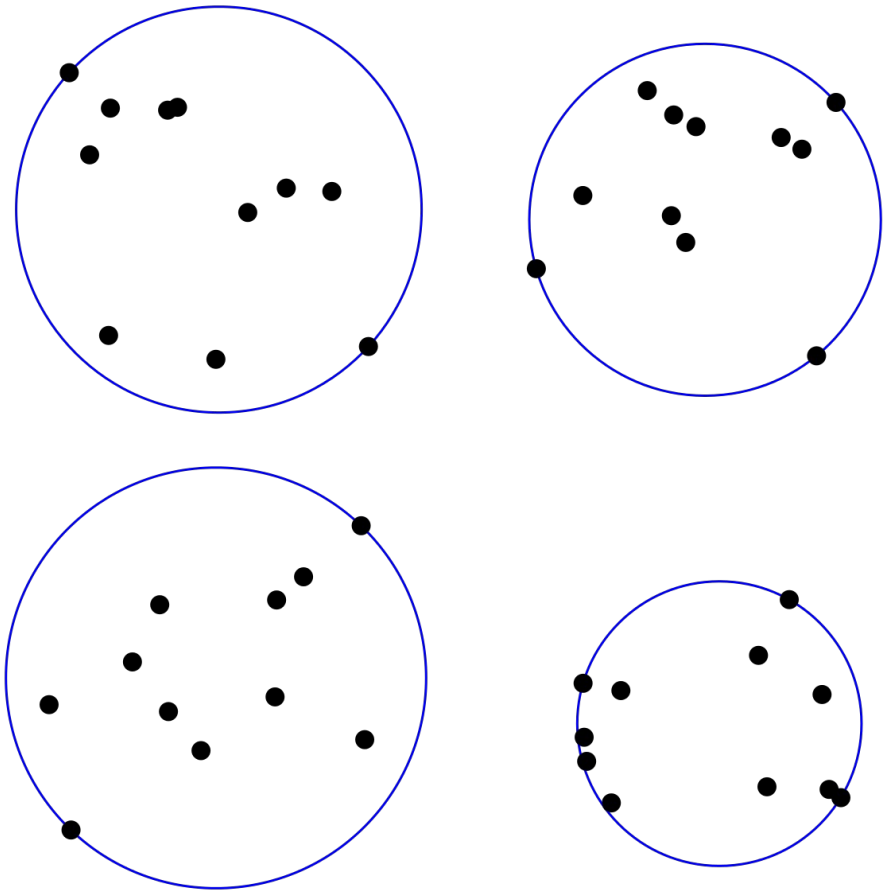
Rapport de Recherche

Algorithme de Welzl
Résolution du problème du cercle minimum



Adil Rouichi

Sommaire

Introduction

Nous avons été invités à réaliser une implémentation de l'algorithme de Welzl permettant de calculer le cercle minimum contenant une liste de points dans le cadre de notre cours de « Conception et pratique de l'algorithmique ». Ce projet a pour but d'analyser et de comprendre cet algorithme afin de pouvoir le confronter avec l'algorithme « naïf » de résolution du cercle minimum vu en cours.

Afin de pouvoir implémenter cet algorithme nous avons d'abord du comprendre son fonctionnement. En effet, nous avons été invités à nous documenter à l'aide d'un article* le présentant de manière détaillée. De plus, nous avons fait dérouler l'algorithme à la main dans le but de rendre son processus plus intelligible.

Une fois celui-ci intégré, nous avons crée un projet Java dans son intégralité pour avoir un contrôle total sur son fonctionnement. Nous avons pu créer les classes permettant de gérer la représentation des objets géométriques ainsi que celles associées à la logique algorithmique et métier.

Dans ce rapport, je vais vous présenter d'une part le contexte de ce projet ainsi que les diverses approches et concepts théoriques qui y sont associés. D'autre part, nous verrons la manière dont nous avons conçu ce projet afin de pouvoir le mener à bien. Enfin, nous exploiterons les données de tests afin d'en tirer des hypothèses et conclusions.

* http://www.stsci.edu/~RAB/Backup%20Oct%2022%202011/f_3_CalculationForWFIRSTML/Bob1.pdf

CONTEXTE

Présentation

La résolution du cercle minimum est un problème très connu en algorithmique. Il consiste à trouver, sur un plan Euclidien, le cercle de plus petit rayon permettant d'englober une série de points.

