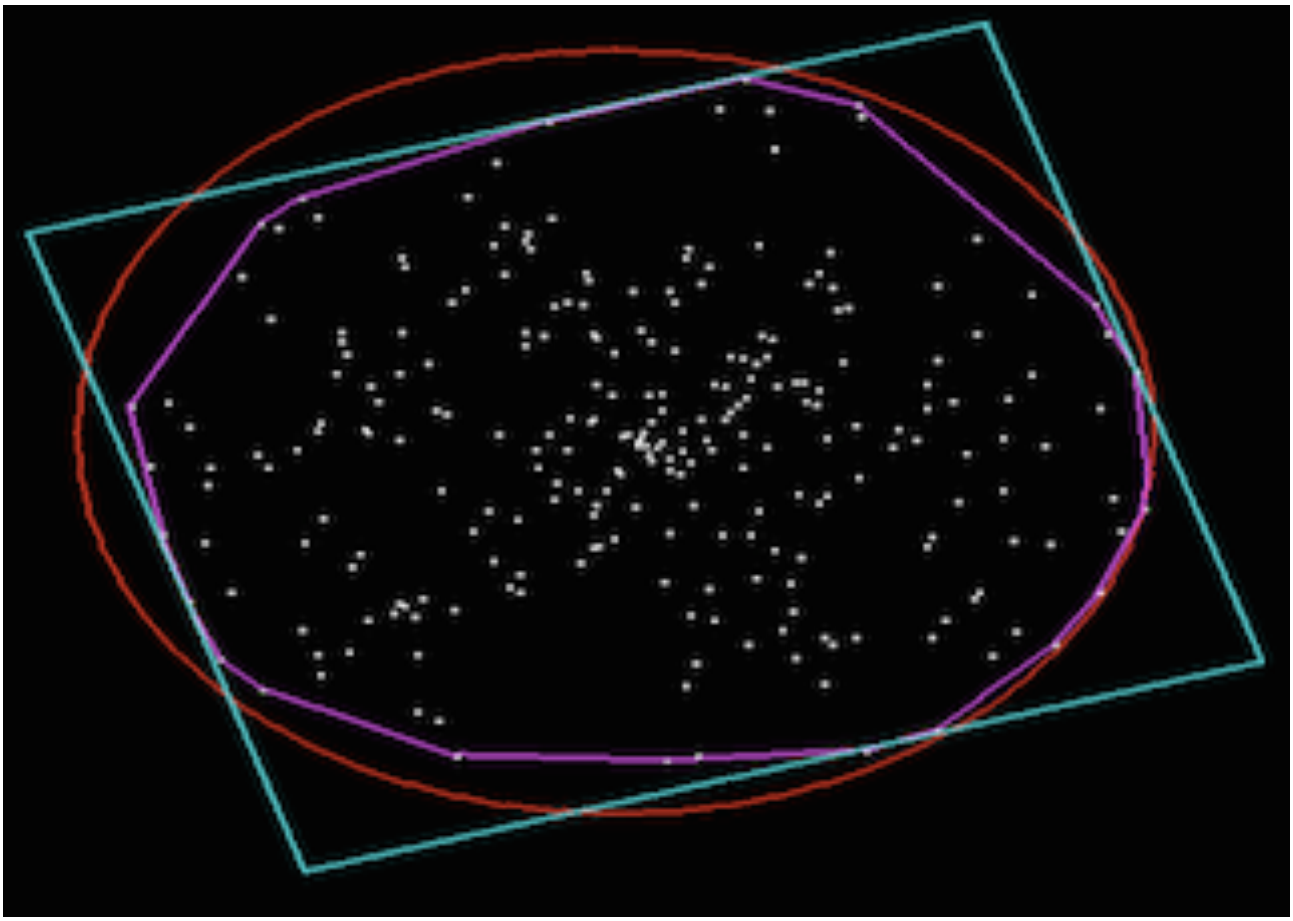
Rapport projet 1 CPA

Problème du rectangle minimum dans un nuage de point - Algorithme Toussaint

Problème du cercle minimum dans un nuage de point - Algorithme Ritter

AFFES Mohamed Amin «3262731» - 29 mars 2015



Problématique

Nous avons un nuage de points fourni, et voulons calculer le rectangle minimum englobant tout ces points, ce problème est un classique en algorithmique plusieurs implémentations sont possibles cependant le but ici est de faire une implémentation qui soit acceptable en terme de complexité en temps car l'écart entre une solution naïf pour ce problème et une autre optimale ou proche de l'optimale peut être très grand et donc à prendre en considération car ceci peux amer de très grands gains, surtout si l'algorithme est exécuté sur des milliers voir des millions de données. Ici nous avons une base de test constitué de plusieurs fichiers pour tester l'efficacité de l'algorithme proposé.

