Minimum Enclosing Disk Problem

Burakhan Celenk

12.06.2021

1. Introduction

Minimum disk problem is initially proposed by english mathematician James Joseph Sylvester in 1857. It is a mathematical and also an algorithmic problem. The simplest definition of the problem is to find the minimum circle enclosing given a set of points.

There are a lot of scenarios that we can apply the solution of this problem. Therefore, solving this problem efficiently is more beneficial than we expect. Here are 3 examples of these scenarios;

1 - Let's say, in a video game we are moving our character which has a fluid body. While moving, its shape is consistently changing. In such a scenario, we need to calculate the minimum disk enclosing our character in each frame in order to find out if there is any collision with another object. This reveals that this problem needs to be solved very quickly.

2 - Imagine that an area with full of enemies has been detected by drones and the army wants to eliminate these enemies. As we know, every bomb has its own impact radius. In order to use the most efficient and cheapest bomb in the army, this problem must be solved. If we take into account the movements of the enemies , bomb need to solve this problem continuously and adjust its orientation along the path.

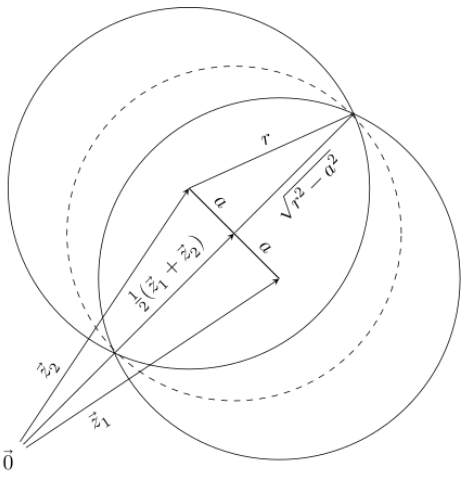
3 - The last example which is known as 1-center problem is the facility problem. Let's assume that we want to construct a new facility. The location of the new facility must be chosen to provide service to a number of customers, minimizing the farthest distance that any customer must travel to reach the new facility. We can clearly see that the facility problem is actually a minimum disk problem.

- **Characterization of the problem**

To solve this problem, most of the algorithms benefit from two simple facts;

1 - The minimum covering circle is unique.

Let P be any set of points in the plane, and suppose that there are two smallest enclosing disks of P, with centers at $z1$ and $z2$. Let r be their shared radius, and let $2a$ be the distance between their centers. Since P is a subset of both disks it is a subset of their intersection. However, their intersection is contained within the disk with center



$\frac{1}{2}(z1 + z2)$ and radius $\sqrt{r^2 - a^2}$. Since r is minimal, we must have $\sqrt{r^2 - a^2} = r$, meaning $a = 0$, so the disks are identical.

2 - The minimum covering circle of a set P can be determined by at most three points in P which lie on the boundary of the circle. If it is determined by only two points, then the line segment joining those two points must be a diameter of the minimum circle. If it is determined by three points, then the triangle consisting of those three points is not obtuse.

Nearly all algorithms that solve the minimum disk problem can be applied in different dimensions. But for the sake of simplicity we will examine and explain it in 2D. There are more than ten algorithms related to this problem, but we will only cover three of them;

1 - Naive Algorithm

2 - Welzl's Algorithm

3 - Ritter's Algorithm

- **Some notations for further usage**

Let P be a set of $n \in N$ points in R^2,

$p(x, y)$ be a point at coordinate $(x, y) \in R^2$,

$C(x, y, r)$ be a circle at center coordinate $(x, y) \in R^2$ with radius $r \in R$,

$C(p, q)$ be a circle consists of two points $p, q \in R^2$ on its boundary,

$C(p, q, r)$ be a circle consists of three points $p, q, r \in R^2$ on its boundary,

2. Algorithms

• Naive Algorithm

Naive algorithm is the simplest algorithm among the other algorithms. The principle of the algorithm follows through the second clause in the characterization section word by word with brute force approach;

if $|P| = 1$, there is only one point p in P, return $C(p.x, p.y, 0)$,

if $|P| > 1$, for every (p, q) pairs in P, draw $C(p, q)$ and test it against P,

if $C(p,q)$ encloses P, return $C(p,q)$,

if not, then for every (p,q,r) triples in P, draw $C(p,q,r)$ and test it against P,

if $C(p,q,r)$ encloses P, return $C(p,q,r)$.