Dans la pratique, il permet de trouver l'emplacement optimal (le centre du cercle) d'une ressource nécessitant d'être à une certaine distance de plusieurs entités. Par exemple, une antenne radio ayant un rayon d'émission doit être positionnée de manière stratégique vis à vis des installations susceptibles de recevoir ses ondes.

Cas triviaux

Il existe plusieurs cas triviaux pour lesquels sa résolution est évidente, qui sont notamment utilisés dans les algorithmes plus complexes :

-**Cas trivial n°1 - Un seul point** : Le cercle de rayon minimum est celui qui a pour centre les coordonnées du point et pour rayon 0.

-**Cas trivial n°2 – Deux points** : Le cercle de rayon minimum a pour centre le milieu du segment entre ces deux points et pour rayon la moitié de la taille de ce segment.

-**Cas trivial n°3 – Trois points** : Le cercle de rayon minimum est le cercle circonscrit au triangle formé par ces trois points.

Il existe plusieurs approches pour résoudre ce problème. En effet, nous avons pu mettre en place deux méthodes différentes qui sont celles dîtes « naïve » qui repose sur une recherche exhaustive et celle de l'algorithme de Welzl qui est un algorithme de recherche récursif faisant appel à la propriété du recouvrement.

Méthode « naïve »

La méthode naïve repose sur le principe simple et peu optimisé de tester toutes les combinaisons de 2 points (**cas trivial n°2**), puis, si l'on ne trouve pas de solution permettant d'englober tous les points, de tester toutes les combinaisons de 3 points (**cas trivial n°3**).

L'algorithme est défini et implémenté comme ci-suit :

1. Pour tout *p* dans *Points* :
2. Pour tout *q* dans *Points* :
3. *cercle* = cercle de centre (*p*+*q*)/2 de diamètre |*pq*| => MinCercle(p,q)
4. si *cercle* englobe tous les points :
5. alors retourner *cercle*
6. *cercle* = cercle de rayon infini
7. pour tout *p* dans *Points* :
8. pour tout *q* dans *Points* :
9. pour tout *r* dans *Points* :
10. *cercleCourant* = cercle circonscrit au triangle formé par *p*, *q* et *r (MinCercle(p,q,r))*
11. si *cercleCourant* englobe tous les points et son rayon plus petit que *cercle* :
12. alors *cercle* = *cercleCourant*
13. retourner *cercle*

Cette méthode est très contraignante car elle est peu performante. En effet, sa complexité est en $O(n^4)$. On pourra notamment le constater dans la phase de test. De plus, elle ne trouve pas forcément le cercle minimum car elle est susceptible de trouver un cercle à l'aide de deux points qui englobe tous les points du plan. Ainsi, elle le considéra comme le cercle de rayon minimum même si il y en a un plus optimal.

Algorithme de Welzl

L’algorithme de Welzl est un algorithme récursif permettant de résoudre le problème du cercle minimum avec la solution optimale en complexité O(n). Il repose sur trois principes fondamentaux permettant sa grande performance par rapport à l’algorithme naïf.

D’une part, en effectuant des appels récursifs et en excluant les points déjà enclos par un cercle minimum, la taille des ensembles sur lesquels il calcule le cercle minimum diminue et ainsi le nombre d’opérations nécessaire à sa résolution également.

D’autre part, il utilise la propriété du recouvrement qui permet d’éviter les calculs redondants. En effet, une fois un point exclu et le cercle du sous ensemble crée, nous pouvons tout simplement vérifier si le point de l’ensemble englobant se situe dans le cercle, ce qui évite de recalculer le cercle minimum pour ce point et ainsi de suite.

Enfin, la part d’aléatoire dans cet algorithme entraine une certaine imprévisibilité dans le fonctionnement de cet algorithme et ses performances mais permet de diversifier la sélection des points et ainsi permet de répartir la complexité de cet algorithme de manière équilibrée. Ainsi, cela peut permettre de rendre l’algorithme plus performant sur un nombre d’exécution et/ou de points élevés.

```
1. function WELZL (P, R) :
2.     si P est vide ou R contient trois points :
3.         Cercle = MinCercle(R)
4.     sinon :
5.         p = point aléatoire appartenant à P
6.         Cercle = WELZL(P – p, R)
7.         si Cercle est défini et p n’est pas dans Cercle :
8.             Cercle = WELZL(P – p, R – p)
9.     retourner Cercle
```

Le premier appel à la fonction WELZL doit s’effectuer avec P l’ensemble de points du plan Euclidien et R un ensemble vide.

