

**Gebze Technical University
Computer Engineering**

CSE 424- 2019 Autumn

HOMEWORK 2 REPORT

**AKIN ÇAM
151044007**

Course Assistant:
Fatih Erdoğan Sevilgen

Algoritmalar Java ile yazılmış. IntelliJ Idea Ortamında Geliştirilmiştir.

- Brute Force Algoritması

Circle Packing problemi için “Brute Force Algoritması” 3 method ile gerçekleştirilmiştir. İlk method verilen bir input arraydeki her iki circle arasındaki mesafeyi recursive hesaplamakta ve toplayarak sonucu döndürmektedir.

```
CalculateDistance (arr, size=0); → inputlar circles array ve recursive için size(ilk durum=0)
    Sqrt = 0;
    if (size > arr.size) {      → tüm circlelar dolaşıldığında dur.
        Return sqrt;
    }
    else {                      → Pisagor teoremi tersten uygulanır ve iki
                                çember arasındaki mesafe bulunur.
        Sqrt = Math.sqrt(radius1^2+radius2^2) - Math.sqrt(radius1^2-
        radius2^2)
        Size ++ ;

        Return sqrt+ CalculateDistance(arr,size) → bir sonraki circle ile devam eder
    }

}
```

Time Complexity →

$T(n) = T(n-1)$, 0 için → $O(1)$

$T(n) = T(n-1) + c \rightarrow c+c+c+c+c..... = nc$

$O(n)$

İkinci method permutation recursive bir methoddur. Verilen input arraydeki elemanların tüm olasılıklarını bir 2 boyutlu arraye atamaktadır.

Time Complexity → tüm olasılıkları dolaştığı için $n!$ Dir.

Uçüncü method permutasyon listesindeki elemanların oluşturduğu width leri karşılaştırıp en küçük olanı döndürmektedir. Bir for loop içinde tüm olasılıkları karşılaştırarak en optimal değeri bulur.

Time Complexity → ilk method $O(n)$ + ikinci Method $O(n!)$ + for loop → $O(n)$ olmak üzere çalışma zamanı $O(n!)$ dir.

- Greedy Algoritması

Circle Packing problemi “Greedy Algoritması” na göre şöyle gerçekleşmiştir:

Greedy Algoritması bir problemi çözerken bir sonraki adıma ileriye dikkate almadan o andaki en iyi değere göre karar vermektedir. Bir circle packing probleminde mesafeyi en küçük bulmak için Büyük yarıçaplı çemberler arasına küçük yarıçaplı çemberler koyulmalı ve bu iki büyük çember arasına kalan boşluktan faydalanılmaktadır. Ayrıca başlangıç ve bitiş çemberleri çok büyük değerler olmamalıdır. Bu problemde optimal çözüme ulaşmak için ilk olarak input listesi küçükten büyüğe sıralanmaktadır. Java merge sort kullanır ve çalışma zamanı $O(n \log n)$ dir. Greedy de başlangıç durumu için çözüm arrayinin ilk elemanına ortanca değerler set edilir. Bu işlemden sonra greedy algoritma çalışırken her bir çember için bir büyük bir küçük olacak şekilde karşılaştırma yapılır ve output arraye eklenir. Bu durum çalışma zamanı için $O(n^2)$ dir.

Time Complexity $\rightarrow O(n^2) + O(n \log n) \rightarrow O(n^2)$

- Variable Neighbour Search Algoritması

Circle Packing problemi “Variable Neighbour Search Algoritması” ‘na göre şöyle gerçekleşmiştir:

Komşu array sayısı belirlenir.

Verilen input array den rastgele bir çözüm belirlenir.

Algoritmanın durma koşu belirlenir. Bu işlemlerden sonra algoritma durma koşuluna kadar devam edecek ve buradaki her iterasyonda komşu array sayısı kadar bir döngüde işlemler yapacaktır.

While(Durma koşulu

While($k < k_{max}$

İlk olarak Shaking işlemi yapılmaktadır. Burada en iyi çözümden rassal olarak bir komşuluk kümesi oluşturulur ve buradaki k. Eleman döndürülür.

Local search kısmında verilen bu komşuluk kümesi ve shakingden gelen eleman ile bu komşuluk kümesindeki en iyi çözüm bulunur.

Bu çözüm şu ana kadar bulunan çözümden daha iyi ise en iyi çözüm güncellenir ve $k=0$ olarak değiştirilir. Aksi takdirde k artırılır ve bir sonraki durumdan devam edilir.

