cercle circonscrit de ces trois points.

Illustration

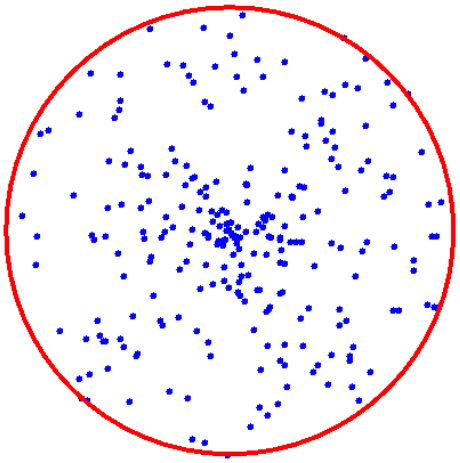


Figure 1 : Exemple d'un cercle minimum

Première approche : algorithme naïf

Pour calculer le cercle minimum, une des premières approches est d'utiliser l'algorithme naïf qui est le plus simple à comprendre[2].

Principe de l'algorithme

Si $|P| = 1$, alors il n'existe qu'un seul point p appartenant à l'ensemble P. Dans ce cas, on retourne un cercle $C(p.x, p.y, 0)$.

Si $|P| > 1$, alors :

Pour chaque paire de points (p, q) de l'ensemble P, on créé le cercle $C(p, q)$ puis on regarde si ce cercle contient tous les points de l'ensemble P. Si c'est le cas on peut choisir ce cercle comme étant le cercle minimum.

Il est cependant possible qu'après avoir testé toutes les possibilités qu'on ne trouve pas de cercle minimum.

[2] Dans le cas général...

Illustration du problème

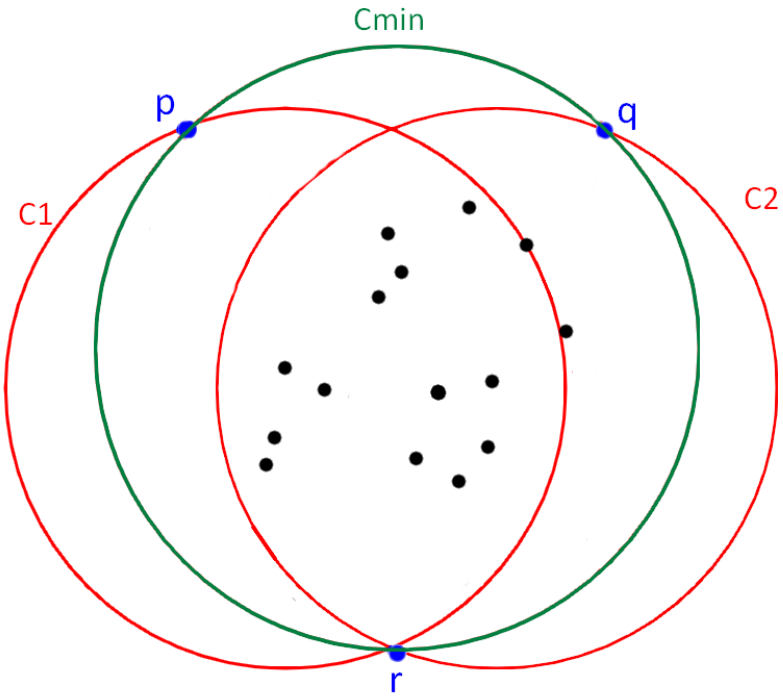


Figure 2 : Cas du triangle

A parti de l'illustration ci-dessus, on définit notre ensemble P comme étant l'ensemble des points noirs en plus des points p, q et r. $Cmin$ est le cercle minimum que l'on cherche à obtenir. On voit clairement que les distances maximales sont $||\overrightarrow{pq}||$ et $||\overrightarrow{qr}||$. Avec l'algorithme que l'on a actuellement, on aurait à un moment donné construit les cercle $C1$ et $C2$ mais l'algorithme ne les aurait pas retenu comme étant le cercle minimum vu qu'ils ne contiennent pas tous le points. Pour cela il faut donc rajouter un cas dans l'algorithme pour trouver le cercle $Cmin$ qui est en fait le cercle circonscrit des trois points p, q et r.