Pseudo Code

```
Naive_Algorithm ( P set of points )

   \\ one point case

   if |P| = 1
      return C ( P[0].x , P[0].y , 0 )
   end if

   \\ if there are two points on the boundary

   for each point p in P
      for each point q in P
         if p != q
            if C ( p , q ) is enclosing P
               return C ( p , q )
            end if
         end if
      end for
   end for

   \\ if there are three points on the boundary

   C_min = C ( 0 , 0 , Infinity)
   for each point p in P
      for each point q in P
         for each point r in P
            if p , q and r are different and not co-linear
               C_temp = C ( p , q , r )
               if C_temp.radius < C_min.radius and C_temp is enclosing P
                  C_min = C_temp
               end if
            end if
         end for
      end for
   end for

   return C_min
end Naive_Algorithm
```

Time Complexity

- Best scenario (There is only one point in P)

$$O(1)$$

- Two points on the boundary

$$O(n^3)$$

- Three points on the boundary

$$O(n^4)$$

In the worst case scenario we will have $O(n^4)$ time complexity, which means as the number of points in P increases, necessary time for the calculation will also increase exponentially.

• Welzl's Algorithm

Welzl's algorithm is a little bit more complex than other algorithms in terms of comprehension. On the other hand, it is easy to implement after understand the general idea. Before start to explain the principle, keep in mind that this is a randomized incremental recursive algorithm. In other words it tries to do its best but it doesn't give you the absolute solution. The principle of the algorithm;

The minimum circle can consist of three points at most. And these three points are on the boundary. Let R be the set of points on the boundary of minimum circle. Welzl's algorithm is based on following three lemmas;

1 - If there is a circle covering P, and R is a set of 3 points on the boundary, this circle is the minimum enclosing circle.

2 - If a point p doesn't belong to a circle covering P - { p }, then p must belong to R.

3 - If there is a circle covering P, and R is a set of 3 points on the boundary, then there must be a circle covering S which is subset of P contains at most $3 - |R|$ points, where minimum circle enclosing P and minimum circle enclosing S are equals with same R.

Pseudo Code

```
Welzl_Algorithm( P set of points )

    return minimum_circle( P , {} )

end Welzl_Algorithm


minimum_circle( P set of points , R set of points )

    C_min = null

    if P is empty or |R| = 3
        C_min = circle containing R on its boundary
    else
        p = choose a random point in P
        C_min = minimum_circle( P - {p} , R )
        if C_min != null and p doesn't belong to C_min
            C_min = minimum_circle( P - {p} , R U {p} )
        end if
    end if

    return C_min

end minimum_circle
```

Time Complexity

At each time, the algorithm draws a random point p in P and calls itself with P - { p }. At the end, it executes first if block and return back step by step to the function where it's called. So the time complexity is $expected\ O(n)$. You can find a detailed explanation about time complexity of this algorithm and the meaning of $expected$ in the Emo Welzl's article.

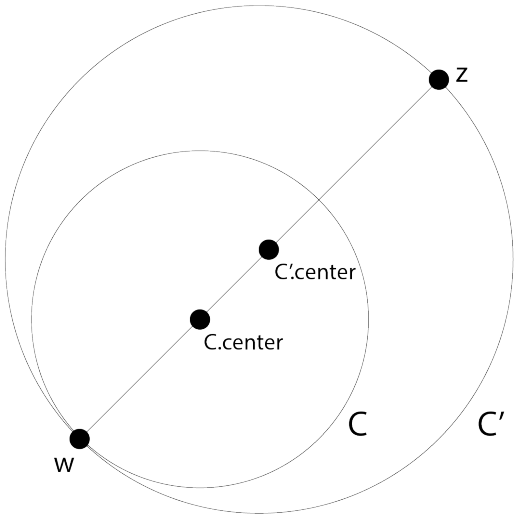
• Ritter's Algorithm

As we mentioned in the previous algorithm, this algorithm is also a randomized incremental algorithm but this one isn't recursive and more easy to understand and implement. The principle of the algorithm;

1 - Pick a random point p in P, find the point q in P, which has the largest distance to p. Then find the point r in P, which has the largest distance to q.