La deuxième problématique est le calcul du cercle minimum contenant un nuage de point pour ceci nous utilisons l'algorithme Ritter qui donne un résultat très satisfaisant en terme de vitesse malgré le fait qu'il ne soit pas optimal pour les raisons cités dans la suite de ce rapport.

1-Introduction

Nous avons cités la problématique ci dessus par rapport à l'algorithme Toussaint et avons souligné le fait que l'algorithme implémentée doit avoir une complexité meilleure qu'un algorithme naïf pour ce problème.

Pour résoudre ce problème j'implémente l'algorithme Toussaint, cet algorithme a plusieurs variantes ayant des complexités différentes comme j'ai dit cette différence n'est pas à prendre à la légère par exemple après calcul de l'enveloppe convexe pour calculer le rectangle minimum pour un nuage de point en utilisant les points de l'enveloppe convexe, pour m points de l'enveloppe on trouve un algorithme Toussaint trouvant le rectangle minimum en O(m2) et un autre la trouvant en O(m) (au cours de mes recherches j'ai lu aussi qu'une solution en ayant une complexité logarithmique existe aussi) donc nous constatons que cette différence joue beaucoup sur la complexité de notre algorithme car pour une très grosse masse de données l'algorithme ayant la première complexité

peux prendre un temps beaucoup plus long que le deuxième pour finalement donner le même résultat.
J'ai essayé d'implémenter la première solution ayant une plus grande complexité je vais donc détailler les démarches suivies pour l'implémentation de cet algorithme puis par la suite parler de la solution que j'ai implémentée au final et que je vous ai rendu en code Java. La première solution je ne l'ai pas implémentée en code mais je la détaille juste afin de comparer avec la méthode ayant une meilleure complexité que j'ai implémentée.

Remarque:

Pour tester le code vous avez à votre disposition la classe Test.java dans le package test. Si vous appelez la méthode Test() de l'instance f dans le main Ritter, Graham et Toussaint sont lancées sur tous les fichiers de la base de test la qualité pour chaque fichier est affiché, le rectangle minimum sur tous les rectangles minimum de tous les fichiers est affiché à la fin dans une fenêtre graphique, les qualités moyennes sont aussi affichés sur la console.

la méthode Testfile(i) de l'instance f effectue un affichage graphique correspondant au rectangle minimum et enveloppe convexe et cercle minimum du fichier i, vous pouvez appuyez sur le touche 'e' de votre clavier pour passer au fichier suivant.

2-Définitions et Formules

Dans cette section, je définit quelques définitions et formules mathématiques permettant de comprendre l'implémentation de mon algorithme afin de pouvoir bien comprendre l'implémentation de cet algorithme citée dans la suite.

- Un cercle est une courbe plane fermée constituée des points situés à égale distance d'un point nommé centre. La valeur de cette distance est appelée rayon du cercle. Celui-ci étant infiniment variable, il existe donc une infinité de cercles pour un centre quelconque, dans chacun des plans de l'espace.

La classe Circle dans le package tools joue ce rôle dans ce projet.

- Une ligne ici est une droite théoriquement infini tracé ayant un Point contenue dans cette droite et un vecteur directeur qui est un point sur un plan 2D ayant donc deux coordonnés et définissant le sens de cette droite dans une plan 2D.

La classe Ligne dans le package tools joue ce rôle dans ce projet.

- Un Segment est une ligne tracé reliant deux points dans un plan.

La classe Segment dans le package tools joue ce rôle dans ce projet elle contient deux attributs a et b représentant les deux points.