Pseudocode Aşağıdaki gibidir→

Başlangıç

- (1) N_k , Komşuluk yapıları kümesini seç ($k = 1, \dots, k_{max}$)
- (2) π , Başlangıç çözümünü bul
- (3) Durma koşulunu belirle

Durdurma koşulu sağlanıncaya kadar aşağıdaki adımları tekrarla

- (1) $k=1$ olarak al
- (2) $k=k_{max}$ oluncaya kadar aşağıdaki adımları tekrarla
 - (a) Karıştırma: π 'in, k . komşuluğundan, rassal olarak bir π' noktası üret
($\pi' \in N_k(\pi)$)
 - (b) Yerel arama: π' başlangıç çözümüne yerel arama metodunu uygula ve elde edilen yerel en iyi çözümü π'' olarak belirle
 - (c) Hareket et ya da dur: eğer bu yerel en iyi daha iyi bir sonuç ise $\pi = \pi''$ yap ve adım aynı komşulukta aramaya devam et. Diğer durumda, $k = k+1$ yap.

Time Complexity → İlk olarak durma koşuluna kadar çalışacaktır. * ikinci döngüde komşuluk kümesi sayısı kadar çalışır. * Son olarak burada shaking linear zamanda, local search linear zamanda yani n zmaanda çalışmaktadır. Bu durumda çalışma zamanı

→ $O(\text{stoppingCondition} * k_{Max} * \text{NeighbourSize})$;

İnput sayısı az olduğunda ve komşuluk sayısı çok olduğunda normalde çalışması gerekenden daha çok çalışacak ve çalışma zamanı artmaktadır.

- Simulating Annealing Algoritması

Circle Packing problemi “Simulated Annealing Algoritması” na göre şöyle gerçekleşmiştir:

İlk olarak Bir sıcaklık değeri iterasyon sayısı ve sıcaklık azalma oranı belirlenir.

Verilen input array den rastgele bir komşuluk uzayından bir çözüm belirlenir. Daha sonra sıcaklık uygun bir dereceye ulaşana kadar işlem devam edecektir. İterasyon sayısı boyunca ilk olarak bir çözüm uzayı oluşturulur. Bu uzayda bir localSearch yapılır ve o komşuluk uzayında en iyi değer belirlenir. Eğer bu değer su anki en iyi değerden daha iyiye bu değer güncellenir. Aksi takdirde bir random değer belirlenir. Bu değer iki değer arasındaki farkın exponential değerinden küçükse şuanki değer yeniden güncellenir. Şu anki değer bestSoFar dediğimiz algoritma çalışması zamanındaki en iyi değer ise bu değer kaydedilir. Sıcaklık belirli bir miktar azaltılır.

Time Complexity → Sıcaklık $O(\log n)$ olarak değişir. Her sıcaklıkta $O(n)$ şeklinde iterasyon oluşmaktadır. Bu iterasyonda local search ya da komşuluk uzayı linear zamanda çalışmaktadır. Bu durumda çalışma zamanı →

$O(\text{iterasyonSayısı}^2 * \log(T))$.

Aşağıdaki pseudocode a göre gerçekleşmiştir →

```
1   $S \leftarrow$  an initial solution;
2   $S^* \leftarrow S$ ;
3   $iter \leftarrow 0$ ;
4   $T \leftarrow T_0$ ;
5  while ( $T > T_f$ ) do
6    while ( $iter < SA_{max}$ ) do
7       $iter \leftarrow iter + 1$ ;
8      PICK (a random neighbor  $S' \in \psi(S)$ );
9       $\Delta \leftarrow f(S') - f(S)$ ;
10     if  $\Delta < 0$  then
11        $S \leftarrow S'$ ;
12       if  $f(S') < f(S^*)$  then
13          $S^* \leftarrow S'$ ;
14       end
15     else
16       TAKE ( $x \in [0, 1]$ );
17       if  $x < e^{-\Delta/T}$  then
18          $S \leftarrow S'$ ;
19       end
20     end
21   end
22    $T \leftarrow \alpha * T$ ;
23    $iter \leftarrow 0$ ;
24 end
25 Return  $S^*$ ;
```

- Iterated Local Search Algoritması

Circle Packing problemi “Iterated Local Search Algoritması” na göre şöyle gerçekleşmiştir:

Verilen input array den rastgele bir çözüm uzayı belirlenir. Çözüm uzayından localSearch ile bir çözüm belirlenir. Daha sonra bu algoritma iterasyon sayısı kadar çalışmaktadır. İlk olarak mutate yapmaktadır. Verilen listeyi karıştırarak listeyi mutasyona uğratarak bir komşuluk kümesi oluşturur. Bu kümeden local search ile en iyisini seçer. Eğer değer bir optimum değer ise bestSoFar güncellenir. Bu sayede minimum width distance bulunur.

