FAST METHOD FOR SOLVING THE MINIMAL OVERLAPPING CIRCLE EXPANSION PROBLEMS

ANDY ZHUANG

Abstract. In this paper, we first introduce the Minimal Overlapping Circle Expansion (MOCE) problem. Solution to such a problem has real-world applications, such as finding the location to best communicate with a number of wireless devices, finding the quickest way for a number of vehicles to get to a rendezvous location etc. We present several algorithms to compute the solution with different running time and accuracy. The first uses enclosing square to get an approximate solution; the second only considers pair-wise overlap to approximate; the third uses the the results of the first two and a few other methods to speed up the computation. Our results show that (1) the approximate algorithm can be 1000 times faster than the accurate algorithm, and get to 99.9% of the correct value. (2) improvements can cut down the compute time by 50% for the accurate algorithm.

1. Introduction

Several cases of real-life problems can be described as:

- There are a number of circles centered at specific, fixed locations.
- The size (radius) of all circles can be simultaneously scaled (expanded) with a global scaling factor.
- We want to find the minimal scaling factor such that all circles overlap.

The following are some examples.

One type of transportation optimization problem is to find a rendezvous point that a number of vehicles located in different places initially can all reach with the minimal amount of travel time. Figure 1 shows an example that three vehicles are initially located at three separate spots. They can travel at different speed. In this case, the scaling factor is time. All circles expand linearly as time progresses. The objective is find the minimal time such that all three circle overlaps.

The second scenario is wireless communication with multiple people. Each person has a mobile device and stay at a particular location, and their device has different battery levels. Now, the question is: if someone want to communicate with all of them for as long as possible, which location should the person pick. In this case, we know the battery level of those mobile devices. In order to communicate as long as possible, the power consumption rate should be proportional to the battery levels. Then, power consumption rate determines the radius of circle the phone signal can reach and communicate. The scaling factor controls the average power consumption rate, which can be translated to the radius of the circle centered at each person's location.

The third scenario is wireless charging [1] [2]. There are a number of devices that need to be remotely charged and their battery level can be different. A charger needs to pick a location such that all devices are charged together. The power received by each device

Date: October 27, 2024.

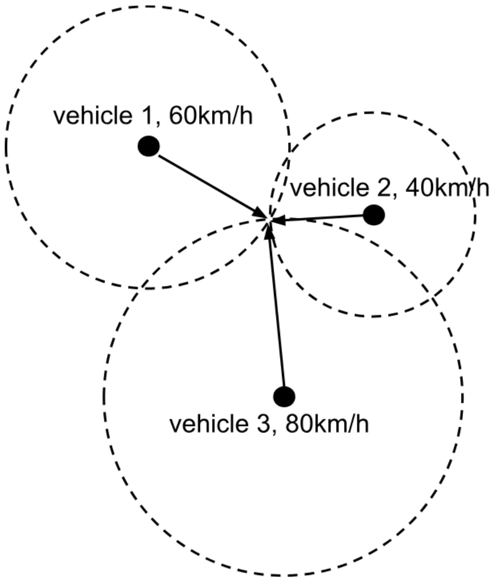


Figure 1. Vehicle rendezvous problem.

is given by $p_r = \frac{\alpha}{(\beta+d)^2}p_c$, where p_c is the charging power, d is the distance and α, β are constants. In order to save energy, the charger should aim to charge all devices for the same period of time. In other words, p_r should be proportional to the energy needed to fully charge their battery. From p_r, we get the radius for each circle based on the formula above. Similar to the second scenario, the scaling factor is the inverse of the mean of p_r. A higher scaling factor increases the size of each circle, until we find a common point that all circles can reach.

The rest of this paper is organized as follows: Section 2 talks about problem definition and modeling; Section 3 presents preliminaries; Section 4 discusses the algorithms; Section 5 presents some simulation results and Section 6 concludes the paper.

2. Problem Definition and Mathematical Modeling

Before laying out the definition of the problem, we would like to introduce a couple of definitions about how objects overlap on a plane.

Definition 2.1. A set of objects are said to *Pair-wise Overlap* if each pair of objects overlap, i.e. share at least one common point.

Definition 2.2. A set of objects are said to *Fully Overlap* if all objects share at least one common point.

Definition 2.3. The *Minimal Overlapping Circle Expansion (MOCE)* problem is defined as follows:

We are given a scaling factor $sf \in \mathbb{R}^+$ and n circles. For circle $i, i \in [1..n]$, it has fixed center at c_i on a 2-D plane, and its radius $r_i(sf)$ is a monotonically non-decreasing function of sf. Find the minimal value of sf_{full_cir} s.t. all circles fully overlap.

3. Preliminaries

In this section, we will introduce some basic theorems and corollaries about finding shared points among circles on a plane. They will be used in our fast algorithms later on.

3.1. **Helly Numbers.**

We would like to introduce a famous theorem by Helly, which lays the foundation for the algorithm developed in this article.

Theorem 3.1. *(Helly [3]) Let $\mathcal{F}$ be a finite family of convex sets in $\mathbb{R}^d$. If every $d+1$ (called Helly number) or fewer sets in $\mathcal{F}$ have a point in common, then all sets $\mathcal{F}$ have a point in common (fully overlap).*

When applied to 2-D convex objects, the Helly theorem essentially says, as long as every three or fewer objects overlap, all objects fully overlap.

The Helly theorem applies to all convex shapes. Although circle is a special case, the Helly number is not lower.

Corollary 3.2. *The Helly number for circles on a 2-D plane is 3.*

Proof. According to Helly's theorem, the Helly number is 3. We just need to show 2 or lower is not possible. As a counter-example, Figure 2 shows that for three circles, even if the circles pair-wise overlap, they do not fully overlap. ■

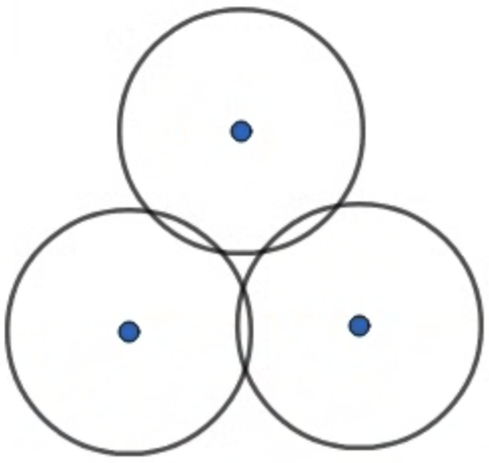


Figure 2. Helly number is not 2 for circles on a plane.