- Un Rectangle est un quadrilatère dont les quatre angles sont droits, ici un rectangle est constitué de quatre segments chaque segment se croise avec un autre segment dans chacun de ses points.

La classe Rectangle dans le package tools joue ce rôle dans ce projet.

- La distance entre deux points dans un plan 2D :

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}.$$

- Le produit vectoriel entre un segment constitué de deux points ‘p’ et ‘q’ et un point ‘r’:

pv = ((q.x - p.x) * (r.y - p.y)) - ((q.y - p.y) * (r.x - p.x)).

La méthode produit_vectoriel dans la classe Graham.java dans le package algorithms implémente cette formule.

- Le calcul d’un angle entre deux droites (ici deux instances de la classe Ligne) :

soit v1 le vecteur directeur de la droite 1 et v2 le vecteur directeur de la droite 2, le résultat est donc :

angle = |atan(((v1.y * v2.x - v2.y * v1.x) / (v1.x * v2.x + v1.y * v2.y)))|

- La rotation d'une droite d'un certain angle s'effectue comme suit :

Soit v le vecteur directeur de la droite, angle l'angle de rotation et deux valeurs décimales 'x' et 'y', et 'vn' le nouveau vecteur directeur de la droite après la rotation.

x = (v.x * cos(angle)) + (v.y * sin(angle)).
y = (-(v.x * sin(angle))) + (v.y * cos(angle)).
vn = Un nouveau point de coordonnée x et y.

- L'intersection entre deux droites l1 et l2 :

Soit la valeur décimale 't' et et les points 'q', 'p', 'a', 's', 'r', 'n' avec n le point d'intersection entre les deux droites:

q = un point de coordonnées (l1.PointDroite.x, l1.PointDroite.y).
p = un point de coordonnées (l2.PointDroite.x, l2.PointDroite.y).
a = un point de coordonnées (q.x - p.x, q.y - p.y).
s = le vecteur directeur de l1.
r = le vecteur directeur de l2.
t = (a.x * s.y - a.y * s.x) / (r.x * s.y - r.y * s.x).

n = nouveau point de coordonnées ((p.x + (t * r.x)) , (p.y + (t * r.y))).

- Aire Polygone :

Soit A un nombre décimal il est égal à la somme pour j allant de 0 a (Taille enveloppe -1)
A += ((elemEnveloppe(j).x * elemEnveloppe(j + 1).y) - (elemEnveloppe(j + 1).x * elemEnveloppe(j).y)).
Aire Polygone = A / 2.

- Aire Rectangle = longueur * largeur.
- Aire cercle = PI * Rayon^2.
- qualité Rectangle = (Aire rectangle / Aire Polygone) - 100%.
- qualité Cercle = (Aire cercle / Aire Polygone) - 100%.

3-Algorithme

Calcul de l'enveloppe convexe - Algorithme Graham :

Cet algorithme commence par une phase de pré calcul ou on prend pour un ensemble de points ayant la même abscisse dans notre nuage de point le point en lui même s'il est unique, sinon le point d'ordonné minimal et la point d'ordonné maximal. On parcours notre ensemble de point et on remplis deux tableaux une contenant les points d'ordonnés max et l'autre les points d'ordonnés min comme cité au dessus. Ensuite on ajoute le tout dans une liste de sort a avoir les points **dans un sens anti-trigonométrique**.

Ensuite on re parcourt la liste et dans l'itération i pour un segment constitué des points i et ((i+1) % taille enveloppe) de l'enveloppe si le résultat de son calcul vectoriel avec le point ((i+2) % taille enveloppe) est négatif (c'est a dire que le point ((i+1) % taille enveloppe) est inutile car on peux relier directement i et (i+2) % taille enveloppe) qui couvrent ce point) on supprime le point ((i+1) % taille enveloppe) afin du supprimer les points inutiles et avoir une enveloppe couvrant tous les points du nuage et ayant le minimum de cotés possibles.

La complexité de ce calcul est en **O(n)**.

Calcul du rectangle minimum - Algorithme Toussaint :

- Tout d'abord on lance le calcul de l'enveloppe convexe sur notre nuage de points en utilisant l'algorithme Graham cité au dessus.

