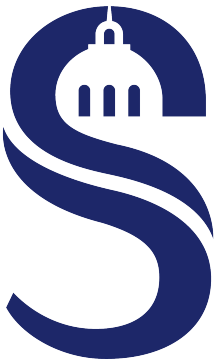

SORBONNE
UNIVERSITÉ

Calcul du cercle minimum d'un nuage de points à partir de l'algorithme de Welzl

Vivien Demeulenaere

Remis le 20 Mars 2022

Calcul du cercle minimum d'un nuage de points à partir de l'algorithme de Welzl

Résumé : Étant donné un ensemble fini $\mathcal{P}$ de points de $\mathbb{N}^2$, le cercle minimum est le plus petit cercle contenant tous les points de $\mathcal{P}$. On propose alors de mettre en oeuvre l'algorithme récursif imaginé par Welzl. On veillera également à dérécursifier cet algorithme pour le rendre utilisable en pratique et ne pas craindre de dépassement de pile.

Mots-clés : Cercle minimum, Dérécursification, Géométrie algorithmique, Welzl

1 Introduction

Étant donné un ensemble fini $\mathcal{P}$ de points de $\mathbb{N}^2$, le cercle minimum est le plus petit cercle contenant tous les points de $\mathcal{P}$.
Le calcul d'un tel cercle peut s'avérer utile dans de nombreux domaines comme le jeu-vidéo notamment, pour approcher des objets par des cercles. En effet, il est très facile d'évaluer si une collision a lieu entre deux cercles $\mathcal{C}_1$ et $\mathcal{C}_2$ de centre respectif c_1 et c_2. Il suffit alors de vérifier si la distance entre c_1 et c_2 est supérieure à la somme des rayons des deux cercles, auquel cas il n'y aura pas de collision.
Cette méthode permet donc de ne faire qu'un seul calcul plutôt que de vérifier que chacun des points n'est pas dans les deux cercles, ce qui n'est pas négligeable surtout si les calculs doivent être effectués en temps réel.
C'est donc en partie pour cette raison que de nombreux algorithmes permettant de calculer le cercle minimal d'un nuage de points ont été mis en place. On peut notamment citer l'algorithme de Welzl qui sera l'objet de notre étude.

2 Notations utilisées et définitions formelles

Nous noterons $\mathcal{P}$ l'ensemble contenant n points où $n \in \mathbb{N}$, on notera la taille de $\mathcal{P}$ de la manière suivante : $|\mathcal{P}| = n$.
Nous travaillerons avec des points de la forme $p = (x, y)$ où x et y représentent respectivement l'abscisse et l'ordonnée de p.
La distance euclidienne entre deux points p et q sera notée $\delta(p, q)$.
Les cercles seront représentés par un point définissant leur centre ainsi que par leur rayon.
Plus formellement, ce rayon est défini par $max_{p \in \mathcal{C}} \delta(c, p)$ où c est le centre du cercle $\mathcal{C}$.
Le cercle ne contenant aucun point sera noté $\emptyset$. Il aura alors un rayon égal à zéro.
De plus, on parlera de cercle circonscrit au triangle prq, le cercle passant par les trois sommets de ce triangle.
Enfin, les lemmes suivant seront utilisés :

Lemme 1 : Si $n = 0$, alors le cercle minimum de l'ensemble $\mathcal{P}$ sera $\emptyset$.

Lemme 2 : Si $n = 1$, alors le cercle minimum de l'ensemble $\mathcal{P}$ sera le cercle ayant pour centre le seul point de $\mathcal{P}$ et pour rayon 0.

Lemme 3 : Si $n = 2$, alors le cercle minimum de l'ensemble $\mathcal{P}$ sera le cercle ayant pour centre le milieu de ces deux points et pour rayon la distance entre le centre et un de ces points.

Lemme 4 : Si $n = 3$, alors le cercle minimum de l'ensemble $\mathcal{P}$ sera le cercle circonscrit au triangle pqr où $(p, q, r) \in \mathcal{P}^3$ et $p \neq q \neq r$.

3 Calcul du cercle minimum par approche naïve

3.1 Principe de l'algorithme

Un algorithme facile à implanter consiste à regarder tous les points $(p, q) \in \mathcal{P}^2$ tels que $p \neq q$, de calculer le cercle dont le centre c se situe sur le segment $[pq]$ et vérifier si tous les points de $\mathcal{P}$ sont bien dans le cercle. Cet exemple est illustré sur la figure 1. On a ainsi un algorithme avec une complexité en temps de $\mathcal{O}(n^3)$ avec n le nombre de points de $\mathcal{P}$. En effet, il faut itérer sur les n points n fois pour vérifier toutes les combinaisons possibles. De plus, il faut vérifier pour chacun des n points s'ils sont contenus dans le cercle.

