An Exploration of Optimized Circle Packing

Samuel Rosen, Angel Diaz-Cedeno, Jonathan Gjeltema

May 8, 2018

1 Abstract

Background

The general idea for circle packing is to "Find the minimum radius r_0 of a circular container so it can pack N arbitrary sized and nonoverlapping circles" [3]. That's not to say all circle packing is restricted to just circular containers. In a few ideal scenarios there are proven optimal packings, for instance in the case of there being 7 circles with equal area the best possible way to pack them into a circlular container is in a hexagonal manner. A few of these scenarios can also be found on `www.packomania.com` given with their calculation forms. Additionally there are some people who have dedicated most of their lives to studying and solving circle packing problems, such as a Ken Stephenson who owns the rights to the *CirclePack* program which can be found at `http://www.circlepack.com/software.html`.

Applications

Some uses for circle packing include the production of foam, and origami. Both foam production and origami utilize circle packing in somewhat different ways. Origami involves using circles on the different surface shapes and appendages, while foam is primarily constructed volumetrically by spheres. Applications in manufacturing are rampant as any problem which requires a large amount of circular objects to fit in a tight space would require an optimal circle packing for maximum efficiency.

Goals

Our goal is to design an algorithm to pack circles of arbitrary sizes. The algorithm will be relatively simple to understand and also pack circles in an efficient, but obviously not completely optimal way. The algorithm demonstrated is called Snake Packing and allows for a quick way to pack circles with an average density of about 70%. The source code is available at `https://github.com/GreenVars/Circle-Packing`.

2 Introduction

As previously stated, our goals are to pack circles of arbitrary sizes in a specific sequence. We will accomplish this by implementing the Snake Packing algorithm. This algorithm will place circles in a spiral like motion, of which we will analyze the density and efficiency of area covered. By generating images and efficiency quantities using computer programming, we will decide whether or not this algorithm effectively covers the area in a given space.

Theory

When working with circle packings, there are many different variations of optimal solutions that a particular set of circles can accomplish. Generally speaking in regards to all sets, if the number of circles exceeds 30 then the optimal solutions become difficult or (to date) impossible to generate and/or prove [3]. One way in which a method can by applied is seen within our algorithm. Our algorithm will give a specific outcome that results in packing N circles inside a circular contained with radius r_0. This is put forth in [3], which outlines these conditions and determines the minimal radius of the container, r_0; this specific problem is then computed with the following constraints:

$$\sqrt{x_i^2 + y_i^2} + r_i < r_0$$
$$i = 1, ..., N$$
$$r_i + r_j \leq d_{ij}$$
$$i, j = 1, ...N$$

NP-HARD

Circle packing is equivalent to the Bin-Packing problem. The Bin-Packing problem says "Suppose we have n objects, each of a given size, and some bins of equal capacity...[Assign] the objects to the bins, using as few bins as possible". [1] Because the problems are equivalent and Bin-Packing is **NP-HARD**, Circle Packing is also **NP-HARD**. **NP-HARD** problems can not be solved in polynomial time; simply put, finding the proven most optimal layout of circles is extremely difficult and approximate solutions are used to get as close to optimal as possible in a reasonable amount of time.

3 Approaches to Circle Packing

Deterministic Approaches

One simple method referred to as "square shelf packing" is to treat each circle as a square, and then pack the squares in shelving manner. The squares are

then organized into squares that are stacked decendingly in respect to their perimeter. When the total areas of the circles in the shelf sum to over 1, a new shelf if formed above the current shelf and the process is repeated [5]. Furthermore, an additional approach is to take all the circles and place them in some manner such that they are far apart and do not intersect. The next step is through some iterative process, bring the circles together and terminate when each step has negligible improvement[6].

Lastly, Ken Stephenson proposes the linearization method in which an iterative process switches between estimating both the locations of circle centers as well as the radii of the circles. This results in a highly efficient packing but requires complex skills in analytical geometry [2].

Optimization and Machine Learning

In regards to optimization, its is pointed out in [3] that from a purely algebraic standpoint, it is difficult to generate a truly optimized circle packing once the number of circles exceeds 10. This is why global optimization packages are used to computationally generate an optimized solution. Optimization software packages allow for easy variations under distinct constraints [3]. In regards to the software packages, intuitivley, as the number of circles increases, the number of constriants at a faster rate so the level of optimization becomes less and less accurate.

However, as demonstrated in [4], an approach to deal with placing constraints on a set of circles and its container is the "repair" method. This method utilizes a "repair" function that evalutes a specific movement, and if this movement breaches the constraints it is then subsitituted by a movement that does not. This results in a more efficient optimization despite the number of constraints that is placed on the set [4].