Now let's look at another type of convex shape: squares.

Corollary 3.3. *The Helly number for squares with edges parallel to the x or y axes on a 2-D plane is 2.*

Proof. Assume that circles are pair-wise overlap, we need to prove that all squares fully overlap. First, we project each square to the x-axis, so each square becomes a line segment. Each pair of line segments overlap. We can apply Helly's theorem to the one-dimension case. All line segments should fully overlap. Let's call that point x'. Similarly, we project each square to the y-axis, and there is a point covered by all line segments called y'. It is clear that (x', y') is a common point on the plane that is shared by all squares. ■

3.2. **Minimal Enclosing Square of a Circle.**

Since the Helly number for square is smaller than the one for circles, we want to make use that to speed up the overlap checking for circles.

Definition 3.4. *Minimal Enclosing Square* of a circle is the smallest square that covers the whole circle.

Apparently, there are many minimal enclosing square for the same circle just by rotating the square around the center of the circle. To distinguish the minimal enclosing squares, we define their angle.

Definition 3.5. *Angle of a Minimal Enclosing Square* of a circle: the acute angle between the extension of one of the sides of the square and the x-axis.

One of the ideas of using minimal enclosing square comes from the following proposition.

Proposition 3.6. *Given a set of circles, if they full overlap, their minimal enclosing squares (at a particular angle) should also fully overlap.*

The proof is trivial for this proposition. In addition, this is a necessary but not a sufficient condition for the set of circles to fully overlap. In other word, the inverse may not be true. However, if the circles do fully overlap, their minimal enclosing squares at any angle should also fully overlap. In other words, if we find that there is an angle such that their minimal enclosing square at that angler do not fully overlap, the circles must not fully overlap. Therefore, we should try different angles for enclosing squares to check if the circles fully overlap.

3.3. **Radical Axis and Radical Center.**

To further study the overlapping of circle, we would like to mention *Radical Axis* and *Radical Center (aka Power Center)*.

Remark 3.7. Both radical axis and radical center are well-known concepts. In this study, we only look at two circles that intersect, so the radical axis is the line going through the two intersecting points.

Theorem 3.8. *(Dörrie [4]) The three pair-wise radical axes of three circles are concurrent in a point known as the radical center.*

Remark 3.9. If the centers of the three circles are collinear, the three pair-wise radical axes are parallel to each other, or the radical center is at infinity.

3.4. **Intersection Chord.**

For two overlapping circles, we are particularly interested in the intersection part.

Definition 3.10. For two intersecting circles, define the *intersection chord* as the chord connecting the two intersection points. It is a chord shared by both circles, and it is part of the radical axis.

As will be stated later, the intersecting chord is particularly important for figuring out how circles intersect with each other. Thus, we want to illustrate how to quickly find the intersecting chord given the coordinates of the centers and radii of two circles.

The calculation of intersection only requires basic method using Pythagorean theorem, as shown in Figure 3. Based on analytic geometry, once we get h and d_a, we can find the coordinates of P and Q. Note that, in some cases, we are only interested in the angle of the endpoints of the chord, when dealing with intersection chords for a particular circle. We only need to calculate the radian coordinates for P an Q with regard to A as the origin.

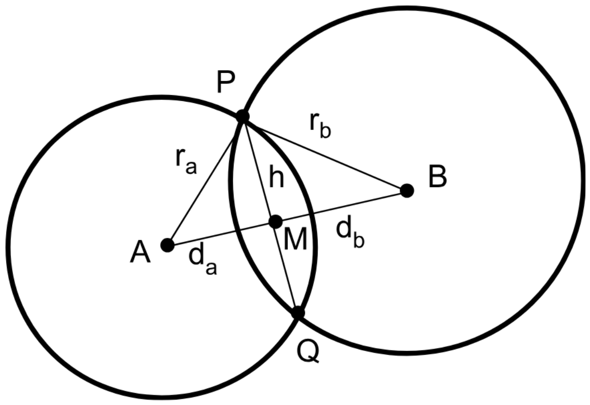


Figure 3. Calculation of the intersection chord.

3.5. **Cases for Three Pair-wise Overlapping Circles.**

As mentioned earlier, based on Helly's theorem, the focus of this paper is to study how three circles overlap.

From now on, let's assume that the circles are pair-wise overlap, because it is much easier to check if two circles overlap by just comparing the center-to-center distance with the sum of their radii.

In the meantime, we rely on the radical center to explain some of the scenarios, so it is helpful to first separate cases depending on whether the centers of the three circles are collinear or not.

So, let's first cover two trivial scenarios.

Proposition 3.11. *If circle A is fully inside circle B, we do not need to consider circle B when deciding if the circles fully overlap.*

This is obvious, since if there is common point shared by the rest of circles, the point is also inside circle B, because it is completely inside circle A.

Another more powerful extension of this proposition takes into consideration of how their radii grows with the scaling factor.

Theorem 3.12. *If circle A subsumes circle B when scaling factor equals sf_{thd} and the derivatives of their radius function satisfy $r'_a(x) \geq r'_b(x)$, then circle A subsumes circle B for all $sf \geq sf_{thd}$.*

Proof. Since circle A subsumes circle B when scaling factor equals sf_{thd}, it means:

$$r_a(sf_{thd}) \geq r_b(sf_{thd}) + d$$

where d is the center-to-center distance.

Also, because $r'_a(x) \geq r'_b(x)$, we take the integral from sf_{thd} to sf on both sides of this inequality:

$$\int_{sf_{thd}}^{sf} r'_a(x)dx \geq \int_{sf_{thd}}^{sf} r'_b(x)dx \implies r_a(sf) - r_a(sf_{thd}) \geq r_b(sf) - r_b(sf_{thd})$$

Adding both sides of the two inequalities, we get:

$$r_a(sf) \geq r_b(sf) + d$$

Thus, circle A subsumes circle B for all $sf \geq sf_{thd}$ ■

Since the derivative of the radius function typically satisfies the condition, this theorem can be used to reduce the need to check circle subsumption repeatedly.