Ensuite, dans la façon de faire numéro 1 (la mauvaise celle qui parcours l'enveloppe en 2 boucles imbriqués):

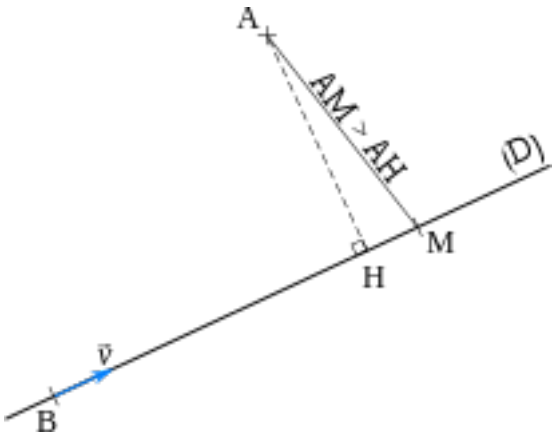
- Parcourir l'enveloppe dans une première boucle «on appelle l'itération actuelle dans la première boucle l'itération 'i' , celle dans une deuxième boucle imbriqué l'itération 'j' ».
- Construire un segment constitué du point i et du point ((i+1) % taille enveloppe).

- Re parcourir l'enveloppe dans une deuxième boucle imbriqué afin d'appliquer une formule mathématique pour retrouver les coordonnés des points orthogonaux à la droite superposé sur le segments ce qui nous donne la longueur du coté du rectangle superposé à ce segment, ensuite on calcule la distance entre la droite et les points de l'enveloppe pour trouver le point le plus éloigné de la droite tout en étant orthogonal à celle ci ensuite on construit un segment ayant le même vecteur directeur du segment actuel dans ce point puis on construit les deux coté qui restent en reliant les deux extrémités des deux segments (on décale les deux extrémités du premier segments en utilisant le vecteur allant de la première droite au point trouvé de la deuxième droite «le plus lointain de la droite actuelle »).
- On calcul l'air du rectangle et remplace la valeur de l'air minimal si l'air du rectangle de l'itération 'i' actuelle est inférieur à la valeur min enregistré «au départ une valeur très grande ».
- On passe à l'itération suivante de la boucle à index 'i' jusque parcourir toute l'enveloppe.

Voici les formules concernant le projeté orthogonal sur une droite dont j'ai parlé dans le paragraphe précédent :

$$\overline{\mathrm{BH}} = \frac{(x_\mathrm{A} - x_\mathrm{B})x_v + (y_\mathrm{A} - y_\mathrm{B})y_v}{\sqrt{x_v^2 + y_v^2}}$$

$$\begin{cases} x_\mathrm{H} = x_\mathrm{B} + \dfrac{\overline{\mathrm{BH}}}{\sqrt{x_v^2 + y_v^2}} x_v \\ y_\mathrm{H} = y_\mathrm{B} + \dfrac{\overline{\mathrm{BH}}}{\sqrt{x_v^2 + y_v^2}} y_v \end{cases}$$



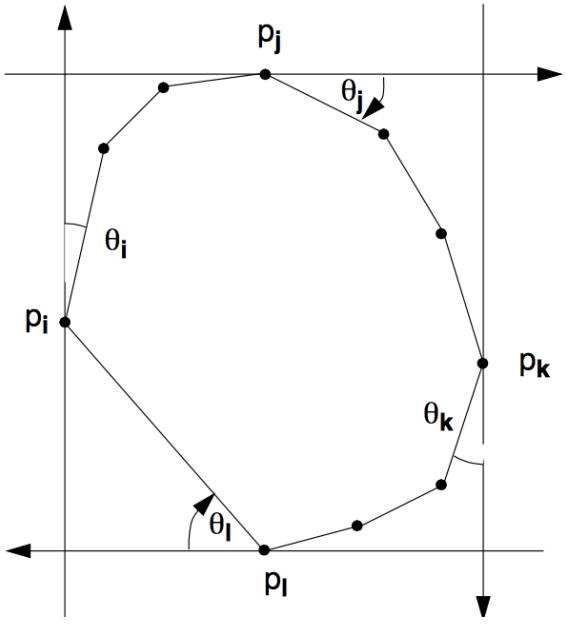
Vous comprenez peut être maintenant pourquoi je n'ai pas choisi cette solution je me suis effectivement rendu compte qu'elle était très couteuse (si on prend m le nombre de points de l'enveloppe convexe cet algorithme est en **O(m^2)**) du fait qu'elle contient de nombreux calculs demandant plusieurs parcours de l'enveloppe au sein de la première boucle la parcourant (et oui c'est bien moche).