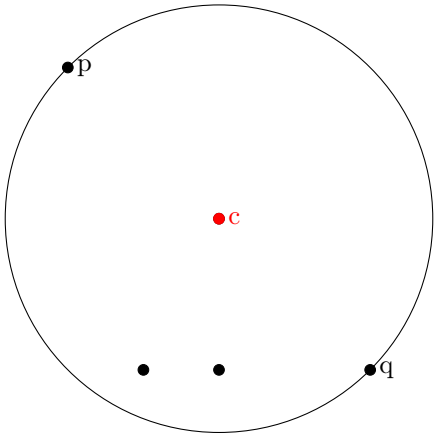

Figure 1 – Cas sans triangle équilatéral

Cependant, si les deux points les plus éloignés forment un triangle équilatéral avec un autre, cet algorithme ne fonctionne plus. On peut d'ailleurs le constater sur la figure 2.
Une solution à ce problème serait de prendre trois points différents p, q et r dans $\mathcal{P}$, de calculer le cercle circonscrit au triangle pqr et vérifier si tous les points de $\mathcal{P}$ appartiennent à ce cercle.
Cette fois-ci on obtient un algorithme avec une complexité en temps en $\mathcal{O}(n^4)$ puisqu'il faut effectuer des combinaisons de trois points et non plus deux.
Ainsi, en combinant les deux algorithmes décrits plus haut, on obtient un moyen de calculer le cercle minimum de l'ensemble de points $\mathcal{P}$ de manière exacte.

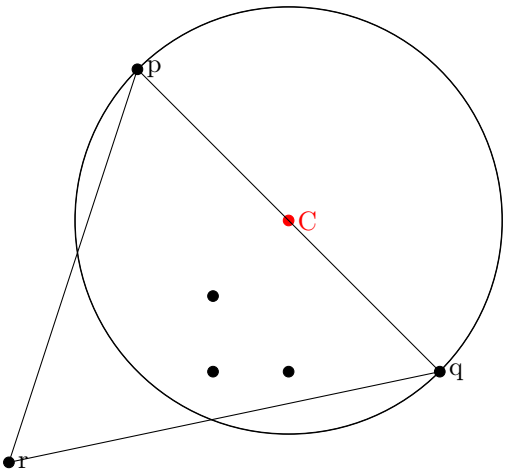

Figure 2 – Cas avec triangle équilatéral

3.2 Pseudo-code

Voici un pseudo-code de l'algorithme 1 décrit ci-dessus. Une version implantée en Java est également disponible dans le code source fourni avec le rapport. Elle se situe dans la classe `Algorithms.algorithms.java`.

input : Un ensemble de points $\mathcal{P}$
output : Un cercle minimum *resultat* englobant les points de $\mathcal{P}$

```
1  forall p ∈ P do
2  |  forall q ∈ P do
3  |  |  c ← milieu du segment [pq] ;
4  |  |  cercle ← cercle de centre c et diamètre δ(p, q) ;
5  |  |  if cercle contient tous les points de P then
6  |  |  |  return cercle
7  |  |  end
8  |  end
9  end
10 resultat ← cercle de rayon infini ;
11 forall p ∈ P do
12 |  forall q ∈ P do
13 |  |  forall q ∈ P do
14 |  |  |  cercle ← cercle circonscrit au triangle pqr ;
15 |  |  |  if cercle contient tous les points de P et cercle plus petit que resultat then
16 |  |  |  |  resultat ← cercle ;
17 |  |  |  end
18 |  |  end
19 |  end
20 end
21 return resultat ;
```

Algorithme 1 : Algorithme naïf

4 Calcul du cercle minimum à l'aide de l'algorithme de Welzl

4.1 Principe de l'algorithme

L'algorithme précédent ayant une complexité temporelle assez élevée, on se propose d'implémenter l'algorithme proposé par Emo Welzl. Il s'agit d'un algorithme récursif. L'idée consiste à utiliser deux ensembles de points : $\mathcal{P}$, l'ensemble de départ et $\mathcal{R}$ les points que l'on a traité. Ainsi, initialement, $\mathcal{R} = \emptyset$.