Theorem 3.13. *If three circles pair-wise overlap, and their centers are collinear, they must fully overlap.*

Proof. Let's assume circles A and C overlap with centers at point A and point C. Let's also assume the center of circle B is in between of A and C. Since circles A and C overlap, they should intersect with line segment AC. As shown in Figure 4, the two intersection points are P and Q. There are three cases for the location of the center of circle B:

- If B is to the right of Q, since circles A and B intersect, so Q must be on all three circles.
- If B is to the left of P, since circles B and C intersect, so P must be on all three circles.
- If B is in between P and Q, B must be on all three circles.

■

Next, we would like to discuss cases when three circles are pair-wase overlap but their centers are not on the same line. The main theorem is as follows.

Theorem 3.14. *For three circles that pair-wise overlap, there are a total of five non-trivial cases as shown in Figure 5.b to Figure 5.f.*

Proof. Let's start with two circles: circle A and B. Assume they intersect at two points. The intersection creates an intersection chord and splits circle A into two arcs: a_1 and a_2. It also splits circle B into arcs b_1 and b_2. We call the arc inside the other circle *overlap arc.* Therefore, a_2 and b_2 are overlap arcs, whereas a_1 and b_1 are non-overlap arcs.

Circle C needs to intersect with both circle A and circle B, creating two additional intersection chords. There are several cases for the two additional intersection chords.

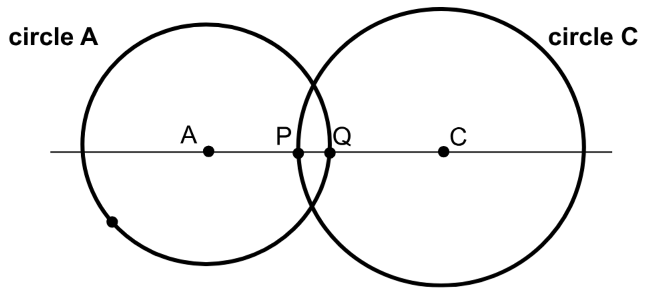


Figure 4. Three pair-wise overlap circles with centers collinear must share a common point.

- Case 1: They both touch one overlap arc and one non-overlap arc, as shown in Figure 5.b.
- Case 2: One intersection chords is on an overlap arc and the other is on a non-overlap arc, as shown in Figure 5.c.
- Case 3: Both intersection chords are on an overlap arc, as shown in Figure 5.d.
- Case 4: Both intersection chords are on non-overlap arc and circle C encloses the intersection chord between circles A and B, as shown in Figure 5.e.
- Case 5: Both intersection chords are on non-overlap arc and circle C is outside of the intersection chord between circles A and B, as shown in Figure 5.f.

■

Based on the five cases, it is easy to deduce:

Corollary 3.15. *If the radical center lies outside of the union of the three circles and its enclosed region, the three circles fully overlap.*

Proof. This is essentially case 2, 3, and 4 on Figure 5. For case 3 and 4, a point on the intersection chord of circle A and circle B is the common point they share. For case 2, a point on the intersection chord of circle A and circle C is the common point they share. ■

Interestingly, Theorem 3.13 can be considered a special case when the radical center is at infinity.

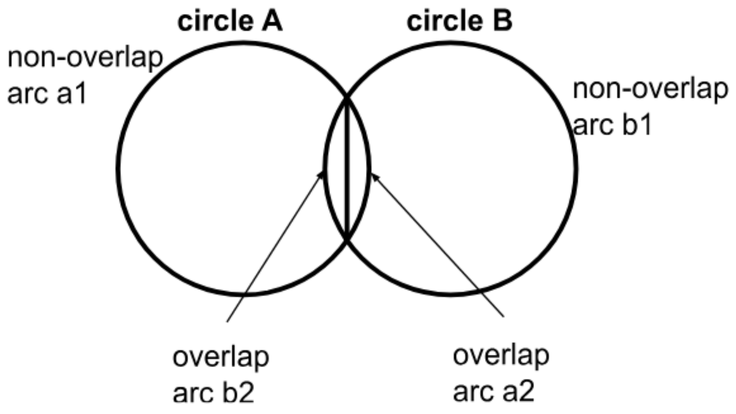
Going back to figure out if three circles fully overlap, the follow corollary says we only need to consider one of the cases on Figure 5.

Corollary 3.16. ***(Radical Center Rule)*** *Three pair-wise overlap circles do not fully overlap if and only if the radical center is inside the enclosed region of the union of the three circles but not inside any of the circles.*

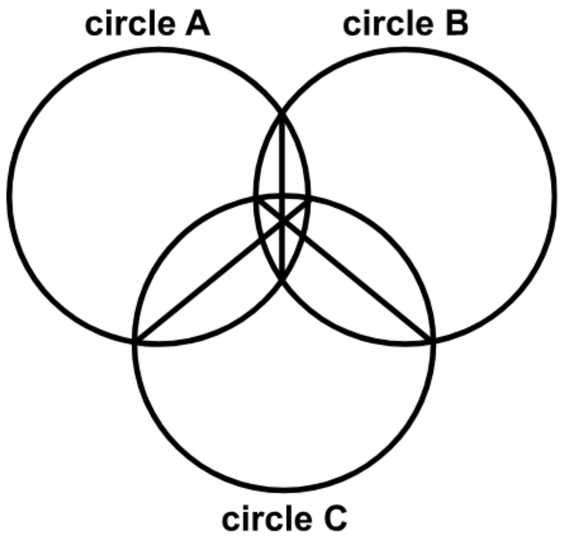
Proof. Basically, the three circles do not fully overlap if and only if case 5 happens, for which the radical center is inside the gap between the three circles. ■

3.6. **One-to-Many Circle Overlapping Check.**

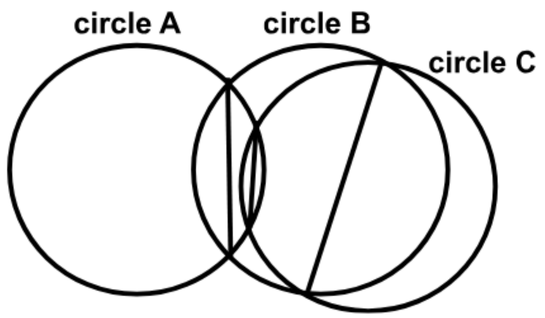
The theorems and corollaries in the previous subsection only considers three circles. For n circles, it is expensive to check every three of them, because there are $\binom{n}{3}$ possibilities. In