IMPLÉMENTATION

Description

Nous avons décidé d'implémenter notre algorithme en Java afin de pouvoir s'inspirer des classes et des méthodes vues en cours et également car c'était conseillé par notre enseignant. De plus, nous sommes à l'aise avec l'utilisation de classes qui permettent de séparer le code afin de fournir une meilleure lisibilité et organisation lors de son élaboration.

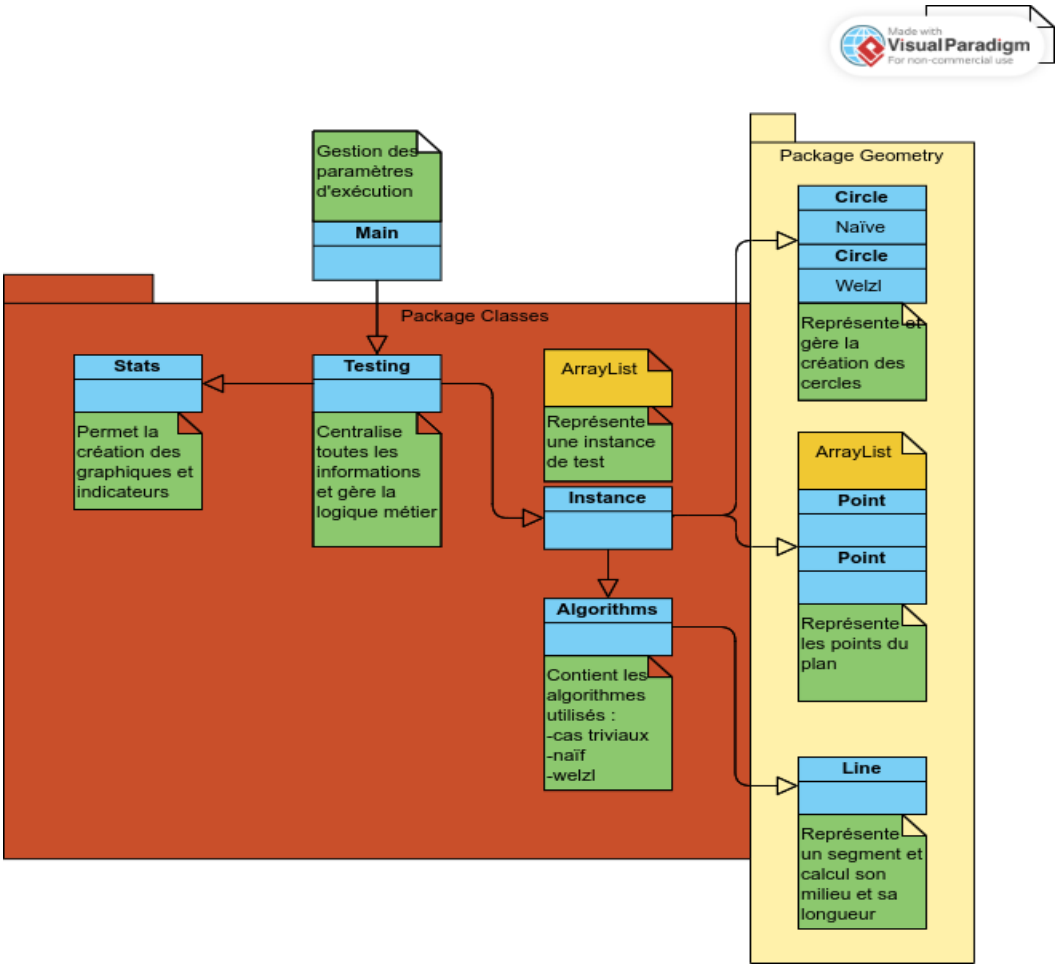


Diagramme de classes du projet

Notre classe Main permet de charger les paramètres d'exécution qui malheureusement, par faute de temps ne peux pas être exécuté en ligne de commande et fournits dans le args []. Ainsi il faut modifier les valeurs des variables à la main dans la classe (**cf.readme.txt**).