Ce pourquoi j'ai choisi d'implémenter une autre façon de faire moins couteuse que je décrit dans le paragraphe qui suit :

- Parcourir l'enveloppe pour stocker les coordonnés de 4 points, le point le plus au nord 'Pleft', celui le plus à l'est 'Pright', celui le à l'ouest 'Ptop' et enfin celui le plus au sud 'Pbottom' de notre enveloppe. On stocke aussi l'ordonné min et maxi et l'abscisse min et max.
- Construire quatre droites ayant comme point les points construits à l'étape précédente et ayant comme vecteur directeur v(0, -1) pour Pleft on appelle la droite 'leftV', v(0, 1) pour Pright on appelle la droite 'rightV', v(1, 0) pour Ptop on appelle la droite 'topH', v(-1, 0) pour Pbottom on appelle la droite 'bottomH'.
- Calculer les points d'intersections entre :
  (topH, leftV) ; (bottomH, leftV) ; (topH, rightV) ; (bottomH, rightV).
  Puis on construit le rectangle construit à partir des quatre points résultants et on le prend comme rectangle minimal et son Aire comme l'air du rectangle minimal.

A partir d'ici on entre dans une boucle qui parcours l'enveloppe, on appelle l'itération courante de la boucle l'itération 'j'.

- On calcule les quatre angles indiqués sur l'image à droite.



Pour calculer l'angle nous avons besoin des quatre droites précédentes, des quatre points précédents et du point suivant pour chacun de ces quatre derniers points. La façon de faire est indiquée dans la section précédente qui spécifiait les formules mathématiques utilisées.

- Puis on prend l'angle minimal des quatre qu'on appelle 'angleMin'.

- Puis on effectue une rotation pour nos quatre droites de (-anglesMin) la valeur négatif est du au fait que notre rotation est dans le sens anti-trigonométrique.

- Puis On recherche les quatre points d'intersection des quatre nouvelles droites comme précédemment et on construit le rectangle résultant puis calculons son air et remplaçons la valeur de airMin si air < airMin et dans ce cas enregistrons le rectangle courant comme étant rectangle minimal aussi.
- Puis on prend le point parmi nos quatre point correspondant au point contenue dans la droite superposé à l'angle minimal pris dans cette itération et le remplaçons par le point qui le suit dans l'enveloppe, nous le modifions aussi dans la droite qui lui correspond.

Ici on sort de la boucle parcourant l'enveloppe.

- Arrivant à cette étape Airmin contient l'air du rectangle minimal et le rectangle minimal est enregistré, nous retournons ce dernier.

Cet algorithme ne fait donc comme vous pouvez remarquer aucun parcours des points de l'enveloppe et donc aucune boucle au sein de la boucle principale parcourant l'enveloppe, la boucle ne contient que des calculs ne coutant quasiment rien à la machine.

Si l'on prend donc m la taille de l'enveloppe convexe la complexité de cet algorithme est en **O(m)**. Ce qui est bien meilleur que celui de la stratégie précédente nous ne pouvons voir la grande différence entre les deux complexité sur notre jeux de test mais sur un autre beaucoup plus conséquent cette différence peux être plus visible.