4.1.1 Cas de bases

L'idée est donc de s'arrêter dès que $\mathcal{P} = \emptyset$ ou bien dès que $|\mathcal{R}| = 3$.
En effet, il est possible de créer un cercle unique de taille minimum à partir de trois points d'après le lemme 4 (cf. section 2). En revanche, si c'est $\mathcal{P}$ qui est vide, alors on utilisera les lemmes 1 à 3 de la section 2 pour créer un cercle minimum.

4.1.2 Appels récursifs

Si nous ne sommes dans aucun des cas de bases définis précédemment, on retire aléatoirement un point p dans $\mathcal{P}$. On appelle ensuite l'algorithme avec la nouvelle valeur de $\mathcal{P}$. Noton $\mathcal{C}$ le résultat de cet appel. Si $p \in \mathcal{C}$, alors $\mathcal{C}$ est le résultat. Sinon, on ajoute p à $\mathcal{R}$ et le résultat est alors l'appel à l'algorithme avec les nouvelles valeurs de $\mathcal{P}$ et $\mathcal{R}$.

4.2 Pseudo-code

Pour définir le pseudo-code de cet algorithme, on utilisera un algorithme principal (cf algorithme 2) et un algorithme auxiliaire (cf algorithme 3). On supposera également que les cercles créés lorsque l'on arrive au cas de base, le seront en suivant les principes des lemmes définis à la section 2.
Une version implantée en Java est également disponible dans le code source fourni avec le rapport. Elle se situe dans la classe `Algorithms.algorithms.java`.

input : Un ensemble de points $\mathcal{P}$
output : Un cercle minimum $\mathcal{C}$ englobant les points de $\mathcal{P}$

1 **return** *b_ minidisk(*$\mathcal{P}$*,* $\emptyset$*)*

Algorithme 2 : Algorithme de minidisk

input : Un ensemble de points $\mathcal{P}$ et un ensemble de points contenus dans le cercle minimum $\mathcal{R}$

output : Un cercle minimum $\mathcal{C}$ englobant les points de $\mathcal{R}$

```
1  if P = ∅ or |R| = 3 then
2  |  C ← un cercle minimum contenant tous les points de R
3  else
4  |  p ← un point choisi aléatoirement de P ;
5  |  C ← b_minidisk(P − {p}, R) ;
6  |  if p ∉ C then
7  |  |  C ← b_minidisk(P − {p}, R ∪ {p}) ;
8  |  end
9  end
10 return C
```

Algorithme 3 : Algorithme de minidisk

4.3 Complexité

Chaque point p dans $\mathcal{P}$ a une chance de $\frac{1}{|\mathcal{P}|}$ d'être choisi. Notons $t(n)$ et $c(n)$ le nombre d'étapes respectif des algorithmes 2 et 3 où $n = |\mathcal{P}|$. On notera également l le temps de calcul du cercle minimum avec un ensemble de trois points au maximum.
Ainsi, on obtient l'inégalité suivante :

$$t(n) \leq l + t(n-1) + Prob(p \notin b\_minidisk(\mathcal{P} - \{p\})).c(n-1)$$

Puisqu'il y a au plus trois points dans $\mathcal{R}$, on peut en déduire la majoration suivante :

$$Prob(p \notin b\_minidisk(\mathcal{P} - \{p\})) \leq \frac{3}{n}$$

donc

$$t(n) \leq (l + 3c)n$$

On en conclu alors que l'algorithme 2 a une complexité temporelle en $\mathcal{O}(n)$.

5 Dérécursification de l'algorithme de Welzl

Si l'algorithme de Welzl a une complexité bien moindre que l'algorithme naïf, il n'est pas pour autant meilleur en pratique. En effet, le caractère récursif de l'implantation peut assez vite engendrer des dépassements de capacité de pile.
Afin de pallier ce problème, on propose une version itérative de l'algorithme précédent.

