Circles in a square

Jan Boucek

March 31, 2018

1 Problem statement

The general problem is called a circles packing. The goal is to pack circles in some shape, that the circles are not overlapping.

$$\rho = \frac{1}{6}\pi\sqrt{3} \doteq 0.907 \tag{1.1}$$

This gives us a theoretical upper bound of

$$r = \sqrt{\frac{\sqrt{3}}{6N}} \doteq \frac{0.53}{\sqrt{N}} \tag{1.2}$$

The solution will approach this formula when N approaches to infinity.

2 Representation

Each circle has 2 degrees of freedom considering position and 1 degree of freedom given it's radius.

Radius of all the circles is the same and can be computed from knowing the positions of all circles. Computing the distances between each two circle centers, the radius is either half of the minimum distance, or a minimum of the distance from circle centers to a square side.

That takes us to the representation of a solution of N circles as a vector of $2N$ real numbers, further referred as dimensions:

$$s = [x_1, y_1, x_2, y_2, ..., x_n, y_n] \tag{2.1}$$

However working with real numbers is not feasible for computer and it must be quantified. For evolutionary algorithms the solution should not be too long and easily represented as a binary array. That led me to a quantization of the square size 256 working with integers. That way each number can be represented by one byte as unsigned integer. That results in the solution being a binary array of length $8 * 2 * N$, which is good enough representation, especially for small N and simple to work with.

3 fitness function

The fitness function given by the problem is the radius R of circles. That works fine, but to help faster convergence and avoid long times spent in saddle points, it is encouraged for the circles to be far from each other. That is provided by adding a sum of distances among circles multiplied by a small constant

$$U(s) = R + 0.001 \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \sqrt{(c_{ix} - c_{jx})^2 + (c_{iy} - c_{jy})^2} \tag{3.1}$$

4 Local search

The local search with Best-improving strategy has been chosen and implemented. The basic pseudo-code is

```
x := random()
for i:=0 to epoch_num{
        y := bestOfNeighborhood(N(x))
        if fitness(y) >= fitness(x){
                x := y
        }
}
```

For generating neighborhood I have tried all simple gradients in both directions, that means to try to add and try to subtract one in each dimension. That did not work very well,

since from the settle point it is sometimes possible to improve only with a combination of addition an subtraction in more dimensions. That led me to generate the neighborhood by random steps in each dimension by -1, 0, or 1.

5 Evolutionary algorithm

The basic schema of implemented evolutionary algorithm is showed in the figure fig**??**.

The initial population is initialized randomly and the algorithm runs for a certain number of steps. The pseudo-code is:

```
x := random_population()
for i:=0 to epoch_num{
        parents := selection(population)
        pairs := select_pairs(parents)
        children := crossover(pairs)
        best_s = argmax fitness(s)
        population := mutate(children, 0.01)
}
```

5.1 Selection

Selection is an important part of evolutionary algorithms. The goal is to select some solutions, which have good enough fitness function, but still have some diversity. In my case, the selected population is half of the original one.

The selection has been implemented according to the Stochastic Universal Sampling(SUS). That gives higher probability of good solutions be chosen, but still some bad solutions will be chosen also with smaller probability. This allows one solution be chosen more times, but that does not matter. My implementation also ensures, that the best solution will be chosen. This approach also gives upper bound and lower bounds on the number of how many times will the solution be chosen.

5.2 Crossover

The selection of parents is done randomly. Two parent from a current population use their genes for crossover mutation. Their genes are combined together with 4 randomly selected cross sections. Since the solution is represented as a binary sequence, the combination is simple, but combination of two binary sections does not have to make sense. That way a fill population of children is created. A crossover mutation with 3 sections is shown in the figure **??**

5.3 Mutation

This is the simplest operator of all. Each bit is negated with a probability of 1%. This sometimes helps to get from a local optima and explore other possibilities. An example of mutation

is shown in the figure **??**

6 Memetic algorithm

Memetic algorithm is a combination of evolutionary algorithm and local search. The 1st generation memetic algorithm has been implemented:

```
population := random_population()
for i:=0 to epoch_num{
        population := ea_update(population)
        for each s in population{
                if prob(0.5){
                iterations = random(1, 5)
                        s := ls_update(s, iterations)
                }
        }
}
```

The population is first updated by one cycle of the implemented evolutionary algorithm. Then each solution is updated by the linear search k times with the probability of 0.5, where k is sampled from uniform integer distribution from 1 to 5. This aproach combines advantages of both above algorithms. It can overcome a local optima and it also improves in a simpler situations.

7 Comparison of the algorithms

The algorithms have a different advantages and disadvantages. The local search is very simple to implement and it converges to a local optima fast. The evolutionary algorithm on the other hand can find very good solutions, that can not be easily found by gradient descent, especially with higher number of circles. The memetic algorithm combines these features and performs better, than the other algorithms.

The algorithms have been run 10 times on a certain number of circles and the medians have been compared. The results have been shown in the figures **??**, **??**, **??**, **??**, **??**, **??**. The results of each algorithms have been evaluated and shown in the table **??**. The memetic algorithm shows the best results in all situations.