4 Snake Packing

The shelf packing algorithm mentioned earlier can be generalized to packing in tiers of circles. This algorithm takes one input, a set of circles with known radii. It is very simple, first placing the largest circle in the center and then placing the second largest on a tier that is the sum of the first and second radii. Subsequent circles are then placed on the first tier until no circles can fit anymore, then the next tier is made and so on until all circles are exhausted (Figure 1). The tiered packing approach shares the same weaknesses as the shelf packing approach. A tiered approach leads to needless space between each tier which could instead hold circles. Circles between tiers are also not tangent, a quality often desired in circle packing. However, both are easy to implement as they only require simple geometry.

We can improve upon the tiered approach by minimizing the area between tiers by making each circle added onto the packing be tangent to a specific circle as well as its previous circle. This will also increase the number of tangencies

giving a desirable packing that is also compact. Our algorithm is called Snake Packing because, one can follow the circles from the largest in the center to the smallest at the end in a pattern similar to that of a curled snake.

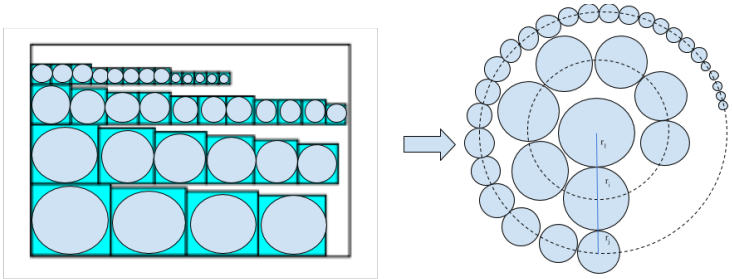
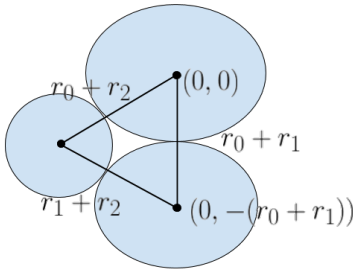


Figure 1: The first circle is placed at the origin and the second circle defines the next tier to place circles on. Each circle is placed such that its center is on the circle defining the tier. After a tier fills up, another tier is made that is the as far the sum of the previous tiers length and the next incoming radii. A shelf and tier approach are shown for the same circles.

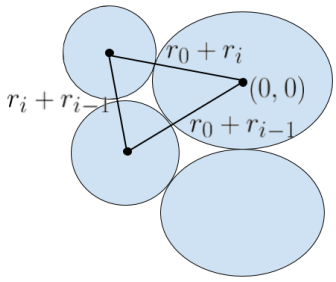
Algorithm 1 Snake Packing

S: a set of all circles sorted in order of descending radius
procedure SnakePack(S)
    r_i: radius of S_i where r_0 is $max(r_i)$
    Place S_0 at the origin
    Place S_1 at $(0, r_0 + r_1)$ ▷ Will be tangent to S_0
    $anchor \leftarrow S_0$
    for each circle S_i in S **do** ▷ $i \geq 2$ since $i = 0, 1$ already placed
        Place S_i such that S_i is tangent to S_{i-1} and $anchor$
        if S_i overlaps a placed circle **then**
            $anchor \leftarrow$ circle S_i overlaps
            repeat placement of S_i until placed

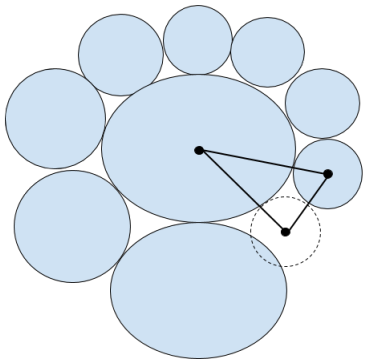
Steps Visualized



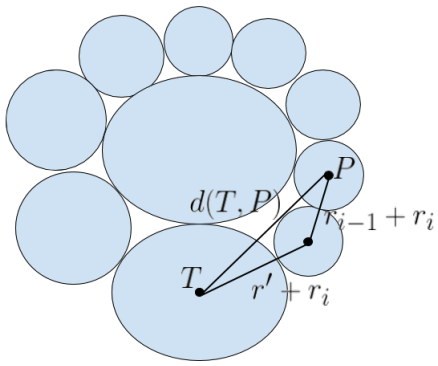
1. After sorting the circles by descending radius, place the first two largest circles in the positions shown. The location of the next circle's center is where that circle is tangent to both circles. This tells us it must be a distance $r_0 + r_2$ from the origin and distance $r_1 + r_2$ from the previous placed circle. This forms a triangle where all 3 side lengths are known; the Law of Cosines can be used to solve for the next point trivially since the other 2 points are known.