Ensuite, la classe Testing va charger tous les instances de test et créer des objets « Instance » pour chacune d'elle en chargeant les points associés.

Chaque instance va effectuer son calcul du cercle minimum via l'algorithme naïf et l'algorithme de Welzl en calculant son temps d'exécution, son nombre d'itérations et ses cercles respectifs via la classe statique « Algorithms ».

Une fois cela fait, la classe « Testing » va transmettre sa liste d'objets « Instance » à la classe « Stats » qui va calculer tous les indicateurs et créer les graphiques afin de les afficher sous la forme d'un dashboard.

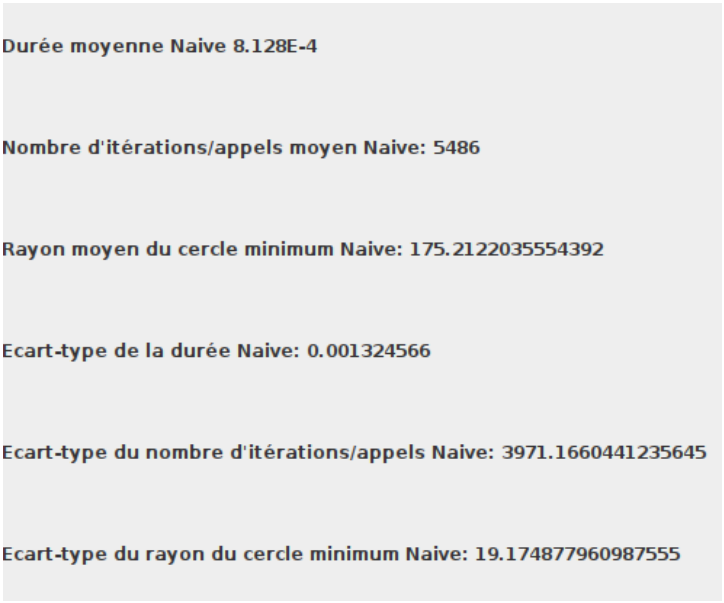
Tests

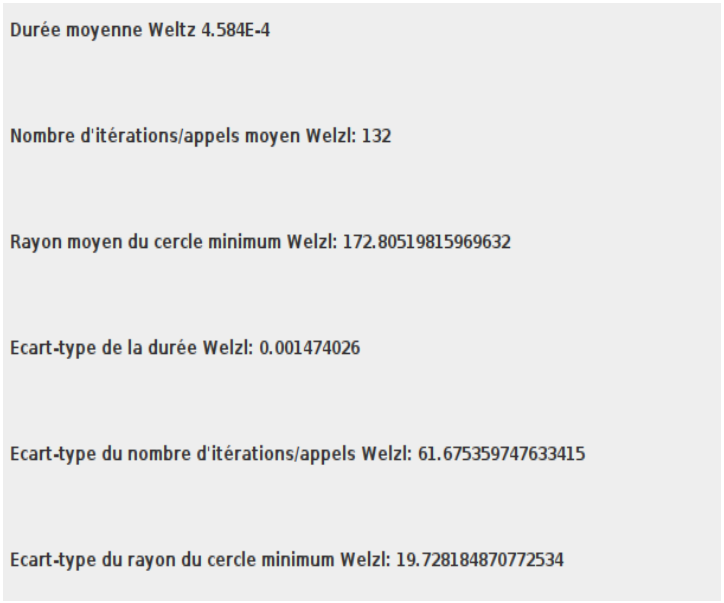
Nous avons effectuer les tests sur les instances 1664 instances de tests contenant chacune 255 points et disponibles ici : http://www-apr.lip6.fr/~buixuan/files/ado22cpa/Varoumas_benchmark.zip.

Nous avons décidé de présenter de deux tests. Un premier test avec des instances de 20 points et un deuxième avec des instances de 200 points afin de mettre en lumière la perte de performance de l'algorithme naïf du à sa complexité vis à vis de l'algorithme de Welzl.