2 - Draw a circle $C(q, r)$.

3 - For each point z in P, check if it's outside of the circle C. If so, draw a new circle C' as illustrated in the image;



4 - After traverse all the points in P, return current circle which is the minimum enclosing circle.

Pseudo Code

```
Ritter_Algorithm( P set of points )

    p = random point in P
    q = point in P which has the farthest distance to p
    r = point in P which has the farthest distance to q

    C_min = C ( q , r )

    for each point z in P
        if z isn't inside C_min
            vector = vector from z to C_min.center
            distance = distance between z and C_min.center
            unit_vector = vector / distance
            w = unit_vector * C_min.radius + C_min.center
            C_min = C ( w , z )
        end if
    end for

    return C_min

end Ritter_Algorithm
```

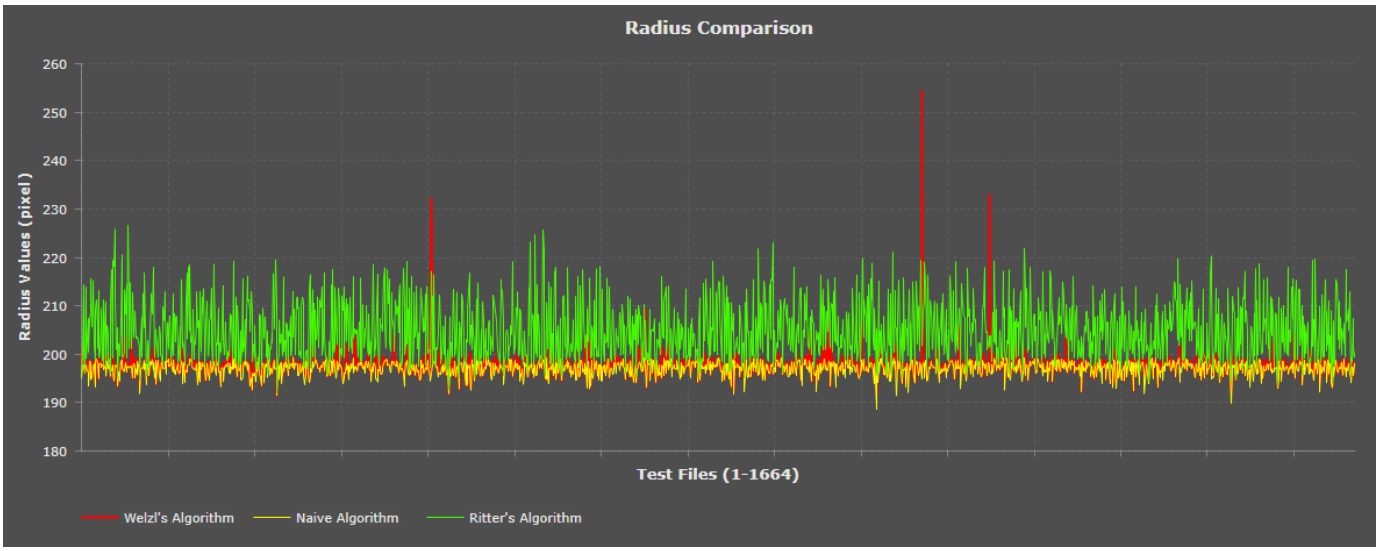
Time Complexity

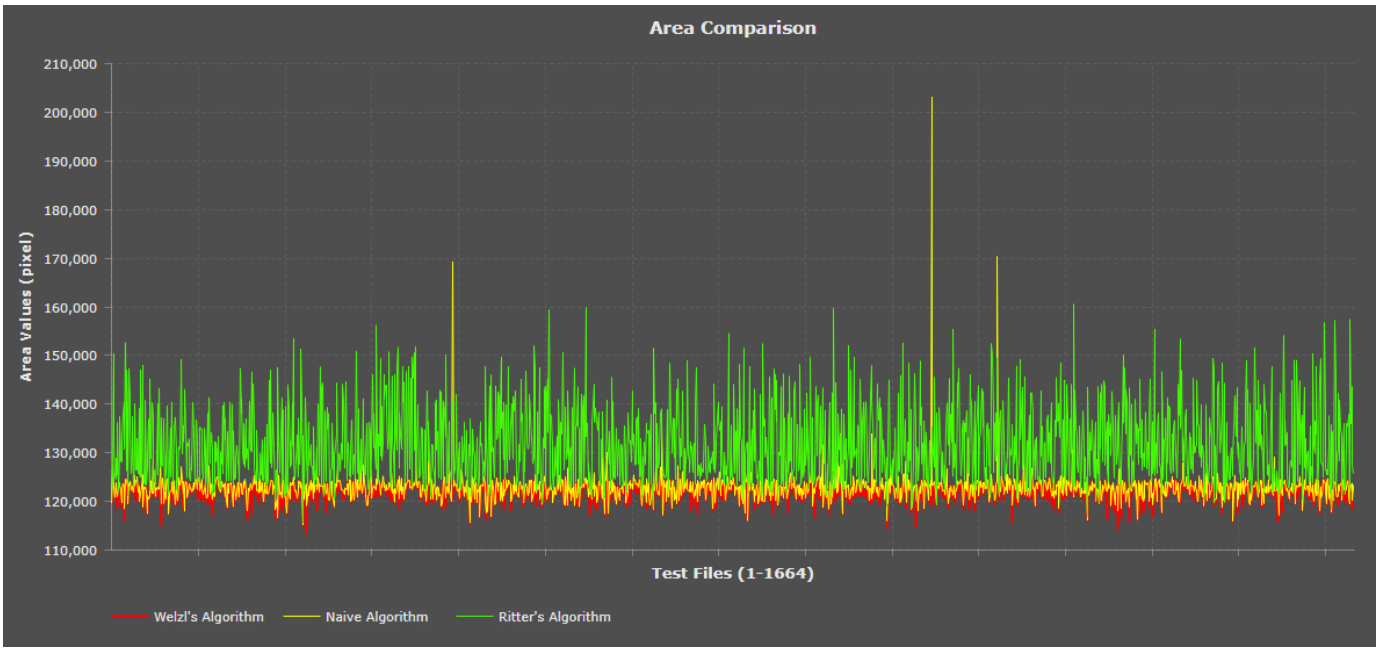
We spend O(n) time to find the points q and r. In the last for loop, we spend O(n) time. Constants doesn't mean anything in lambda notations. This means, in the worst case scenario we have O(n) time complexity.

• Results And Comparison of Algorithms

All the tests were realized on Varoumas benchmark test files composed of 1664 files. Each file is composed of 256 lines which represent (x, y) coordinates of points.

Radius, Center and Area Comparison





As you can see from the graphics, area and radius values obtained from Ritter's and Welzl's algorithms are almost same values in Naive algorithm(absolute values). Therefore, we can say that both algorithms do their job quite successfully. If we compare Ritter's and Welzl's algorithms between each other,