Time Complexity → Iterasyon Sayısı * loop içi linear zamanda çalışmaktadır.

$O(\text{iterationCount} * \text{NeighbourSize}) = O(n^2)$ olmaktadır.

Aşağıdaki pseudocode a göre gerçekleşmiştir →

```
procedure Iterated-Local-Search( $n_{max} : \mathbb{N}$ ) :  $S$ ;  
begin  
  Create starting solution  $s$ ;  
   $s := \text{Local-Search}(s)$ ;  
   $n := 0$ ;  
  repeat  
     $s' := \text{Mutate}(s)$ ;  
     $s' := \text{Local-Search}(s')$ ;  
    if  $f(s') < f(s)$  then  $s := s'$ ;  
     $n := n + 1$ ;  
  until  $n = n_{max}$ ;  
  return  $s$ ;  
end;
```

- Branch And Bound Algoritması

Circle Packing problemi “Branch And Bound Algoritması” na göre şöyle gerçekleşmiştir:

Bir input array alınır. Bu input array den ilk olarak bir 2 boyutlu bir her iki çember arasındaki mesafeyi tutan bir array oluşturulur. Her satırda bulunan minimum değer alınır ve kontrol width oluşturulur. Bu işlemten sonra matrix in 1. Seviyesinden başlayarak o seviyenin minimum değeri bulunur. Buradaki değer kontrol değerinden çıkarılır ve bulunan çemberin değeri eklenir. Bu değer en iyi değerden daha iyiyse güncellenir. Bu tüm dailerin ilk sırada olduğu şekilde tekrarlanır. Sonuç bir sıralamanın indexleridir.

Time Complexity → Daire Sayısı *. Helper Fonksiyon(n^2) → $O(n^3)$ olmaktadır.

- Ant Colony Optimization Algoritması

Circle Packing problemi “Ant Colony Optimization Algoritması” na göre şöyle gerçekleşmiştir:

1) İlk olarak parametreler belirlenmektedir:

----Daha iyi sonuçlar için alpha değeri beta değerinden küçük olarak seçilmelidir.

----Feromon buharlaşma hızı artınca optimal çözüme ulaşamadığı görülmüştür.

---Feromon oranı azaldıkça buharlaşma hızı daha etkili olduğu için optimal çözüme --- ulaşamadığı görülmüştür.

---Ant Factor artıkça algoritmanın çalışma süresi artmaktadır fakat daha optimal bir sonuca ulaşmaktadır.

---Random factor 0 ile 1 arasında olduğunda daha optimal sonuç vermektedir. Çünkü random olarak bir daire yerleştirmesi sonucunda feromon etkisi sonucu seçilen optimal daire seçilememektedir.

2) İlgili Pathlerin feromon değerleri ilklendirilir. Ve karıncalar için bir ilk daire seçilir.

3) moveAnts = bu işlemde 3 olasılıkla bir daire seçilir. İlk olarak random faktörü sağlıyorsa daha önce seçilmemiş bir daire seçilir. İkinci olarak seçilebilme olasılığı hesaplanır. Burada Ant colony optimizasyon formülü kullanılır:

An ant will move from node i to node j with probability

$$p_{i,j} = \frac{(\tau_{i,j}^\alpha)(\eta_{i,j}^\beta)}{\sum (\tau_{i,j}^\alpha)(\eta_{i,j}^\beta)}$$

where

$\tau_{i,j}$ is the amount of pheromone on edge i,j

α is a parameter to control the influence of $\tau_{i,j}$

$\eta_{i,j}$ is the desirability of edge i,j (typically $1/d_{i,j}$)

β is a parameter to control the influence of $\eta_{i,j}$

Son olarak seçilmeyen bir daire seçilir.

4) Daha sonra ilk olarak daha önce gezilen daire listelerinin feromonları buharlaştırılır. Ve Width kalitesine göre gezilen yerlere yeniden feromon salgılanır.

5) En iyi değerler güncellenir. Bu işlemler iterasyon sayısı kadar devam eder.