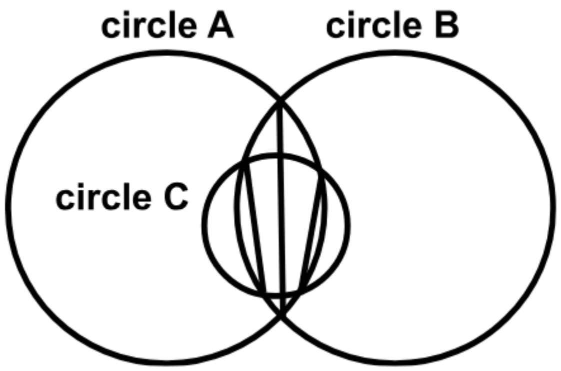
(a) two circles with four arcs



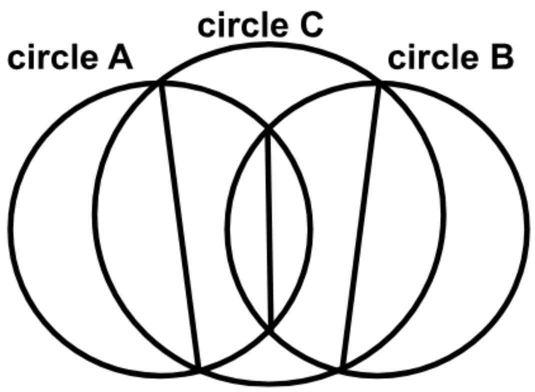
(b) case 1: radical center in overlap region



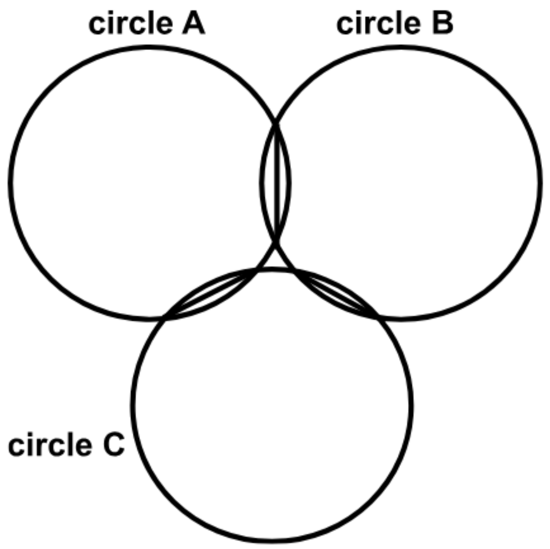
(c) case 2: mixed overlap arc



(d) case 3: overlap arc



(e) case 4: non-overlap arc, external radical center



(f) case 5: non-overlap arc, internal radical center

Figure 5. Five non-trivial cases for three pair-wise overlapping circles.

this subsection, we are interested in the set of circles that overlap with the same circle. For example, we have circle A, which intersects with circle 1 to circle $n-1$ individually. Now, let's consider all three-circle groups that circle A is one of the three-circles. There are $\binom{n-1}{2}$ such three-circle groups.

Definition 3.17. If circle B intersects with circle A, which splits A into two arcs. One of the arc a is completely in B. We say arc a is *covered* by circle B, and it is a *covered arc* by circle B.

With the concept of covered arc, we can derive the following proposition.

Proposition 3.18. *If circle A overlaps with both circle B and circle C, and A's arc covered by B overlaps with A's arc covered by circle C, the three circles fully overlap.*

Proof. This is quite simple to prove, because the arc of A covered by B is on both circle A and circle B, similarly the arc of A covered by C is on both circle A and circle C. Therefore the intersection of the two arcs is on all three circles. ■

Theorem 3.19. *If circle A overlaps with both circle B and circle C, and A's arc covered by B subsumes A's arc covered by circle C, then we only need to consider circle C.*

Proof. It is equivalent to prove that if there is another circle D and circles A, C, D fully overlap, then circles A, B, D also fully overlap. If we look at the five cases on Figure 5, it is clear that if circles A, C, D fully overlap, the arc covered by D on A must overlap with the arc covered by C on A. Since the latter is fully subsumed by the arc covered by B, the arc covered by B must also overlap with the arc covered by D. According to proposition 3.18, circles A, B, D also fully overlap. ■

Given Theorem 3.19, we can now assume that covered arcs with A are either disjoint or partially overlap with each other. Figure 6 illustrates the two possibilities. Figure 6.a is essentially the same as case 5 in Figure 5.f. Figure 6.b is similar to Figure 5.e. In order to distinguish the two, we can use the radical center rule in Corollary 3.16. However, sometimes there is a quicker way called the **Center Distance Rule**.

Theorem 3.20. *(**Center Distance Rule**) If circle A, B and C pair-wise overlap, and assume that the distance from the center of A to the line connecting the centers of B and C is d. Also, assume the radii of circle A , B and C are r_a, r_b and r_c respectively. The distance between centers of circles A and B is d_{ab}, and the distance between centers of circles A and C is d_{ac}. Then the three circles fully overlap if the following two inequalities are satisfied:*

$$\sqrt{d_{ab}^2 - d^2} \le \sqrt{r_a^2 - d^2} + r_b$$

$$\sqrt{d_{ac}^2 - d^2} \le \sqrt{r_a^2 - d^2} + r_c$$

Proof. Figure 7 shows that there are two cases to consider. In Figure 7.a, center of circles B and C are on different sides of AH, where AH is the line segment perpendicular to the line connecting the centers of circles B and C. Based on the Pythagorean theorem, it is easy to see that the two inequalities mean: point P is inside circle B and point Q is inside circle C. Since circle B and circle C intersect, some points between P and Q must be inside both circle B and circle C. Otherwise there is no way for the two circles to intersect. Therefore, those points are inside all three circles.