In terms of the sum of center and radius,

Welzl's algorithm has %0.79 deviation from absolute values

Ritter's algorithm has %2.74 deviation from absolute values

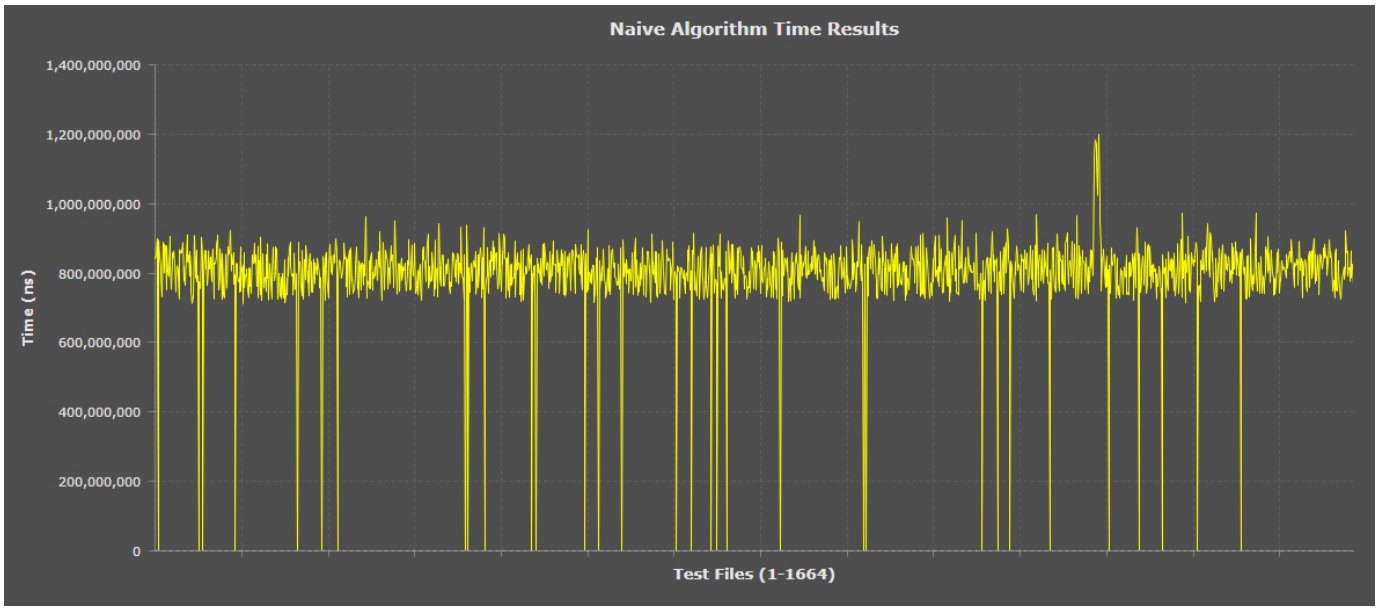
In terms of the area of minimum circle,

Welzl's algorithm has %2.41 deviation from absolute values

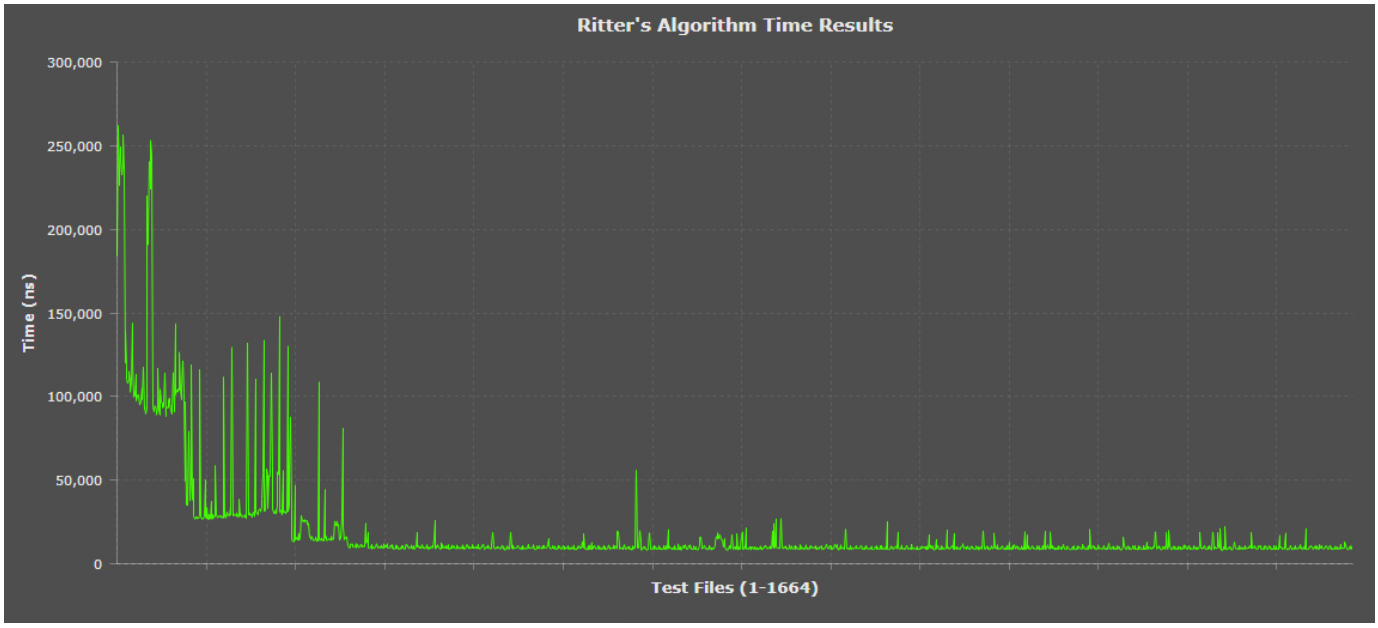
Ritter's algorithm has %10.41 deviation from absolute values

According the the comparison values, it is obvious that Welzl's algorithm is more precise than Ritter's algorithm.