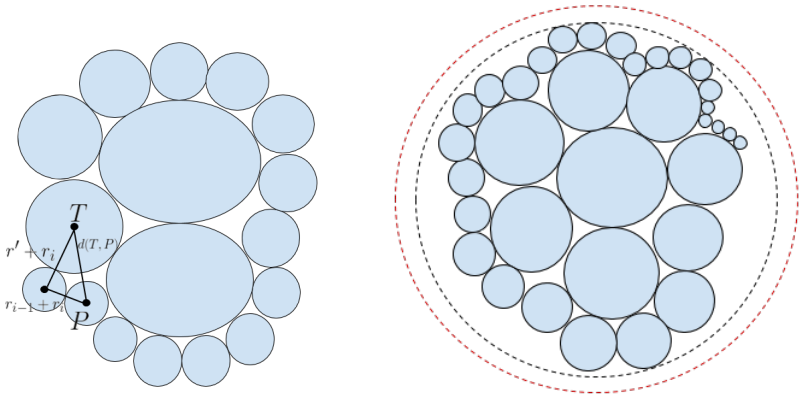
2. The above process can be generalized to solve for the placement of each successive circle, where the i^{th} circle must be $r_{i-1} + r_i$ distance away from the previous circle and $r_0 + r_i$ distance from the origin. The Law of Cosines again gives us the next point.



3. Eventually going through the algorithm, a point will be found that fits the criterea of tangency, but will cause a placed circle to intersect with an already placed circle. This is expected and is addressed below.



4. Simply change the main circle of tangency to the circle that was intersected with, shown above as T. Because the circles are of decreasing size, a placed circle should only intersect with one circle circle if it is in an invalid position. Repeat the process of placement with the new tangent circle so the next circle is tangent to both T and the previous circle P. The radius of the tangent circle will now be referred to as r'. The distance between the circle of tangency and the previous circle is known since both the center points of those circles are known. Once again, since all side lengths are know, the Law of Cosines gives us the next placement point.



5. Continue this process as needed, changing the circle of tangency when a new intersection is found. Eventually all circles will be placed and a somewhat optimal packing is produced. The graphic on the right shows an estimated smallest bounding circle given by this algorithm (in black). The outside circle (in red) is the estimated smallest bounding circle given by the tiered shelving algorithm.

Density and Results

Density is defined as the ratio of area the circles in the packing occupy to the area of the minimum bounding circle (radius R). Density shows how well circles are packed in the minimum bounding circle. The maximum bound of density is $\frac{\pi}{\sqrt{12}} \approx 0.9069$. [5]

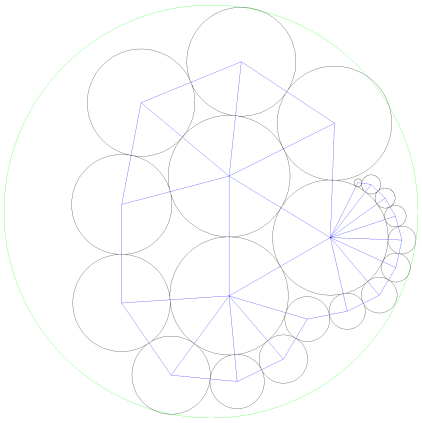
$$\rho = \frac{\sum_{i=0}^{n} \pi r_i^2}{\pi R^2} = \frac{\sum_{i=0}^{n} r_i^2}{R^2}$$

Below is a table of the average density from 30 samples of circles with random radii from 1 to a given max. Snake Packing appears to perform better as the max possible radius increases and also as the number of circles increases.

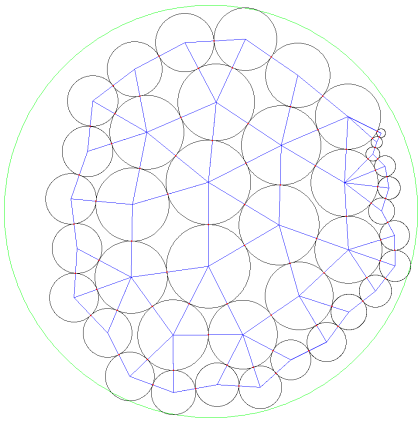
Table 1: Snake Packing Results

	Max radius generated			
n circles	20	40	60	80
20	0.683	0.682	0.687	0.676
40	0.699	0.695	0.701	0.698
60	0.721	0.712	0.717	0.713
80	0.728	0.728	0.724	0.722

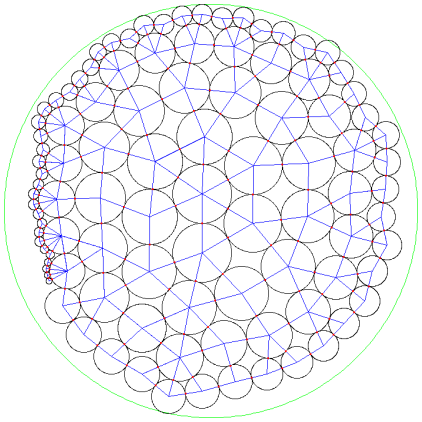
Below are various outputs from the algorithm for various scenarios. The green circle represents an approximate minimum bounding circle. The blue lines represent a the edges of an undirected graph where circles are neighbors to circles they are tangent to. Red dots represent points where circles are tangent to each other. Notice that when the circles are of uniform size, the circles are placed in a hexagonal grid. On each example you can follow the snake like pattern from the center circle to the last circle by starting at the center circle moving down and then counter-clockwise.