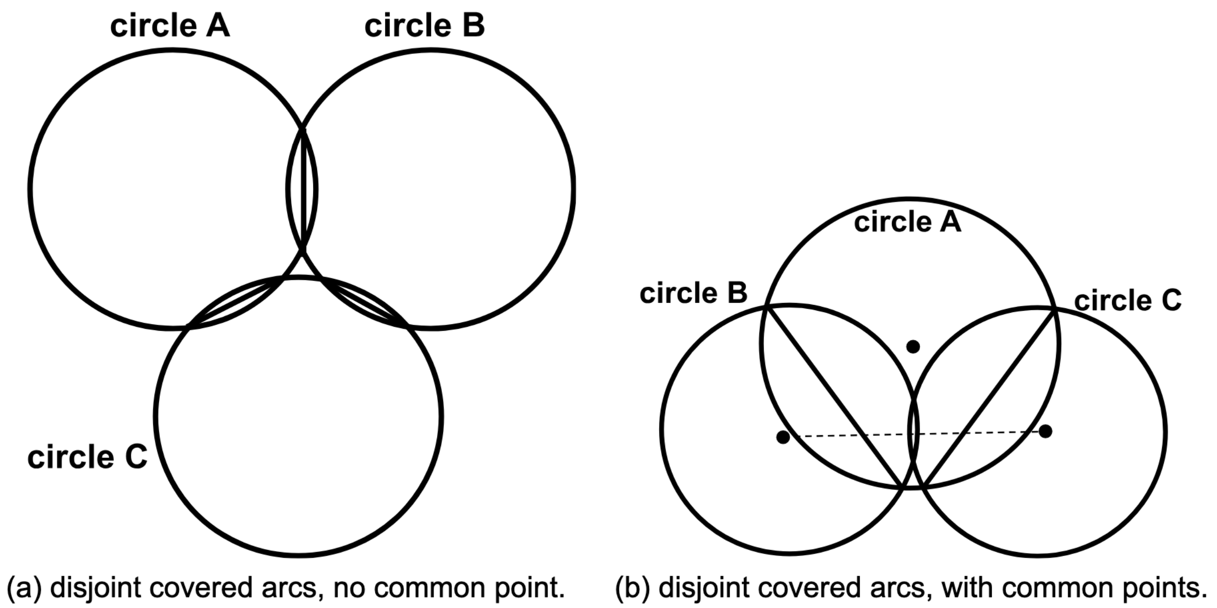

circle A
circle B
circle C
(a) disjoint covered arcs, no common point.
circle A
circle B
circle C
(b) disjoint covered arcs, with common points.

Figure 6. Two cases for disjoint covered arcs, with and without common point.

In Figure 7.b, center of circles B and C are on different sides of AH. Now, point P is in both circles B and C. It is already in circle A, so it belongs to all three circles.

There are other cases, for example one of the center of circle B or circle C is inside circle A or even both centers are inside circle A. The proof in such cases is obvious thus omitted. ■

Note that Theorem 3.13 is a special case of the Center Distance Rule, because when the centers are collinear, $d = 0$. The two inequality are trivially satisfied because they pair-wise overlap.

4. The Algorithm

4.1. **Overview of the Algorithm.**

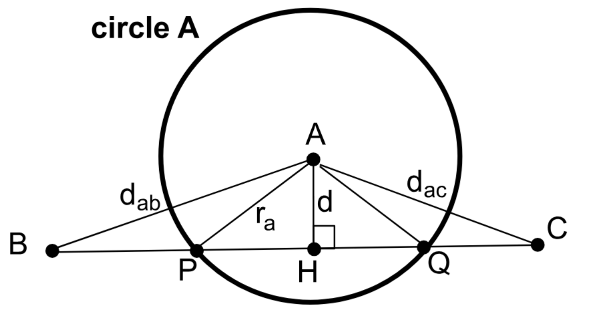
We have talked about several types of overlaps:

- Enclosing squares fully overlap.
- All circles pair-wise overlap.
- All circles fully overlap.

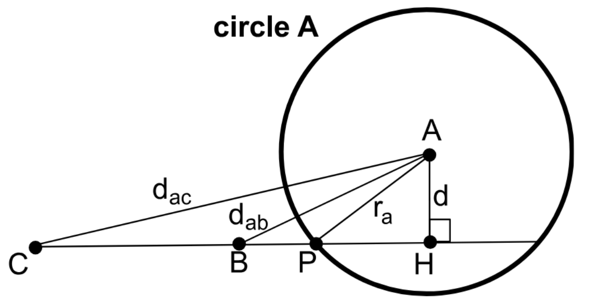
For each type of overlap, there is a minimal sf when the overlap happens. Define the following for such minimal sf values.

Definition 4.1.

- sf_{full_sq}: minimal sf for enclosing square to fully overlap.
- sf_{pair_cir}: minimal sf for circles to pair-wise overlap.
- sf_{full_cir}: minimal sf for circles to fully overlap.



(a) Centers of circles B and C are on different sides of AH.



(b) Centers of circles B and C are on the same side of AH.

Figure 7. Illustration for the proof of center distance rule.

Figure 8 shows their relationships with regard to the value of the scaling factor sf. sf_{full_sq} is less than or equal to sf_{full_cir} and sf_{pair_cir} is also less than or equal to sf_{full_cir}. There is no strict ordering between sf_{full_sq} and sf_{pair_cir}.

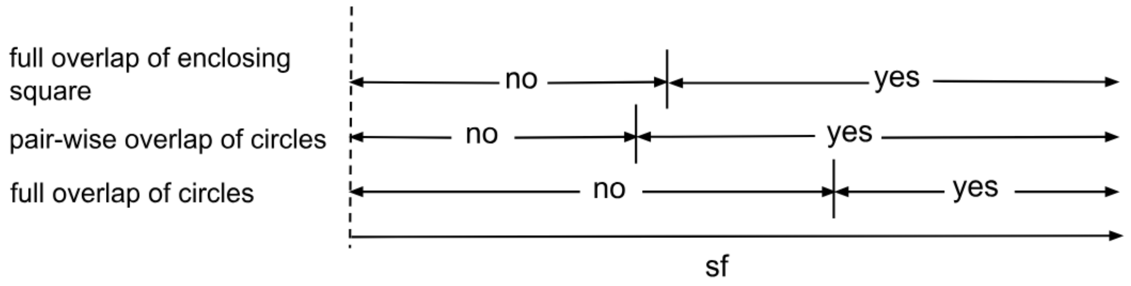

full overlap of enclosing square
pair-wise overlap of circles
full overlap of circles
no
yes
no
yes
no
yes
sf

Figure 8. Scaling factor affects circle and enclosing square overlap.

An overview of our algorithm to find sf_{full_cir} is described in Figure 9.

The two boxes on the left finds sf_{full_sq} and sf_{pair_cir}. We take the maximal of the two and use that as input to find sf_{full_cir} on the right. The cost of getting sf_{full_sq} and sf_{pair_cir}, therefore we want to get them first and the algorithm to find sf_{full_cir} can start at a value that is much close to the the final sf_{full_cir}.

4.2. **Algorithm to Compute** sf_{full_sq}.