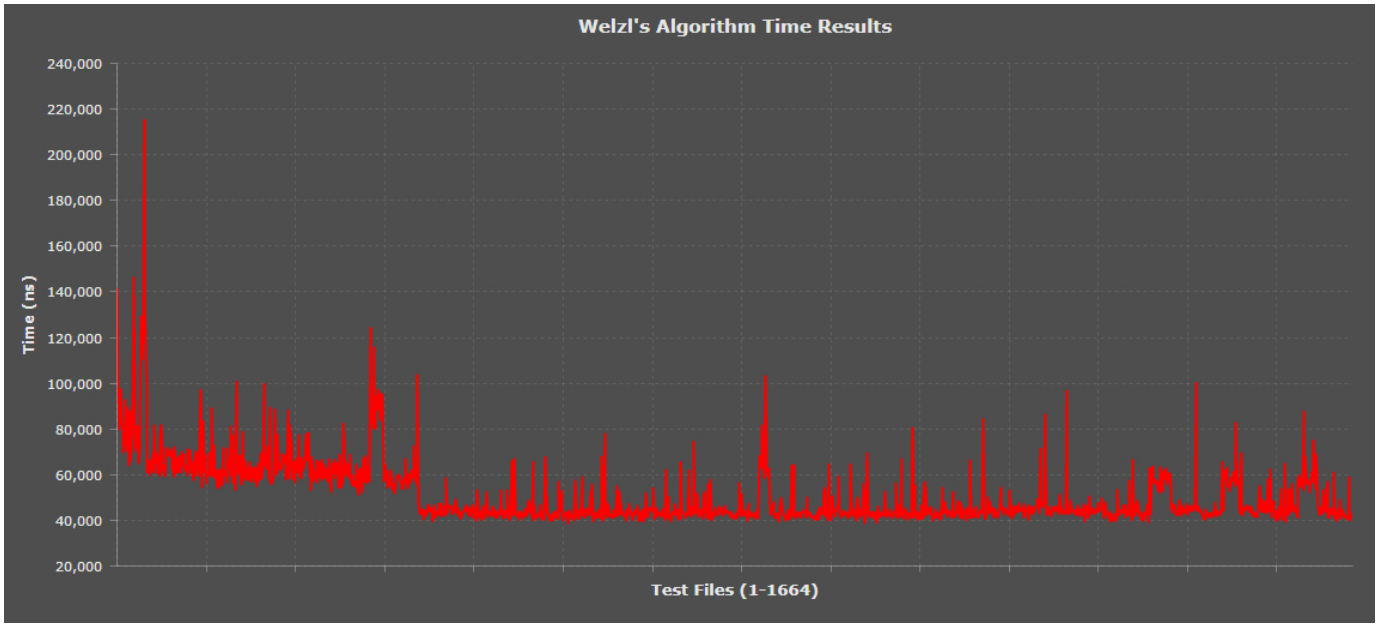
Time Comparison



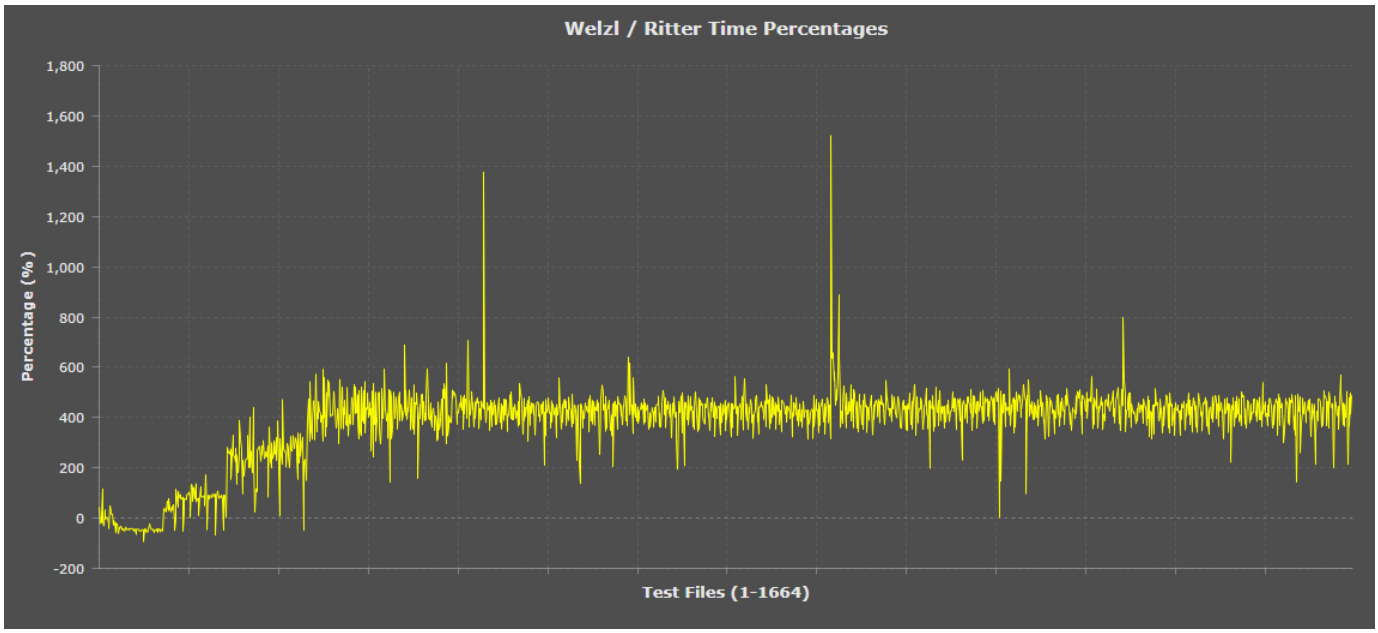
The execution time of the algorithms is measured in nanoseconds. In the chart above we can clearly see that the naive algorithm has finished his task around 800 milliseconds for each file which has 256 points. The total time spent for all files is around 22 minutes.



Ritter's algorithm has performed best timing results. In the chart above, Ritter's algorithm has calculated each minimum circle in approximately 0,02 milliseconds which is quite fast compared to the naive algorithm. To be more specific, it's $4 * 10^4$ times faster than naive algorithm. The total time spent for all files is around 34 milliseconds.



Welzl's algorithm is a little bit slower than Ritter's algorithm but compared to naive algorithm we can undoubtedly see the significant performance gain. For each file, Welzl's algorithm has calculated the minimum circle in roughly 0,05 milliseconds. The total time spent for all files is rounded up to 86 milliseconds.



And lastly, In the chart above we can get a better proportion about the execution times of Ritter's and Welzl's algorithms. It appears that we can execute Ritter's algorithm more than 4 times compared to Welzl's algorithm in a given time.

3 - Conclusion