Il faut donc créer un cercle circonscrit $C(p,q,r)$ pour chaque triplet (p,q,r) et vérifier si celui si contient tous les points. Cependant il peux exister plusieurs cercles minimum dans ce cas là et il faut donc sélectionner le plus petit[3] d'entre eux.

De plus il ne faut pas chercher un créer le cercle circonscrit de trois points qui sont colinéaires[4], cela n'a pas de sens.

[3] Cercle ayant le plus petit rayon

[4] Pour cela on peut calculer l'aire du triangle formée par les trois points

Pseudo code

```
algorithmeNaif(P un ensemble de points)
        /* Cas d'un seul point */
        Si (||P|| = 1)
                Retourner C(P[0].x, P[0].y, 0)
        Fin Si
        /* Cas général */
        Pour chaque point p dans P
                Pour chaque point q dans P
                        Si p != q
                                C_min = C(p, q)
                                Si (C contient tous les points de P)
                                        Retourner C_min
                                Fin Si
                        Fin Si
                Fin Pour
        Fin Pour
        /* Cas du triangle */
        C_min = C(0, 0, MAX)
        Pour chaque p dans P
                Pour chaque q dans P
                        Pour chaque r dans P
                                Si (p, q et r sont différents et qu'il ne sont pas colinéaires)
                                        C_circons = C(p, q, r)
                                        Si (C_circons.r < C_min.r et C_circons contient tous les points de P)
                                                C_min = C_circons
                                        Fin Si
                                Fin Si
                        Fin Pour
                Fin Pour
        Fin Pour
        Retourner C_min
Fin algorithmeNaif
```

Complexité de algorithmeNaif

- Cas d'un seul point :

On a qu'un seul point donc $O(1)$.

- Cas général :

Deux boucles imbriquées et on vérifie si le cercle contient tous les points donc $O(n^3)$.

- Cas du triangle :

Deux boucles imbriquées et on vérifie si le cercle contient tous les points donc $O(n^4)$.

Au pire des cas, on aura donc une complexité en $O(n^4)$.

Deuxième approche : algorithme de Emo Welzl

L'algorithme de Welzl[5] est complètement différent de l'algorithme naïf. C'est un algorithme récursif et qui dit récursivité dit complication à la compréhension.

Principe de l'algorithme

Soit P un semble de n points et on note $md(P)$[6] le plus petit cercle qui contient tous les points de P. Ce cercle est composé d'au plus 3 points, points qui sont situés sur le cercle minimum.

L'idée de l'algorithme repose en deux étapes principales. Supposons que l'on se trouve à une certaine itération i $(< n)$ de l'algorithme et que l'on a déjà calculé un cercle minimum $D = md(\{p_1, \ldots, p_i\})$ de $P = \{p_1, \ldots, p_n\}$, on doit regarder si le point p_{i+1} appartient au cercle D.
Si $p_{i+1} \in D$, alors on n'a rien à faire vu que le cercle D est par conséquent toujours le cercle minimum. On peut alors regarder le point suivant p_{i+2}
Si p_{i+1} n'appartient pas à D, alors il faut mettre à jour D en prenant en compte ce dernier point.

Pour calculer ce nouveau cercle D, on dispose de l'ensemble P et l'ensemble R qui contient un ensemble de points. On suppose que P n'est pas vide et qu'il y a un point p dans P et on note $b\_md(P, R)$ le cercle minimum (à une certaine itération) contenant les deux ensembles.
Welzl a alors défini trois lemmes :

1. S'il existe un cercle contenant tous les points de P avec R étant le périmètre de P, alors on a le cercle minimum (unique).
2. Si p n'appartient pas à $b\_md(P - \{p\}, R)$, alors p se trouve sur le périmètre de $b\_md(P, R)$.
3. Si $b\_md(P, R)$ existe, alors il existe un ensemble S de $max(0, 3 - |R|)$ points dans P tel que $b\_md(P, R) = b\_md(S, R)$.