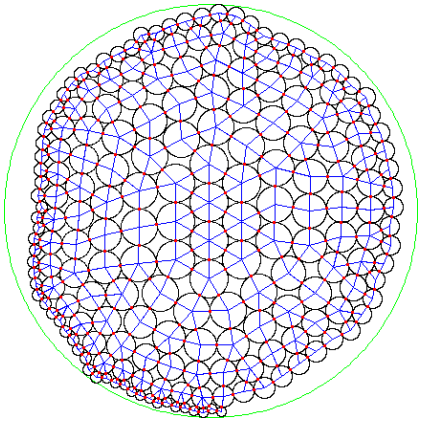
(a) 20 random circles with $r = 10...250$, $\rho = 0.683$



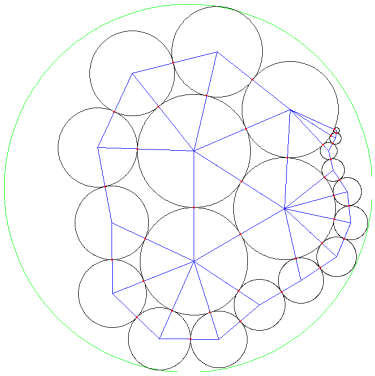
(b) 40 random circles with $r = 10...100$, $\rho = 0.721$



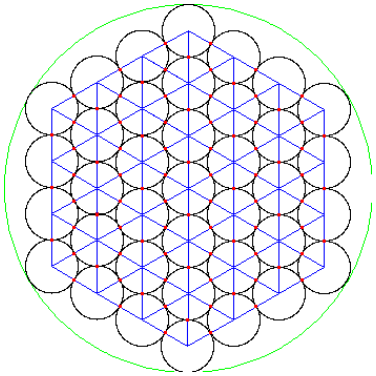
(c) 100 random circles with $r = 5...50$, $\rho = 0.7465$



(d) 200 random circles with $r = 5...20$, $\rho = 0.737$



(e) All circles with radius $r = 5, 10, 15...100$, $\rho = 0.679$



(f) 37 Circles of radius 25, $\rho = 0.755$

5 Conclusion

Snake Packing performs well and usually returns a density in the range of 70% to 75%. As opposed to other algorithms, ours can take a high number of circles and still finish in a reasonable amount of time since each additional circle is placed based on the last placed circle. Snake Packing also follows a hexagonal pattern when all the circles are of equal size, giving an optimal density. One can easily see how Snake Packing could be more optimal but is not. For example, the smaller circles on the edge of the figure could be placed in the gaps between the large circles in the center. Potentially, Snake Packing could be a start step for another circle packing algorithm that corrects these mistakes to make a more optimal packing.

Snake Packing is very simple to implement and requires no mathematics beyond simple geometry. The most strenous math required is using the Law of Cosines to compute the location of the next placed circle. As opposed to other methods, which require advanced optimization, numerical solutions, and computational geometry. We acknowledge more advanced methods create more optimal solutions at the cost of a longer runtime and difficult implementations. When looking for the most optimal solutions to pack a set of circles, these advanced algorithms should be used. However, if one requires a fast implementation that can be used to do many packings for a high number of circles, Snake Packing can be a good fit if a somewhat optimal solutions is all that is required. As a very simple algorithm, our algorithm may be used to as a comparison to other algorithms for packing circles of arbitrary sizes; if a devised algorithm is more complicated than ours yet can not outperform it, then it is likely needs improvement.

References

[1] *Bin-Packing*, pages 426–441. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[2] Charles Collins Gerald L. Orick, Kenneth Stephenson. A linearized circle packing algorithm. *Computational Geometry*, 64, 2017.

[3] Janas D. Pintr Ignacio Castillo, Frank J. Kampas. Solving circle packing problems by global optimization: Numerical results and industrial applications. *European Journal of Operational Research*, 191, 2008.

[4] Felix Calderon Juan Flores, Jose Martinez. Evolutionary computation solultions to the circle packing problem. *Soft Computing*, 4, 2015.

[5] Flvio K. Miyazawa. Approximation algorithms for circle packing. So Paulo School of Advanced Science on Algorithms, Combinatorics and Optimization, July 2016.

[6] K.J. Nurmela P.R.J. stergrd R.L. Graham, B.D. Lubachevsky. Dense packings of congruent circles in a circle. *Discrete Mathematics*, 181, 1998.