Je tiens à préciser qu'une anomalie apparaît sur les valeurs de tests de la première instance, sûrement due à une surcharge (processeur et mémoire) résultant de la compilation et de l'exécution du programme son mon ordinateur.

Test avec 20 instances de 20 points :



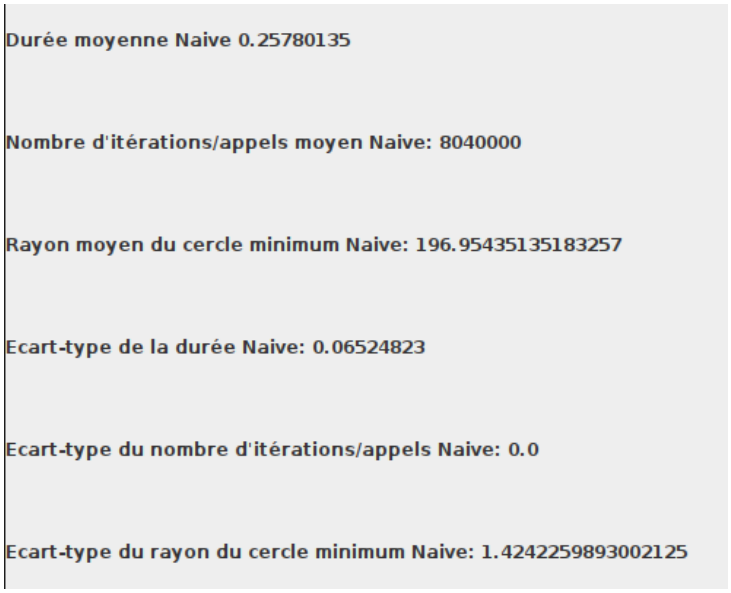


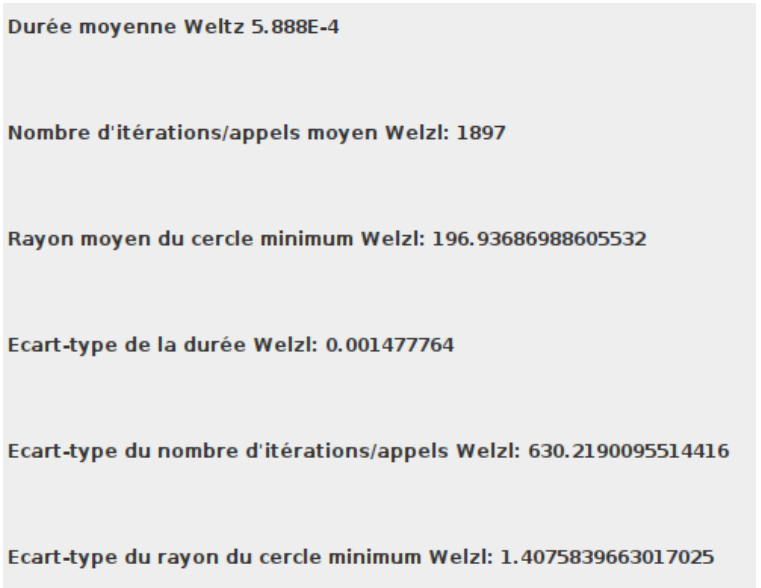
On peut tout d'abord remarquer que la durée moyenne de calcul avec l'algorithme de Welzl est déjà deux fois plus petite que celle de l'algorithme naïf avec un faible nombre de point. De plus, son nombre d'appels/itérations est lui aussi plus faible ce qui est étroitement lié à sa durée d'exécution.

On peut également constater que le rayon moyen du cercle minimum est plus grand pour l'algorithme naïf que pour l'algorithme de Welzl. En effet, cela peut s'expliquer par le fait que dans certains cas, l'algorithme naïf retourne un cercle formé à l'aide deux points qui n'est pas le cercle optimal. Ainsi, cela peut gonfler la taille du rayon moyen de son cercle minimum par rapport à celui de Welzl.

Enfin, on observe un fort écart type du nombre d'appel pour l'algorithme naïf par rapport au deuxième, ce qui témoigne de sa régularité moindre.