[5] Article : http://www.stsci.edu/~RAB/Backup%20Oct%2022%202011/f_3_CalculationForWFIRSTML/Bob1.pdf
[6] "md" pour "minimal disk"

Pseudo code

```
algorithmeWelzl(P un ensemble de points)
        Retourner b_minidisk(P, {})
Fin mindisk
```

```
b_minidisk(P un ensemble de points, R un ensemble de points)
        /* D non défini au début */
        D = Ø

        /* On créé directement le cercle */
         Si (P est vide ou que |P| = 3)
                D = b_md({}, R)
        Fin Si
        Sinon
                /* Choix de p */
                Choisir un point p aléatoire de P
                /* Premier appel récursif */
                D = b_minidisk(P - {p}, R)
                Si (D est défini et que p n'appartient pas à D)
                        /* Deuxième appel récursif */
                        D =  b_minidisk(P - {p}, R U {p})
                Fin Si
        Fin Sinon
        Retourner D
Fin b_minidisk
```

Complexité de algorithmeWelzl

Pour calculer la complexité de algorithmeWelzl , il faut calculer la complexité de b_minidisk.

- D non défini au début :

Complexité en $O(1)$.

- On créé directement le cercle :

Complexité en $O(1)$.

- Choix de p :

Accès direct donc $O(1)$.

- Premier appel récursif :

On retire un point p de P et à chaque fois on rappelle b_minidisk avec P qui contient donc un point en moins. On va donc finir par retomber sur le cas "On créé directement le cercle" au bout de $n = |P|$ fois. On a donc une complexité en $O(n)$.

- Deuxième appel récursif :

Même explication que précédemment.

On a donc une complexité en $O(n)$.

Résultats et comparaison des deux algorithmes

Tous les tests ont été effectués sur la base de test de Varoumas[7] composée de 1664 fichiers. Chaque fichier est composé de 256 lignes dont chaque ligne représente les coordonnées x et y d'un point. Il y a donc 256 points par fichier.

Etude sur les rayons

Avec les fichiers "res_naive_radius.txt" et "res_welzl_radius.txt" :