5.1 Principe de l'algorithme

Afin de dérécursifier l'algorithme 2, on utilise un tableau de booléens de la taille de $\mathcal{P}$. Celui-ci nous permet de savoir si l'on doit effectuer un deuxième appel récursif correspondant à la ligne 7 de la l'algorithme 2. De plus, on utilise un booléen pour vérifier si l'on doit faire l'appel de la ligne 5. Ensuite, les variables sont similaires à la version récursive, avec notamment l'ensemble $\mathcal{R}$ initialement vide. Il est à noté qu'il faudra tout de même retenir le nombre de points de $\mathcal{P}$ traités et ceux dans $\mathcal{R}$. Le principe est donc le suivant : on mélange les points de $\mathcal{P}$ afin de simuler le tirage au sort aléatoire de la ligne 4, puis on itère jusqu'à avoir traiter tous les points de $\mathcal{P}$.
L'itération est séparée en deux partie : soit l'on simule l'appel récursif de la ligne 5, soit celui de la ligne 7. Si l'on se trouve dans la première situation, on créé le cercle minimum contenant les points de $\mathcal{R}$ si l'on est sur le cas d'arrêt (*i.e.* si tous les points de $\mathcal{P}$ ont été traités ou si $\mathcal{R}$ contient trois points). Sinon, il est nécessaire de simuler un nouvel appel avec un point de moins dans $\mathcal{P}$.
Si l'on ne doit pas effectuer l'appel de la ligne 5, alors deux solutions sont possibles : soit l'on a l'appel de la ligne 7 à simuler et dans ce cas on met le dernier point de $\mathcal{P}$ en première position, il faudra alors effectuer un deuxième appel pour tous les points déjà traités. Soit l'on est déjà dans le cas d'un deuxième appel et on ajoute le point courant à $\mathcal{R}$ si ce dernier ne fait pas parti de $\mathcal{C}$.
Ainsi, on garde une complexité linéaire tout en se détachant du caractère récursif de l'algorithme 2.

5.2 Complexité

La complexité en temps de cet algorithme est la même que pour sa version récursive. Cependant, il est à noté qu'un peu plus de mémoire doit être allouée, notamment pour maintenir n booléens dans un tableau. Ceci ne fait pas réellement changer la complexité en espace qui était déjà en $\mathcal{O}(n)$ en raison des n points à traiter.

5.3 Pseudo-code

Vous trouverez sur la page suivante un pseudo-code de l'algorithme 4 décrit ci-dessus. Une version implantée en Java est également disponible dans le code source fourni avec le rapport. Elle se situe dans la classe `Algorithms.algorithms.java`.
Il est tout de même à noter que pour occuper moins de mémoire, la version Java remplace le tableau de booléen par un `BitSet`. En effet, un bit est suffisant pour décrire un état soit vrai soit faux.

input : Un ensemble de points $\mathcal{P}$

output : Un cercle minimum $\mathcal{C}$ englobant les points de $\mathcal{P}$

```
1  mélanger P ;
2  R ← ∅ nbPoints_p ← 0 ;
3  nbPoints_r ← 0 ;
4  appel ← vrai ;
5  secondAppel ← tableau de |P| booléens initialisés à faux ;
6  C ← ∅ ;
7  do
8      if appel then
9          if nbPoints_p = 0 or nbPoints_r = 3 then
10             C ← cercle minimum contenant tous les points de R;
11             nbPoints_p ← nbPoints_p + 1;
12             appel ← faux ;
13         else
14             nbPoints_p ← nbPoints_p − 1;
15             appel ← faux ;
16         end
17     else
18         retourSecondAppel ← secondAppel[nbPoints_p] ;
19         if not retourSecondAppel then
20             p ← P_{nbPoints_p} ;
21         else
22             if p ∈ C then
23                 nbPoints_p ← nbPoints_p + 1 ;
24             else
25                 R_{nbPoints_r} ← p ;
26                 nbPoints_r ← nbPoints_r + 1 ;
27                 secondAppel[nbPoints_p] ← vrai ;
28                 appel ← vrai ;
29             end
30             forall i, 0 < i < nbPoints_p + 1 do
31                 secondAppel[i] ← faux ;
32             end
33             mettre le dernier point de P en première position ;
34             nbPoints_r ← nbPoints_r − 1 ;
35             nbPoints_p ← nbPoints_p + 1 ;
36             appel ← faux ;
37         end
38     end
39 while nbPoints_p ≤ |P|;
40 return C
```

Algorithme 4 : Algorithme de Welzl itératif

6 Test des implantations

Afin de tester les différents algorithmes présentés plus haut, on utilise une base de 1663 tests contenant chacun 256 points. Cette base de test peut être trouvée ici. On cherche notamment à comparer leur temps d'exécution.
Il est à noté que des tests sur des ensembles de points plus grands pourraient conduire à une erreur de dépassement de capacité de pile en raison du caractère récursif de l'algorithme 2.