Test avec 20 instances de 200 points :



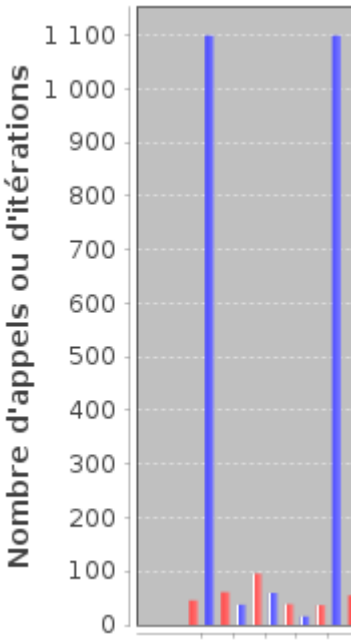


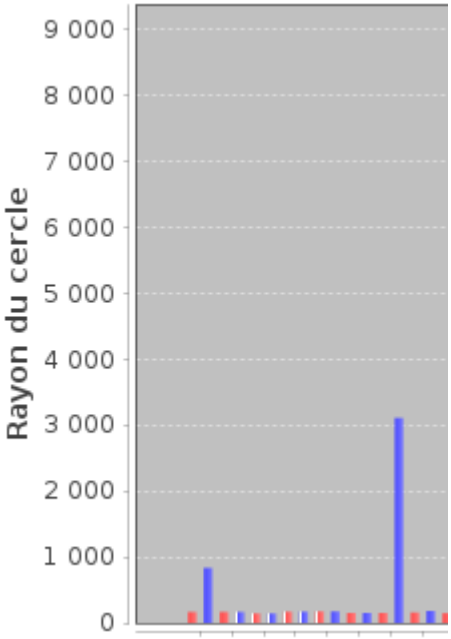
Avec un nombre beaucoup plus élevé de points, on se rends vite compte que l'algorithme naïf est nettement moins performant que l'algorithme de Welzl.

En effet, la durée moyenne est environ 1000 fois plus grande pour l'algorithme naïf. De plus, son écart-type est également plus élevé. Cela montre une fois de plus que l'algorithme de Welzl fait preuve d'une plus forte régularité. Il en va de même pour la moyenne du nombre d'itérations/appels des deux algorithmes ce qui est étroitement lié avec les indicateurs précédents.

Enfin on peut voir grâce à l'écart type du nombre d'itérations de l'algorithme naïf que désormais, il est obligé d'aller au bout de l'algorithme car il ne trouve plus de cercle minimum formé avec seulement deux points lorsque leur nombre est élevé. C'est tout du moins le cas pour cette barrière de tests avec 20 instances mais il est tout à fait possible que cela arrive.

Nombre d’appels et rayon du cercle minimum (20 points par instance)