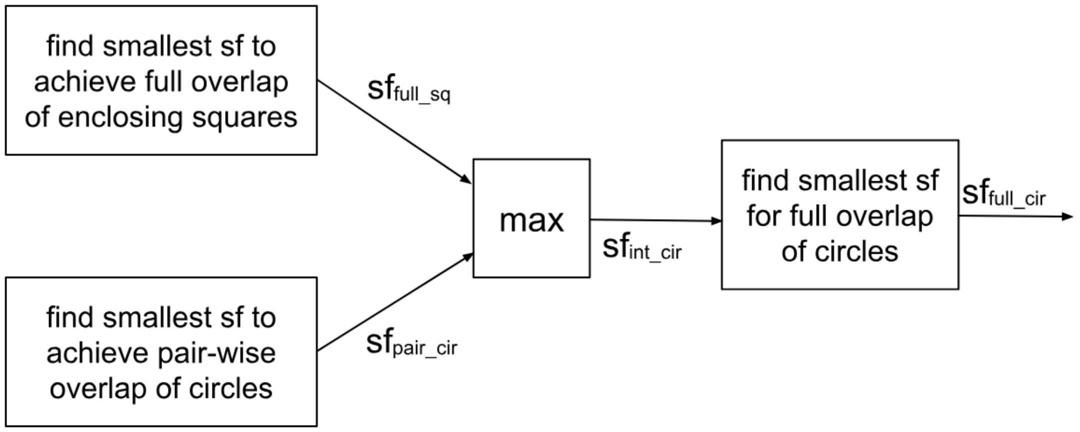

find smallest sf to achieve full overlap of enclosing squares
sf_full_sq
find smallest sf to achieve pair-wise overlap of circles
sf_pair_cir
max
sf_int_cir
find smallest sf for full overlap of circles
sf_full_cir

Figure 9. Overview of algorithm workflow.

The first step is to compute sf_{full_sq}. The pseudocode is shown in Algorithm 1. The ComputeSFSquare functions takes three parameters: the array of centers for all the circles, the array of radius functions for all the circles and the number of circles.

At the high level, it uses a binary search to find the smallest scaling factor. sf_{low} is a scaling factor that the enclosing squares do not overlap; whereas sf_{high} is a scaling factor that the enclosing squares fully overlap. If the two are close enough (less than a predetermined small value δ, we just terminate the search and return sf_{high}, otherwise we check the midpoint sf_{mid} of the two values, and assign sf_{mid} to either sf_{low} or sf_{high} depending on whether or not the enclosing squares fully overlap at sf_{mid}.

There are two for-loops between lines 7-18. Proposition 3.6 says we should iterate through different angles to detect if the enclosing squares overlap. So, the outer loop starting at line 7 goes through different angles. The inner loop starting at line 8 goes through all the circles, and project the circle to two lines that are perpendicular to each other, one with angle θ degree and one with angle $\theta + 90$ degree. We end up getting two sets of line segments.

Line 12 and 13 checks if the line segments in ls_x fully overlap. Since each line segment has a start point an end point with start ¡ end, as long as the maximum of all start points is less than or equal to the minimum of all end points, they must overlap at a certain point. Similarly, we do this for ls_y.

4.3. Algorithm to Compute sf_{pair_cir}.

The second algorithm (Algorithm 2) is to compute sf_{pair_cir}. The binary search part is the same as in the previous algorithm. Since we care about the overlap between pairs of circles. The first step on line 2 is to generate all the circle pairs.

For each sf_{mid} value, a simple approach is to go through all pairs of circles and check if they overlap. But note that as long as one of the pairs do not overlap, we can exit the check. Therefore, we use a heuristic to examine the pairs that are most unlikely to overlap first, hoping that we can finish early. The idea is to estimate how quickly the two circles might overlap. We estimate the growth rate, which is essentially the change of the sum of their radii over the change of the scaling factor. Then divide the distance between their centers by the growth rate. This estimate is implemented in function *SortByDistOverGrowth* at line 7.

One improvement we made is to not sort the *cir_pairs* at line 7 during each iteration, because we do not need to always recalculate the growth rate, it is just a rough estimation.

4.4. Algorithm to Compute sf_{full_cir}.

The last algorithm (Algorithm 3) computes sf_{full_sq}. As described on Figure 9, we take the maximal value of the outputs from the previous two algorithms and use that as the starting scaling factor (i.e. the initial value of sf_{low}).

We keep the state for each circle in cir_states. This essentially stores the location of the center, the current radius, and the neighboring circles we need to check overlap. The states are initialized at line 2 by calling *GenerateCircleState*.

Inside the while loop, once sf_{mid} is calculated on line 6, we call *UpdateCircleState*. This function recalculates the radii. Next, function *RemovingEnclosingCircles* removes any circle that encloses another circle according to Proposition 3.11.

Again, a heuristic is used to decided the order of checking circle overlap. We can exit early as soon as we can find three circles that do not fully overlap. The heuristic is that we should start with circles with the biggest radius. Since for big circles, we are more likely to find two of its neighbors that might have a gap in the middle as shown in Figure 5.f.

We go through each circle in the for loop starting line 11. This is essentially the one-to-many case in Subsection 3.6. Theorem 3.19 tells us that if one neighbor's covered arc is completely subsumed by another neighbor's covered arc, the latter neighbor can be ignored. This is implemented by function *RemoveArcSubsumingNeighbors*. For the remaining neighbors, we generate each pairs and check them together with circle i using *CheckOverlap-CenterDist* at line 15. This function uses the center distance rule (Theorem 3.20) to check if each three-circle group fully overlap. As mentioned in the theorem, it only applies when $d <= r_a$, which is the majority of the cases. For the remaining ones, we use the radical center approach to check if the radical center is inside the union of the three circles or not as a way to determine if they fully overlap.

One optimization to *RemoveEnclosingCircles* at line 8 is that we can use Theorem 3.12. Once a circle subsumes another circle, it is always the case with a higher sf value, therefore once a circle is removed it is always removed when sf gets bigger. Similarly, if the derivative of the radius function satisfies the condition in Theorem 3.12, we only need to call *SortByRadius* once.