6.1 Méthodologie

Les tests ont été automatisés à l'aide du script `run_test.sh` écrit en bash. Celui-ci fait appel à l'exécutable (au format *jar* et compilé avec la version 60.0 de la *JRE*) calculant le cercle minimum, mesure son temps d'exécution à l'aide de la commande UNIX `time` et écrit dans un fichier au format *csv* les résultats. Ainsi, on obtient le temps utilisé par le processeur, ce qui n'aurait pas été le cas si l'on avait utilisé les bibliothèques standards de Java qui affiche un temps réel et ne prennent donc pas en compte les autres programmes s'exécutant sur l'ordinateur et pouvant fausser les tests.

6.2 Comparaison des résultats

Les figures 3, 4 et 5 représentent le temps d'exécution des différents algorithmes en fonction du test. La moyenne de temps des tests est également affichée en orange.
On remarque, comme attendu que la moyenne de l'algorithme naïf est plus élevée que celles des deux algorithmes de Welzl qui prennent sensiblement le même temps à calculer un cercle minimum de $\mathcal{P}$.

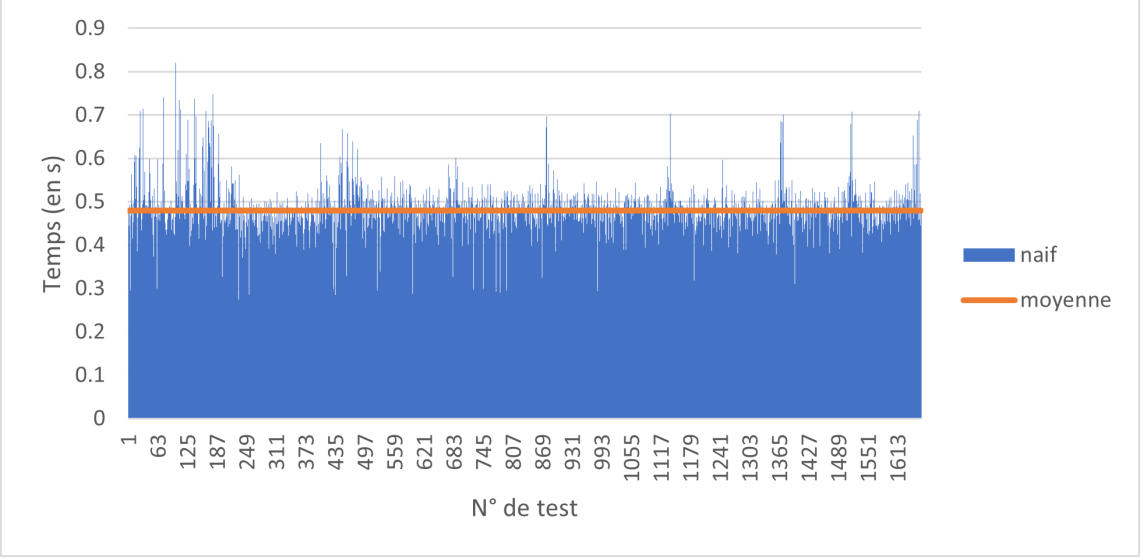

Figure 3 – Temps d'exécution de l'algorithme naïf

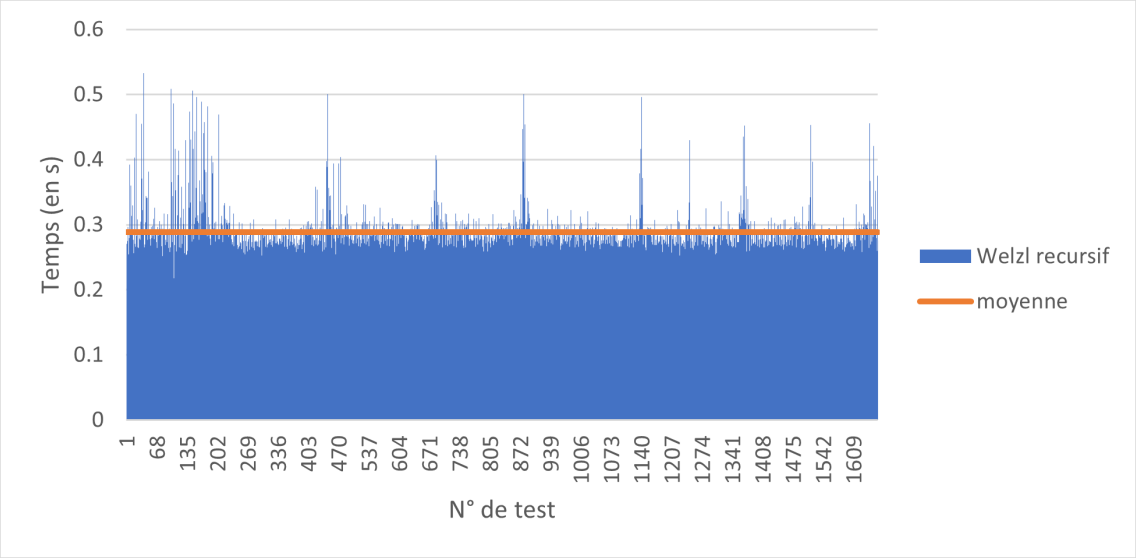

Figure 4 – Temps d'exécution de l'algorithme de Welzl récursif

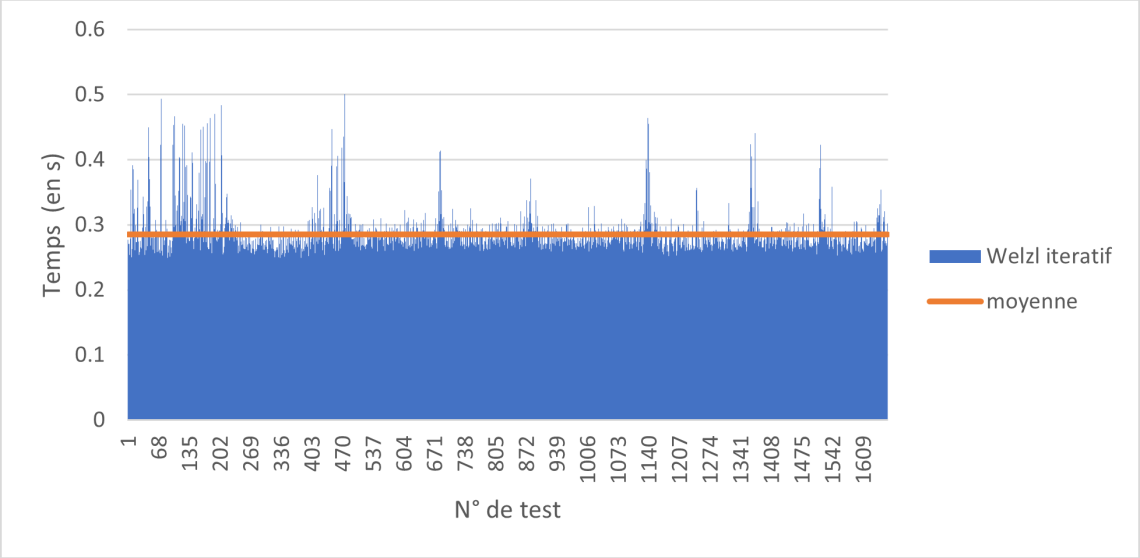

Figure 5 – Temps d'exécution de l'algorithme de Welzl itératif