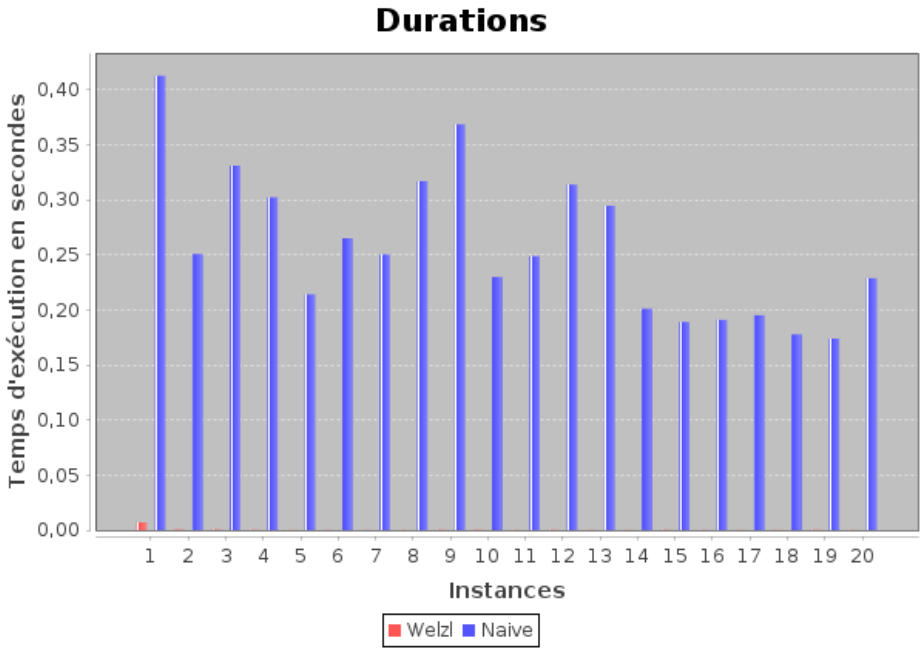


En bleu : Algorithme naïf **En rouge : Algorithme Welzl**

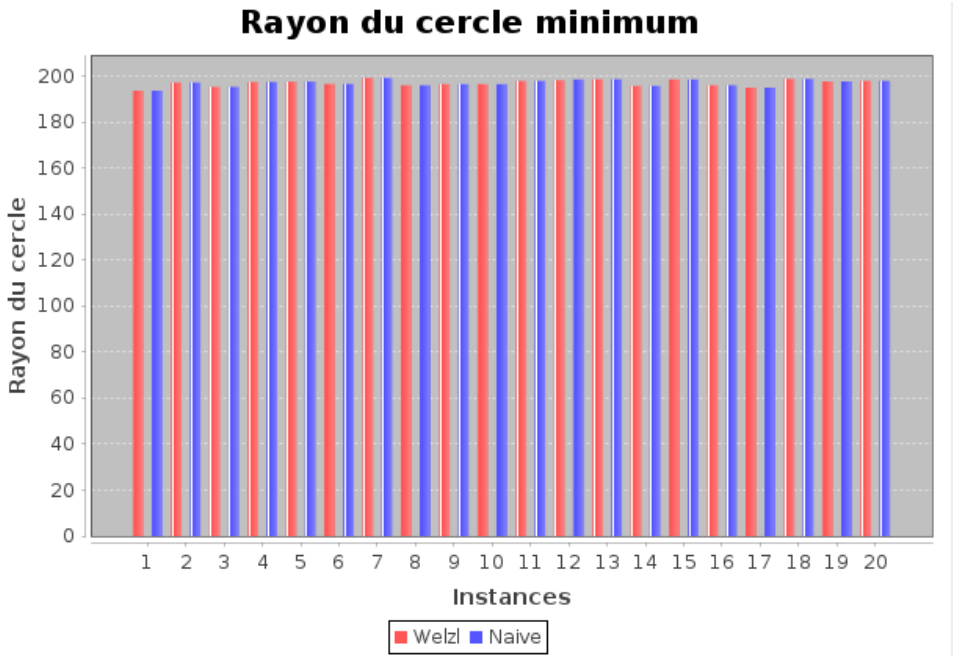
On peut voir sur le premier graphique du nombre d’appel que l’algorithme naïf n’est pas régulier (ce qui est le cas pour un faible de points) alors que l’algorithme de Welzl reste plus ou moins constant.

De plus, sur le deuxième graphique représentant la taille du rayon du cercle minimum, on peut voir que pour un même lot de points, l’algorithme de Welzl trouvera toujours le cercle de rayon minimal alors que dans certains cas (le cas des deux points), l’algorithme naïf ne le trouvera pas.