Time Complexity → İterasyon Sayısı * $O(\text{MoveAnts}(n^2), \text{updatePathPheromone}())(O(n)*n), \text{updateBest}() (O(n))$

→ İterasyon sayısı * $O(n^2)$

- Particle Swarm Optimization

Circle Packing problemi “PSO Algoritması” na göre şöyle gerçekleştirilmiştir:

- 1) İlk olarak bir popülasyon oluşturulur.
- 2) Şu anki en iyi güncellenir.
- 3) İterasyon sayısı boyunca
 - a. Pbest ve gbest değeri ile mevcut popülasyonlar karşılaştırılır. Indexi farklı olanlar c1, ve c2 değerleri ile bir swapOperator oluşturulur.

$$V_{id} = V_{id} \oplus \alpha * (P_{id} - X_{id}) \oplus \beta * (P_{gd} - X_{id}),$$

- b. Bu c1 c2 değerleri bir random sayıyla karşılaştırılır. Sağladığı durumda popülasyonlarda değişiklik yapılır ve en iyi değer tekrar kontrol edilir.

4) **Time Complexity** → İterasyon Sayısı * $O(\text{Daire sayısı} * O(n))$

Algoritmaların Testleri Ve Parametreleri ile ilgili Yorumlar:

Brute Force Algoritması tüm olasılıkları test ettiği için daire sayısı arttıkça optimal sonuca ulaşmak zaman alıyor. 10 daireye kadar en doğru sonucu veriyor fakat daha yüksek daire sayıları için sonuca ulaşmak güçleşiyor.

Branch And Bound Algoritması brute force algoritması gibi olasılıkların çoğunu test ediyor. Fakat optimal sonuçlarda bazen brute force dan geride kalabiliyor. 10 daireye kadar en doğru sonucu veriyor fakat daha yüksek daire sayıları için sonuca ulaşmak güçleşiyor.

Greedy Algoritması daire sırasını algoritma çalışma zamanında anlık kararlara göre oluşturduğu için optimal sonuçları her zaman vermemektedir. Düşük sayılı inputlar daha sonuç vermekle birlikte daire büyüklükleri arasındaki fark arttıkça optimal sonuçtan uzaklaşmaktadır.

Iterated Local Search algoritmasında iterasyon sayısı çok önem kazanmaktadır. Input sayısı büyük olduğunda iterasyon sayısı az ise optimal sonuca ulaşmayabilir.

Simulating Anneling Algoritmasında sıcaklıktaki düşüş hızı optimal sonuca ulaşmayı çok etkilemektedir. Büyük inputlar için optima sonuca ulaşmayı yavaşlatmaktadır. Cooling Rate in büyük olması algoritmanın çabuk bitmesine sebep olur. Populasyon sayısı ve cooling rate in orantılı olması gerekmektedir. Eğer cooling rate büyük ise popülasyon sayısı daha büyük olmalıdır ki daha çabuk optimal sonuca ulaşılabilin.

VNS de oluşturulan komşuluklar önem kazanmaktadır. Bu yüzden popülasyon sayısı büyük inputlar için önem kazanmaktadır. Fakat büyük inputlar için hem popülasyon sayısı hem de durma koşulu yüksek olursa algoritma çalışma zamanı artmaktadır.

PSO da önemli olarak c1 ve c2 değerleridir. En iyi değere ulaşırken c1 ve c2 değerlerinin çok büyük olması random sayıyı sağlamayacak ve en iyi değere ulaşması zor olacaktır. Populasyon sayısının artması optimal sonuç için iyi olsada büyük inputlar için popülasyon sayısı ve iterasyon sayısı dengeli olmalıdır.

Ant Colony Optimizasyonda daha iyi sonuçlar için alpha değeri beta değerinden küçük olarak seçilmelidir. Feromon buharlaşma hızı artınca optimal çözüme ulaşamadığı görülmüştür. Feromon oranı azaldıkça buharlaşma hızı daha etkili olduğu için optimal çözüme ---ulaşamadığı görülmüştür. Ant Factor artıkça algoritmanın çalışma süresi artmaktadır fakat daha optimal bir sonuca ulaşmaktadır. Random factor 0 ile 1 arasında olduğunda daha optimal sonuç vermektedir. Çünkü random olarak bir daire yerleştirmesi sonucunda feromon etkisi sonucu seçilen optimal daire seçilememektedir.