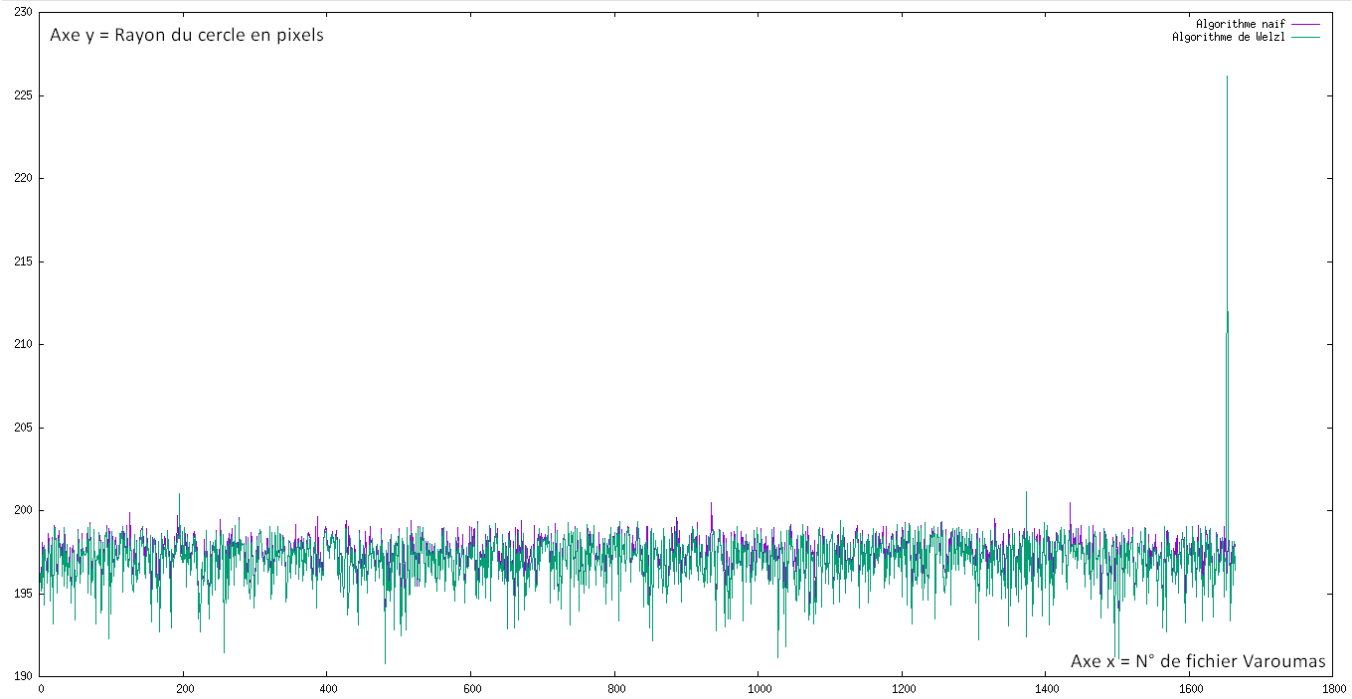


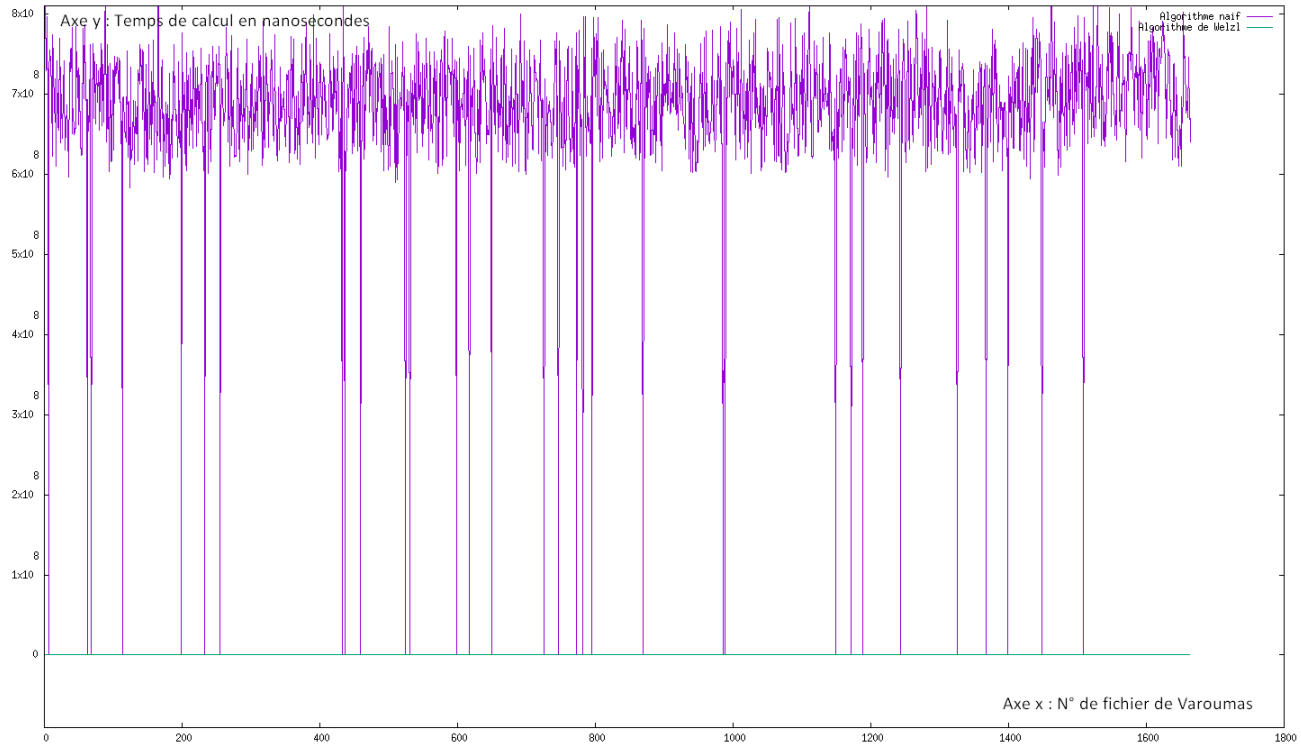
Figure 3 : Graphique de comparaison des rayons