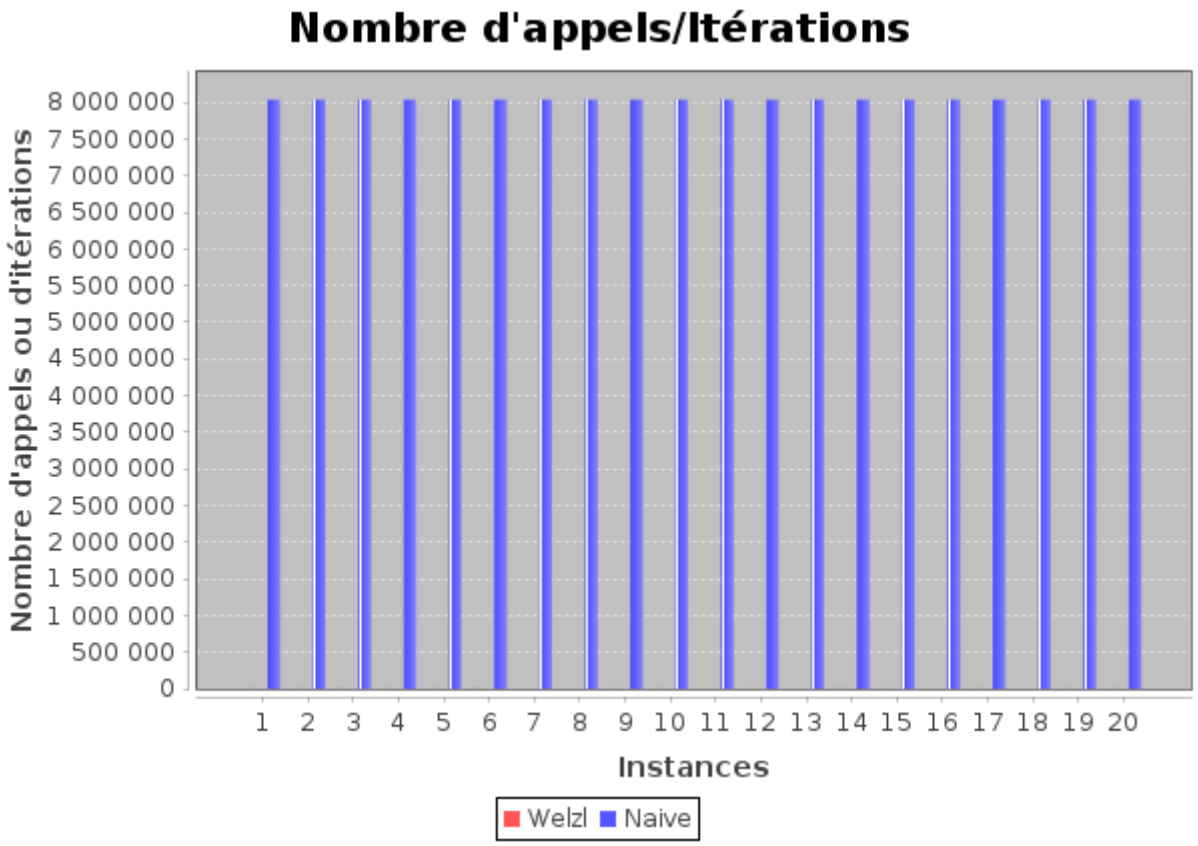
Les trois graphiques suivants concernent 20 instances de 200 points



Encore une fois, pour le temps d'exécution, on peut voir que l'algorithme naïf est nettement moins performant que celui de Welzl qu'on peut très vaguement apercevoir sur la première instance sachant que cette contre performance est une anomalie.



Pour ce qui est du rayon du cercle, il est sensiblement le même sachant que l'algorithme naïf teste toutes les combinaisons possibles de 3 points.



Enfin, pour le nombre d'itérations/appels, l'algorithme naïf est stable pour les mêmes raisons que mentionnées sur le graphique précédent et le delta entre sa quantité d'itérations/appels et celle de l'algorithme de Welzl est tellement grand qu'on ne peut plus apercevoir la sienne graphiquement.

CONCLUSION

Pour conclure, je suis satisfait du travail réalisé. En effet, les résultats de nos tests se rapprochent de l'appréhension théorique de ces deux algorithmes. En effet, on peut remarquer que l'algorithme de Welzl est nettement plus performant que l'algorithme naïf. Cette observation est d'autant plus vraie lorsque l'on augmente le nombre de points et d'instances, ce qui peut confirmer sa complexité en O(n) et celle de l'algorithme naïf en $O(n^4)$. De plus, au-delà de cette performance, l'algorithme de Welzl garantit la meilleure solution et fait preuve d'une régularité plus élevée.

Même si nous pouvons nous rendre compte de l'augmentation de son efficacité proportionnelle à celle du nombre de points, j'aurais aimé que l'on puisse réaliser des graphiques à l'aide des valeurs globales (moyennes, écart-types) de batteries de tests contenant un nombre de point différent afin d'observer ces comportements de manière plus précise.

Je suis déçu de ne pas avoir eu le temps de mettre en place un Makefile ainsi que tout le processus de compilation permettant de pouvoir exécuter ce projet en ligne de commande en fournissant des paramètres.

La perspective de pouvoir réfléchir à la projection de ce problème sur un espace Euclidien me semble des plus intéressante mais, encore une fois, faute de temps, nous n'avons pas pu approfondir ce sujet. J'espère avoir l'occasion de me pencher sur cette question à l'avenir.