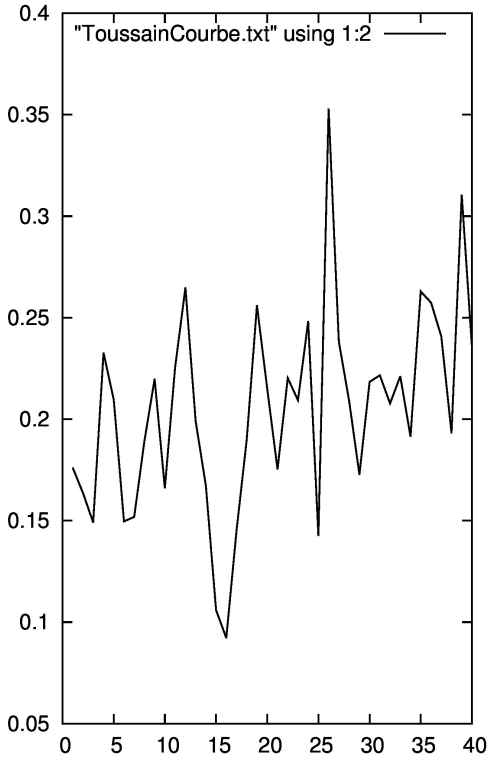
La complexité totale donc Graham + Toussaint est de l'ordre de **O(n+m)** pour n le nombre de points du nuage de points et m le nombre de points de l'enveloppe convexe.

la qualité moyenne (somme qualité pour chaque fichier / nombre de fichiers) est égale à 0.20264 soit de l'ordre de 20% ce qui est tout à fait normal car pour qu'un rectangle puisse couvrir un rectangle convexe il faut forcément que son air soit plus grand et que l'écart soit de cette importance, j'ai essayé d'améliorer cette qualité au maximum dans mon algorithme en prenant des points de coordonnées décimales et non entières pour le vecteur directeur ainsi que pour les points du rectangles et des segments et dans mes calculs pour avoir la meilleur exactitude possible cependant je pense que le résultat reste

peut être quand même assez approximatifs compte tenu de la difficulté d'avoir une précision optimale.

Voici une courbe représentant les différentes qualités calculés sur différents fichiers choisis aléatoirement, l'axe y représente la qualité calculé la courbe a été faite après calcul sur 40 fichiers distincts:

On peux voir sur cet exemple que la qualité moyenne tourne autour de la valeur 0.2 ce qui correspond au calcul précédent effectué, nous pouvons aussi remarquer que sur certains fichier cette valeur baisse et atteint même pour un fichier moins de 0.1 ! ce qui est une bonne qualité donc on peux constater que la qualité varie en fonctions de l'ensemble des points donnée et donc peut être bonne proche de 0.1 comme elle peux être beaucoup moins bonne et atteindre 0.37 environs pour d'autres fichier cependant la qualité moyenne reste acceptable et pas très lointaine de la meilleur valeur trouvé.



Calcul du cercle minimum - Algorithme Ritter :

Je précise tout d'abord que le résultat est approximatif, le résultat est dégradé afin d'avoir un temps de calcul plus rapide qu'un algorithme naïf. Le résultat n'est pas tout de même pas très dégradé comme on peux le voir dans le graphe indiquant la qualité calculé sur quelques tests fait sur quelques fichier à la fin de cette section.

- Prendre le premier point dummy du nuage de points (ou autre point quelconque).
- Trouver le point ‘P’ de distance maximum du point dummy en parcourant la liste de points une première fois puis re parcourir cette liste de points pour trouver le point ‘Q’ de distance maximum au point ‘P.’
- Créer un point ‘c’ se trouvant au centre du segment ‘PQ’.
- Prendre la distance ‘Pc’ comme valeur de la variable ‘rayon’ représentant le rayon du cercle.

A partir d’ici on boucle tant que la liste n’est pas vide.

- Parcourir l’ensemble des points et supprimer les points se trouvant dans le cercle donc ayant une distance avec ‘c’ le centre du cercle < à la variable ‘rayon’.
- S’il reste des points dans l’ensemble de points on prend le point ’s’ qui est le premier élément de cette liste (ou autre point quelconque).
- On calcule ensuite les points ‘a2’ et ‘b2’ qui sont les coordonnés du nouveau centre du cercle comme suit :

cs = la distance ‘cs’.
cps = (le rayon + cs) / 2.
ccp = cs - cps.

a = cps / cs.
b = ccp / cs.

a2 = (a * c.x) + (b * s.x).
b2 = (a * c.y) + (b * s.y).

- Ensuite on calcule le nouveau rayon égal à ((ancien rayon + distance ‘cs’)/2) et on supprime le point ’s’ puis on re boucle jusque au moment ou la liste se vide.

On retourne en sortie un cercle de centre ‘c’ et de rayon ‘rayon’.