Algorithm 1 Compute sf_{full_sq}

```
function ComputeSFSquare($c$, $r$, $n$)
    $sf_{low} \leftarrow 0$
    $sf_{high} \leftarrow HIGH\_VALUE$
    while $sf_{high} - sf_{low} \geq \delta$ do
        $sf_{mid} \leftarrow (sf_{low} + sf_{high})/2$
        $if\_overlap \leftarrow true$
        for $\theta \leftarrow 0, 90$ do
            for $i \leftarrow 1, n$ do
                $ls\_x[i] \leftarrow ProjectLineSegment(c[i], r(i, sf_{mid}), \theta))$
                $ls\_y[i] \leftarrow ProjectLineSegment(c[i], r(i, sf_{mid}), \theta + 90))$
            end for
            if $\max(ls\_x[*].start) > \min(ls\_x[*].end)$ then
                $if\_overlap \leftarrow false; break;$
            end if
            if $\max(ls\_y[*].start) > \min(ls\_y[*].end)$ then
                $if\_overlap \leftarrow false; break;$
            end if
        end for
        if $if\_overlap = true$ then $sf_{high} \leftarrow sf_{mid}$
        else $sf_{low} \leftarrow sf_{mid}$
        end if
    end while
    return $sf_{high}$
end function
```

5. Results

We tested the algorithms with a simulation. In a region of 1000x1000, we randomly generate a number of circles. The radii of the circles grow linearly with the scaling factor. They grow at a fixed rate. The growth rate for each circle is also a randomly generated value between 1 and 10.

We compare four approaches:

- Square Full: computes sf_{full_sq}. We try 18 different angles between 0 degree and 90 degree in 5 degree increments.
- Circle Pairwise: computes sf_{pair_cir}.
- Circle Full: computes sf_{full_cir} with the initial sf_{low} value equal to 0.
- Circle Full with Init: computes sf_{full_cir} with the initial sf_{low} value being the max of sf_{full_sq} and sf_{pair_cir}.

In Figure 10, we show the scaling factor results from the four experiments. They are normalized to the accurate result (”Circle Full” and ”Circle Full with Init”). ”Square full” is a little worse than ”Circle Pairwise”, they both get better as the number of circles increases, getting to 99.9% of the last two for 1000 circles.

In Figure 11, we show the time needed to compute with the four methods. It appears that both ”Square Full” and ”Circle Pairwise” are fast, especially when the number of circles gets large. ”Circle Full” is the slowest, which is not surprising. With ”Circle Full with Init”, we

Algorithm 2 Compute sf_{pair_cir}

```
1: function ComputeSFCirclePair($c$, $r$, $n$)
2:     $cir\_pairs \leftarrow GenerateCirclePairs(c, n)$
3:     $sf_{low} \leftarrow 0$
4:     $sf_{high} \leftarrow HIGH\_VALUE$
5:     while $sf_{high} - sf_{low} \geq \delta$ do
6:         $sf_{mid} \leftarrow (sf_{low} + sf_{high})/2$
7:         $SortByDistOverGrowth(cir\_pairs)$
8:         $if\_overlap \leftarrow true$
9:         for $i \leftarrow 1, length(cir\_pair)$ do
10:            if $CheckOverlap(cir\_pairs[i]) = false$ then
11:                $if\_overlap \leftarrow false; break;$
12:            end if
13:        end for
14:        if $if\_overlap = true$ then $sf_{high} \leftarrow sf_{mid}$
15:        else $sf_{low} \leftarrow sf_{mid}$
16:        end if
17:    end while
18:    return $sf_{high}$
19: end function
```

are able to reduce the running time by 30% to 50%. For 1000 circles, the running time is almost cut to half. With "Square Full", the computation is 1000 times faster than "Circle Full" while getting to 99.9% of the accurate result.

6. Conclusion

In conclusion,

7. Acknowledgments

We would like to thank the science Teacher Mr. Hatten for introducing me to research and helping us with finding new ideas. We also would like to thank Long Island Science and Engineering Fair (LISEF) for providing the opportunity to get access to various research topics, which inspired this work.

References

[1] Riheng Jia, Jinhao Wu, Jianfeng Lu, Minglu Li, Feilong Lin, and Zhonglong Zheng. Energy saving in heterogeneous wireless rechargeable sensor networks. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 1838–1847. IEEE, 2022.

[2] Riheng Jia, Jinhao Wu, Xiong Wang, Jianfeng Lu, Feilong Lin, Zhonglong Zheng, and Minglu Li. Energy cost minimization in wireless rechargeable sensor networks. *IEEE/ACM Transactions on Networking*, 31(5):2345–2360, 2023.

[3] Ed Helly. Über mengen konvexer körper mit gemeinschaftlichen punkte. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923.

[4] Heinrich Dörrie. *100 Great Problems of Elementary Mathematics: their history and solution.* Courier Corporation, 1965.

Email address: `andyzhuang2010@gmail.com`

Algorithm 3 Compute sf_{full_cir}

```
 1: function ComputeSFCircleFull($c$, $r$, $n$, $sf_{init\_cir}$)
 2:     $cir\_states \leftarrow GenerateCircleState(c, n)$
 3:     $sf_{low} \leftarrow sf_{init\_cir}$
 4:     $sf_{high} \leftarrow HIGH\_VALUE$
 5:     while $sf_{high} - sf_{low} \geq \delta$ do
 6:         $sf_{mid} \leftarrow (sf_{low} + sf_{high})/2$
 7:         $UpdateCircleState(cir\_states, sf_{mid})$
 8:         $RemoveEnclosingCircles(cir\_states)$
 9:         $SortByRadius(cir\_states)$
10:         $if\_overlap \leftarrow true$
11:         for $i \leftarrow 1, length(cir\_states)$ do
12:             $RemoveArcSubsumingNeighbors(cir\_states[i])$
13:             $neighbor\_pairs \leftarrow GenerateNeighborPair(cir\_states[i])$
14:             for $j \leftarrow 1, length(neighbor\_pairs)$ do
15:                 if $CheckOverlapCenterDist(neighbor\_pairs[j]) = false$ then
16:                     if $CheckOverlapRadicalCenter(neighbor\_pairs[j]) = false$ then
17:                         $if\_overlap \leftarrow false; break$;
18:                     end if
19:                 end if
20:             end for
21:             if $if\_overlap = false$ then break;
22:             end if
23:         end for
24:         if $if\_overlap = true$ then $sf_{high} \leftarrow sf_{mid}$
25:         else $sf_{low} \leftarrow sf_{mid}$
26:         end if
27:     end while
28:     return $sf_{high}$
29: end function
```