Comme on peut le voir sur ce graphique, les rayons trouvés par les deux algorithmes sont quasi-identiques (ce qui est normal vu que le cercle minimum est censé être unique). Ce graphique ne nous donne pas d'informations pertinentes sur les performances de ces algorithmes mais ils permettent de constater que l'on a des résultats quasi-identiques et donc qu'ils sont équivalents en termes de qualités.

[7] Téléchargeable ici : http://www-apr.lip6.fr/~buixuan/files/cpa2016/Varoumas_benchmark.zip

Etude sur le temps de calcul

Avec les fichiers "res_naive_time.txt" et "res_welzl_time.txt" :



Les temps de calcul ont été mesurés en nanosecondes car comme on peut le voir sur le graphique ci-dessus, le temps de calcul est extrêmement bas (proche de 0 secondes) pour l'algorithme de Welzl (comme il n'y a que 256 points et que l'algorithme est en $O(n)$.

En revanche, pour le temps de calcul avec l'algorithme de Welz, celui prend en général un temps de $6 * 10^8 - 8 * 10^8\ ns$ soit $600 - 800\ ms$ (on tombe dans le "Cas du Triangle"). On peut également constater qu'il arrive que l'algorithme naïf se comporte parfois comme l'algorithme de Welz, à savoir qu'il prend extrêmement peu de temps à trouver le résultat. On trouve en réalité un tel résultat quand on tombe dans le "Cas général".

Avec le fichier " total_time.txt" :

L'algorithme naïf a pris $1133273238908\ ns$ soit environ $19\ min$ (= $67996394334480000\ ms$) pour faire ses calculs.

L'algorithme Welzl a seulement pris $38545908\ ns$ soit environ $38\ ms$ ($= 0{,}00064\ min$) pour faire ses calculs.

Examinons donc le gain de temps.

Avec le fichier "res_naive_vs_welzl_time.txt" :