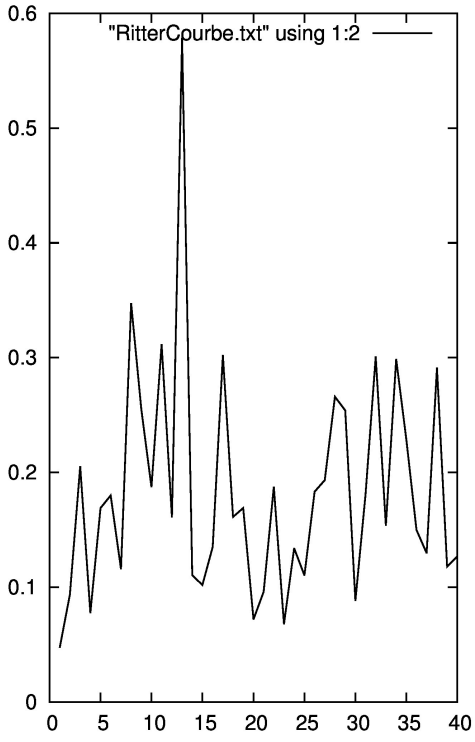
Nous pouvons remarquer dans cet algorithme un deuxième point causant un manque léger de précision (très minime) qui est le cast en entier au niveau de l’affection de la valeur du rayon au cercle final.

La complexité de cet algorithme est en O(n) donc pas très couteux.

La qualité moyenne est de 0.17804757959324974 ce qui est un bon résultat quant au fait que notre algorithme est une approximation pour avoir un gain de temps mais en remarque que cette approximation permet quand même d’avoir une qualité assez bonne même si elle n’est pas optimale.

Voici une courbe représentant les différentes qualités calculés sur différents fichiers choisis aléatoirement, l’axe y représente la qualité calculé la courbe a été faite après calcul sur 40 fichiers distincts:

On peux voir sur cet exemple que la qualité moyenne tourne autour à un peu moins de 0.2 environs donc 0.7 ce qui correspond au calcul précédent effectué, nous pouvons aussi remarquer que sur certains fichier cette valeur baisse et atteint même pour un fichier moins de 0.05 ! ce qui est une bonne qualité donc on peux constater que comme pour l’algorithme Toussaint ici aussi la qualité varie en fonctions de l’ensemble des points donnée et donc peut être bonne proche de 0.05 comme elle peux être beaucoup moins bonne et atteindre 0.57 environs pour d’autres fichier cependant la qualité moyenne reste acceptable et pas très lointaine de la meilleur valeur trouvé.



4-Conclusion

A la fin de cette présentation on peux tout d'abord remarquer l'importance d'avoir un algorithme optimal ou proche de l'optimal ayant une bonne complexité car ceci est un paramètre d'une très grande importance quand il est question de grands projets et d'application d'un algorithmes sur des données conséquentes en terme de taille des données. Ensuite on remarque aussi qu'un problème a souvent plusieurs solutions et souvent la première qui vient à l'esprit est la solution naïve il faut donc prendre le temps de bien réfléchir à l'algorithme à implémenter afin de ne pas juste faire un algorithme qui marche mais un algorithme aussi efficace ayant une complexité raisonnable.
Enfin ces algorithmes peuvent peut être encore être optimisés car la qualité moyenne n'est pas optimale l'existence d'une meilleur solution peut donc être possible.

4-Sources

Dans cette section je présente certains liens qui m'ont servis à retrouver quelques formules mathématiques ou quelques indications pour l'implémentation de l'algorithme.

Calcul de l'angle entre deux droites :
http://integraledesmaths.free.fr/idm/PagePrincipale.htm#http://integraledesmaths.free.fr/idm/GeoAPAngDro.htm

Calcul du point d'intersection de deux segments:
http://openclassrooms.com/forum/sujet/calcul-du-point-d-intersection-de-deux-segments-21661

Rotation vectorielle:
http://fr.wikipedia.org/wiki/Rotation_vectorielle

Toussaint:
http://web.cs.swarthmore.edu/~adanner/cs97/s08/pdf/calipers.pdf

https://geidav.wordpress.com/2014/01/23/computing-oriented-minimum-bounding-boxes-in-2d/