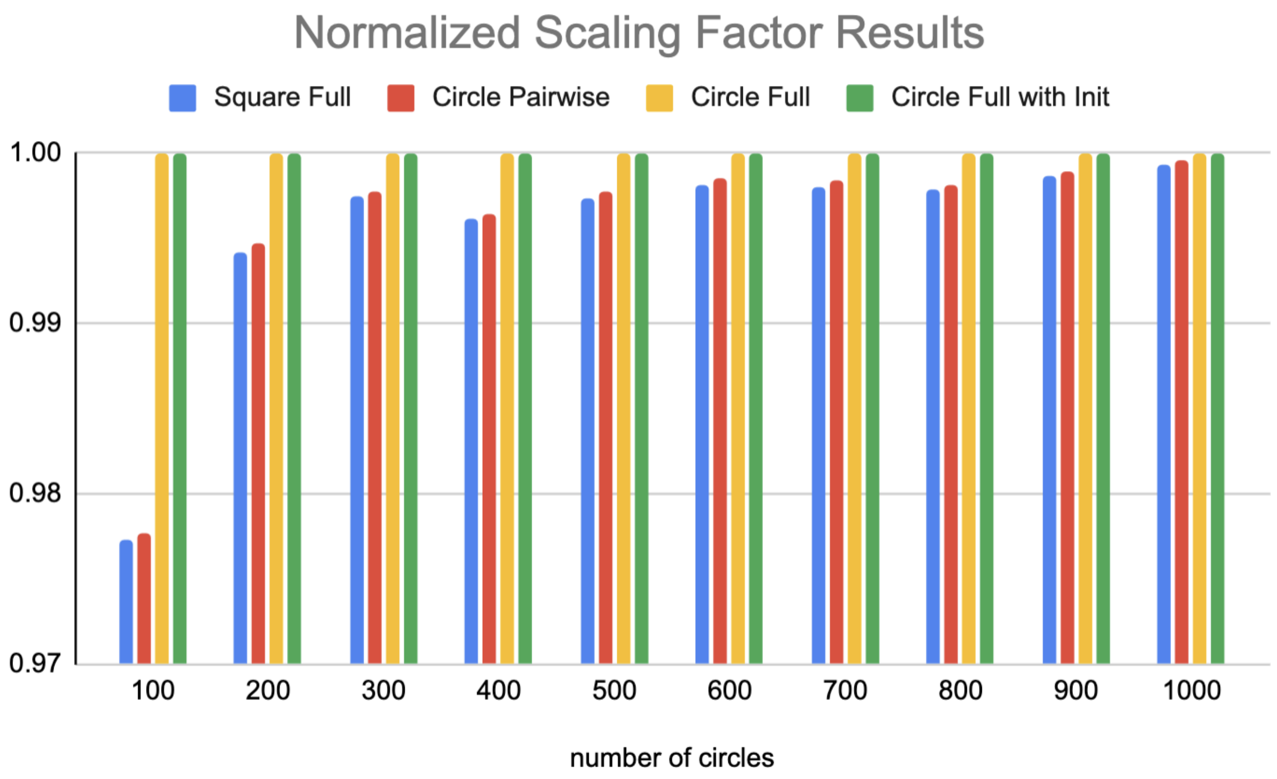


Figure 10. Comparison of normalized results.

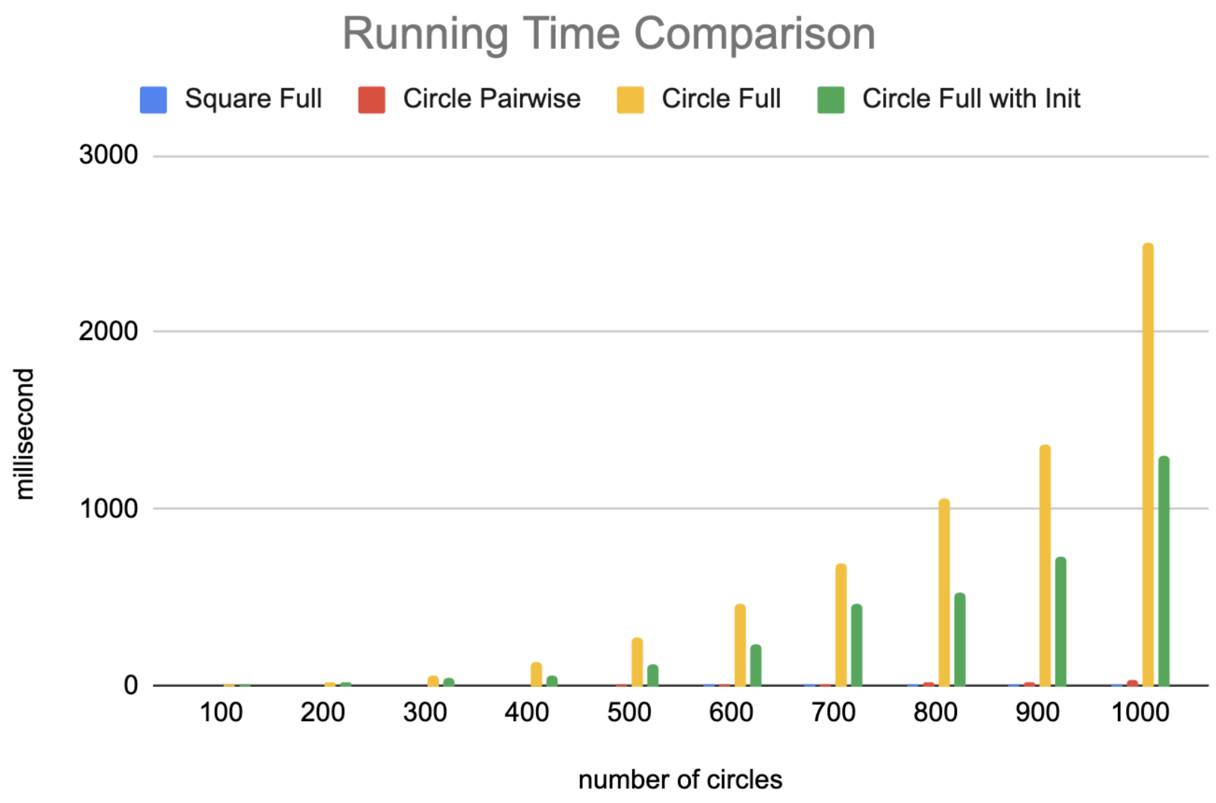


Figure 11. Comparison of running time.