7 Conclusion

L'algorithme de Welzl permet bien de calculer le cercle minimum de $\mathcal{P}$. Cependant, si sa complexité temporelle parait bien plus intéressante que celle de l'algorithme naïf, elle n'est efficace que sur un nombre de points assez restreint. En effet, un trop grand nombre d'appel récursifs peut conduire à des erreurs de dépassement de capacité de pile.
Pour éviter cela, il est possible de rendre cet algorithme itératif, au prix d'un taux d'occupation de la mémoire légèrement plus élevé afin de stocker les appels qui doivent être effectués.

8 Discussion

Ce rapport ne traite que de points dans le plan, mais il est possible d'étendre la réflexion a des points en trois dimensions. On parlerait alors de sphère minimum et les approches vues plus haut resteraient valables. Cependant, le temps de calcul serait plus long car il y aurait une coordonnée de plus à vérifier. De manière générale, le problème est généralisable en dimension n avec $n \in \mathbb{N}^*$. Cependant, les applications pratiques semblent se limiter aux dimensions 2 et 3.

Références

[1] Bui-Xuan, B.-M. Conception et pratique de l'algorithmique. https ://www-apr.lip6.fr/ buixuan/files/cpa2021/cpa2021_cours1.pdf, 2022.

[2] Welzl, E. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science*. Springer, 1991, pp. 359–370.