Because of the nature of recursive algorithms, point sets which have more than 10^5 points will cause stack overflow exception in Welzl's algorithm if we don't change default settings in our IDE. For the Ritter's algorithm, apparently the only disadvantage is its error margin. As a conclusion, it doesn't make sense to pick an algorithm which suppose to be the best in every situation. Because it is an optimization problem, it's better to choose the most satisfactory algorithm for each scenario. For this reason;

If we don't have any boundary about the execution time and if we need the absolute solution then we can go along with the naive algorithm. If we have very limited time and we are ready to comply with bartering a little bit precision away, then we can pick either Welzl's algorithm or Ritter's algorithm according to the amount of precision and the number of points. If we have limited time for the calculation but we need more precise results then we can choose Megiddo's algorithm which we didn't mentioned in this article. It's better to be aware of the difficulty in terms of implementation and understanding of this algorithm.

References

1 - Emo Welzl (1991), Smallest Enclosing Disks (Balls and Ellipsoids)

2 - Jack Ritter (1990), An Efficient Bounding Sphere

3 - Nimrod Megiddo, Linear-time algorithms for linear programming in R^3 and related problems, 766 - 776.

4 - https://www.wikiwand.com/en/Smallest-circle_problem, Section Characterization of the problem

5 - George Marsaglia (2003), Xorshift RNGs, (implemented into source code)

Varoumas Benchmark Files

http://www-apr.lip6.fr/~buixuan/files/cpa2016/Varoumas_benchmark.zip