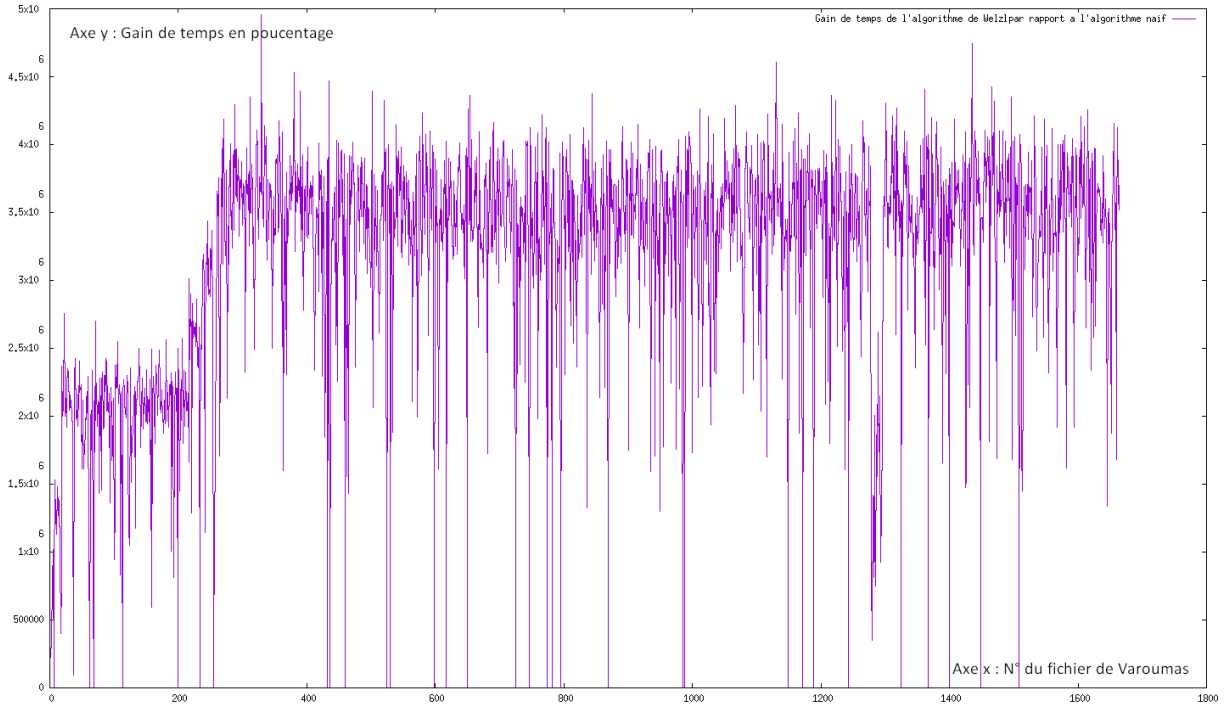


Figure 4 : Gain de temps

Comme on peut le voir sur le graphique, pour chaque fichier de teste on a un gain extrêmement important allant jusqu'à $5 * 10^6$ %.

Mais alors, quel est le meilleur algorithme ?

Et bien pour ma part il n'y a pas vraiment de meilleur algorithme. Certes on vient de voir que l'algorithme de Welzl est extrêmement rapide et qu'il nous trouve le cercle minimum en un rien de temps. Cependant ici nous avons fait des tests qu'avec des fichiers de 256 points et il faut rappeler que l'algorithme de Welzl est un algorithme récursif. Chaque appel récursif est à chaque fois empilé dans la pile. Il faut rappeler que la pile a une taille limitée dans un programme. En testant avec un nombre de points de 10 000, l'algorithme de Welzl lance une exception de type StackOverflow (dépassement de la pile). L'algorithme naïf n'aurait pas ce problème de dépassement de pile. Il trouverait bien un cercle minimum mais il faudrait attendre au moins plusieurs jours.

Pour ma part, je suis pour l'algorithme de Welz quand on cherche le cercle d'un nombre de points peu élevé. L'algorithme naïf ne devrait être jamais être utilisé. Pour un grand nombre de points, j'utiliserais par exemple l'algorithme de Ritter qui me donnerait un (faux) cercle minimum[8] de moins bonne qualité mais qui calculerait de façon extrêmement rapide pour un grand nombre de points vu que sa complexité est en $O(n)$ (comme l'algorithme de Welzl).

[8] C'est un cercle approximatif

Conclusion

On a vu deux algorithmes différents : l'algorithme naïf et l'algorithme de Welzl. L'algorithme naïf est simple à comprendre et à implémenter et l'algorithme de Welzl est plus difficile à comprendre et également plus difficile à implémenter.

On a vu qu'en terme de qualité, les deux algorithmes sont équivalents mais qu'en terme de vitesse d'exécution, l'algorithme naïf est beaucoup trop lent pour pouvoir être utilisé et que l'algorithme de Welzl ne peut être utilisé que si le nombre de points n'est pas trop élevé (nombre de points < 5000 avec 8 Go de RAM).

Il faut donc trouver des algorithmes alternatif tel que l'algorithme de Ritter qui donne un cercle de moins bonne calculer mais qui fonctionne pour un très grand nombre de points.

Informations supplémentaires

J'ai implémenté ces deux algorithmes en Java et en C++11.

L'algorithme de Welzl ne fonctionne pas en C++11 mais je n'ai pas trouvé pourquoi. Sinon pour l'algorithme naïf les calculs étaient deux fois plus rapides en C++11 qu'en Java mais j'ai quand même décidé de mesurer les temps en Java vu que l'algorithme de Welzl ne fonctionne pas en C++11 (pourtant le principe est le même...).

Vous pourrez donc lire le code en Java si vous préférez le Java ou vice-versa. Pour voir l'implémentation des deux algorithmes, il faut regarder la classe "algorithms/AlgorithmMinCircle".

Le dossier "results" contient les résultats avec lesquels j'ai tracé les graphiques.