TEST SONUÇLARI:

1) Birbilerine Yakın Değerler: 5.0, 6.0, 5.0, 9.0, 3.0, 9.0, 5.0, 7.0, 4.0, 12.0 (Size 10)

```
<<<<<<<<<<<<<<<Variable Neighbour Search Algorithm>>>>>>>>>>>>>>>>
```

Best Circles Combination:

7.0, 5.0, 6.0, 5.0, 9.0, 3.0, 12.0, 4.0, 9.0, 5.0,

bestSoFar VNS: 120.82258890236764

Execution Time: 1047 milisecond

```
<<<<<<<<<<<<<<Simulated Annealing Algorithm>>>>>>>>>>>>>>>>
```

Best Circles Combination:

5.0, 9.0, 4.0, 12.0, 3.0, 9.0, 5.0, 7.0, 5.0, 6.0,

```
bestSoFar Simulated Annealing: 120.70029731846353
```

Execution Time: 1239 milisecond

```
<<<<<<<<<<<<<<<Iterated Local Search Algorithm>>>>>>>>>>>>>>>
```

Best Circles Combination:

5.0, 9.0, 4.0, 12.0, 3.0, 9.0, 5.0, 7.0, 5.0, 6.0,

```
bestSoFar Simulated Annealing: 120.70029731846353
```

Execution Time: 1209 milisecond

```
<<<<<<<<<<<<<<<<Greedy Algorithm>>>>>>>>>>>>>>>>
```

Best Circles Combination:

6.0, 3.0, 12.0, 4.0, 9.0, 5.0, 9.0, 5.0, 7.0, 7.0,

```
bestSoFar Simulated Annealing: 125.42307099598503
```

Execution Time: 5 milisecond

```
<<<<<<<<<<<<<<Brute Force Algorithm>>>>>>>>>>>>>>>>
```

```
bestSoFar Brute Force: 120.70029731846353
```

Best Circles Combination:

5.0, 9.0, 4.0, 12.0, 3.0, 9.0, 5.0, 7.0, 5.0, 6.0,

Execution Time: 24999 milisecond

Execution Time: 6331 milisecond

Execution Time: 835 milisecond

Execution Time: 42 milisecond

Execution Time: 3313 milisecond

3) Test:

7.8, 56.9, 4.6, 7.0, 8.0, 1.0, 77.0, 1.0, 8444.0 (Size=9)

```
<<<<<<<<<<<<<<<Variable Neighbour Search Algorithm>>>>>>>>>>>>>>>>>>
```

Best Circles Combination:

42.0, 345.0, 2.0, 995.0, 2.0, 121.0,

bestSoFar VNS: 665.8348964768932

Execution Time: 755 milisecond

```
<<<<<<<<<<<<<<<Simulated Annealing Algorithm>>>>>>>>>>>>>>>>>>
```

Best Circles Combination:

42.0, 345.0, 2.0, 995.0, 2.0, 121.0,

```
bestSoFar Simulated Annealing: 665.8348964768932
```

Execution Time: 163 milisecond

```
<<<<<<<<<<<<<<<<<<Iterated Local Search Algorithm>>>>>>>>>>>>>>>>
```

Best Circles Combination:

2.0, 2.0, 995.0, 121.0, 42.0, 345.0,

```
bestSoFar Simulated Annealing: 665.8348964768932
```

Execution Time: 253 milisecond

```
<<<<<<<<<<<<<<<<<Greedy Algorithm>>>>>>>>>>>>>>>>
```

Best Circles Combination:

121.0, 2.0, 995.0, 2.0, 345.0, 345.0,

bestSoFar Simulated Annealing: 1418.0860647040267

Execution Time: 4 milisecond

```
<<<<<<<<<<<<<<<<Brute Force Algorithm>>>>>>>>>>>>>>>>
```

```
bestSoFar Brute Force: 665.8348964768932
```

Best Circles Combination:

42.0, 345.0, 2.0, 995.0, 2.0, 121.0,

Execution Time: 8 milisecond

```
Process finished with exit code 0
```

4)Test: (Size=60) iteration=100

```
<<<<<<<<<<<<<<<Variable Neighbour Search Algorithm>>>>>>>>>>>>>>>>
```

Best Circles Combination:

33.2, 4.0, 9.0, 11.2, 2.0, 44.5, 3.0, 55.6, 7.0, 4.0, 77.0, 2.0, 34.5, 1.0, 9.0

bestSoFar VNS: 1028.076521261132

Execution Time: 6404 milisecond

```
<<<<<<<<<<<<<Simulated Annealing Algorithm>>>>>>>>>>>>>>>>
```

Best Circles Combination:

9.0, 4.0, 22.1, 6.0, 6.73, 4.0, 4.0, 1.2, 55.6, 1.0, 77.0, 8.0, 4.0, 3.0, 6.0,

```
bestSoFar Simulated Annealing: 996.2830963018234
```

Execution Time: 4663 milisecond

```
<<<<<<<<<<<<<<<<<<Iterated Local Search Algorithm>>>>>>>>>>>>>>>>>>
```

Best Circles Combination:

6.0, 8.0, 8.8, 5.0, 77.0, 2.1, 24.5, 3.0, 6.0, 1.0, 33.2, 6.0, 9.0, 7.0, 4.0,

```
bestSoFar Simulated Annealing: 996.0982945035756
```

Execution Time: 6538 milisecond

```
<<<<<<<<<<<<<<Greedy Algorithm>>>>>>>>>>>>>>
```

Best Circles Combination:

5.6, 1.0, 99.5, 1.0, 77.4, 1.0, 77.0, 1.0, 77.0, 1.2, 67.3, 2.0, 66.3, 2.0, 5!

```
bestSoFar Simulated Annealing: 875.6731792832812
```

Execution Time: 54 milisecond

```
<<<<<<<<<<<<<<Ant Colony Optimization>>>>>>>>>>>>>>>>
```

```
min width:      1012.9114680225679
```

circles:

1.0 55.6 2.0 3.0 4.0 8.0 3.0 24.5 5.5 3.2 33.2 1.0 66.3 8.0 6.0 6.0 5.6 6.7 9.0 4.0 77.0 9.0 9.

Execution Time: 47735 milisecond

```
<<<<<<<<<<<<<<<<<PSO Optimization>>>>>>>>>>>>>>>>
```

GBEST =1008.5864186129511

[6.0, 1.0, 44.5, 8.0, 7.0, 5.5, 2.1, 9.0, 5.3, 66.3, 4.0, 55.6, 3.0, 33.2, 4.0, 22.1, 5.0, 3.0,

Execution Time: 252 milisecond

Process finished with exit code 0

5)Test: (Size=60) iteration=10

```
<<<<<<<<<<<<<<<Variable Neighbour Search Algorithm>>>>>>>>>>>>>>>>>>
```

Best Circles Combination:

```
33.2, 3.0, 9.0, 2.0, 3.0, 4.0, 4.0, 5.0, 6.0, 6.73, 5.0, 9.0, 4.0, 22.1, 6.0, 8.0, 11.2, 2.0, 77.0, 4.0,
bestSoFar VNS: 1049.4420305004628
```

Execution Time: 966 milisecond

```
<<<<<<<<<<<<<<<<Simulated Annealing Algorithm>>>>>>>>>>>>>>>
```

Best Circles Combination:

```
1.0, 6.0, 9.0, 8.8, 6.0, 4.0, 9.0, 3.0, 5.0, 3.0, 1.0, 99.5, 5.3, 8.0, 6.73, 5.6, 34.5, 22.1, 3.0, 11.2,
bestSoFar Simulated Annealing: 1025.6399651209686
```

Execution Time: 79 milisecond

```
<<<<<<<<<<<<<<<Iterated Local Search Algorithm>>>>>>>>>>>>>>>>>>
```

Best Circles Combination:

```
3.0, 6.0, 2.1, 1.2, 9.0, 6.7, 11.2, 5.0, 1.0, 4.0, 1.0, 34.5, 3.0, 5.0, 9.0, 9.0, 5.0, 4.0, 77.0, 1.0, 5
bestSoFar Simulated Annealing: 1034.5121014895624
```

Execution Time: 167 milisecond

[illegible]

Best Circles Combination:

```
5.6, 1.0, 99.5, 1.0, 77.4, 1.0, 77.0, 1.0, 77.0, 1.2, 67.3, 2.0, 66.3, 2.0, 55.6, 2.0, 54.4, 2.0, 44.5,
bestSoFar Simulated Annealing: 875.6731792832812
```

Execution Time: 71 milisecond

```
<<<<<<<<<<<<<<Brute Force Algorithm>>>>>>>>>>>>>>
```

Best Circles Combination:

5.6, 1.0, 99.5, 1.0, 77.4, 1.0, 77.0, 1.0, 77.0, 1.2, 67.3, 2.0, 66.3, 2.0, 55.6, 2.0, 54.4, 2.0, 44.5,
Execution Time: 0 milisecond

Execution Time: 0 milisecond

