Evolution Strategies (ES)

Assignment 5 CS472 F20 DUE: Fri Nov 13, 2020 at 5pm PT

REWARD: 200 points

1 The problem

Let's play with solving a real number vector problem. The problem is to place k points in a unit circle so that the the minimum distance between any two points is maximized.

$$\max_{P} \left(\min_{p_i, p_j \in P} \text{dist}(p_i, p_j) \right)$$

where P is a set of points in the circle. This is a relatively hard problem. Let's try to solve it with evolution strategies (ES). Some example solutions are seen in Figure 1. Use whatever ES formula you would like. Be sure to explain which you used. For example: $ES(4 + 40)$ or $ES(8, 20)$ or ....

Your program will be called points and take the number of points as a single argument on the command line. Your representation will be a list of points in polar coordinates. The first point in the list has a fixed θ of 0 and radius of 1!

First try the direct fitness of using the square of the minimum distance as your fitness function. Here is the formula for the distance between two points (θ_1, r_1) and (θ_2, r_2):

$$\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\theta_1 - \theta_2)}$$

The square of the distance means you don't have to take the square root in the fitness computation during evolution, saving time. Note: if you are only relying on whether $fitness(x) > fitness(y)$ then you can always do $fitness(g(x)) > fitness(g(y))$ provided g is a monotonically increasing function as in this case for speed.

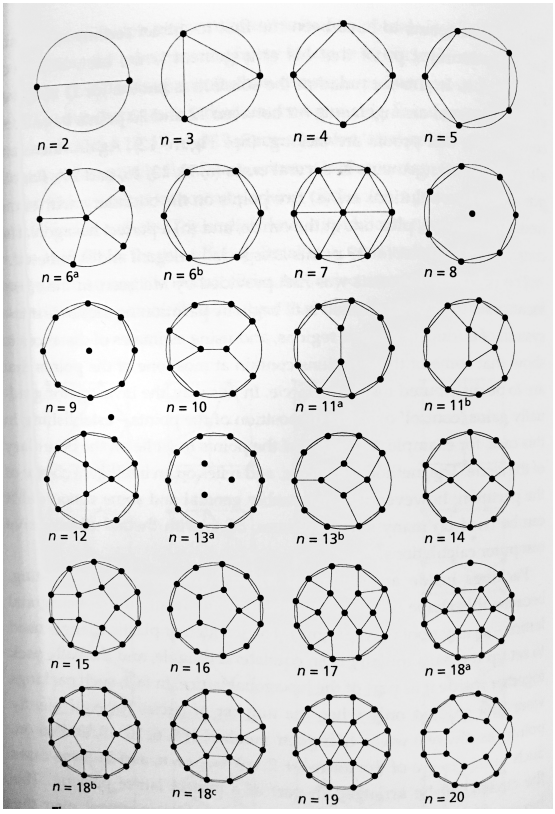

Figure 1: Dots on a circle to maximize minimum distance between points. (From *How to Cut a Cake: And Other Mathematical Conundrums* by Ian Stewart)

Using the minimum distance means that most points don't directly contribute to the fitness since they aren't involved the minimum edge, except indirectly. This makes for lots of neutral places in the landscape. Furthermore, points can be moved around arbitrarily as long as they don't effect the minimum. Another approach would be to try to solve a related problem which is to minimize the sum of the forces between the points and let the force be proportional to $1/d^2$? Does that give you the same answer? Is it any faster? Just something to think about.

The output (see sample below) of your program should given in the following in order:

1. On the polar coordinates of the points on separate lines presented as r with $0 \leq r \leq 1$ and θ with $0 \leq \theta \leq 2\pi \approx 6.2831853$. The first point in the list is r=1, $\theta = 0$. That can be assumed to be a fixed point. You can put it in your representation or special case it, as you like.
2. The maximum minimum distance you found. See format below.

3. A list of ALL $\binom{n}{2}$ distances between points sorted from longest to shortest. This is the "signature" for your solution.

Here is the example of the output format (for 4 points) you should use.

```
  1.000000    0.000000
  1.000000    1.257914
  1.000000    2.512080
  1.000000    3.769721
  1.000000    5.028705
Fitness:   1.173570
**   0    1.90
**   1    1.90
**   2    1.90
**   3    1.90
**   4    1.90
**   5    1.18
**   6    1.18
**   7    1.18
**   8    1.17
**   9    1.17
```

For this problem use polar coordinates in your representation. Use a simple ES($\mu + \lambda$). You may use sigmas or not and in any way you choose. Half the problem is getting it up and evolving and the other half of your time will be in getting it to work efficiently. The program will be called `points` and take one argument: the number of points.

2 Implementation Hints

One way to do ES is to create a single array of size $\mu + \lambda$. Then use the first μ elements as the parents and mutate them to make the last λ elements. Then you sort the whole array by fitness so the best μ elements occur first. (you can be more efficient by just putting the best μ elements first. But whether you sort or use a select-μ-best you need to do the selection in an unbiased way. Canned sorts often don't make that an option. If you want to make sure your sort routine is unbiased it needs to be sure to choose arbitrarily between equal elements. Here is a compare routine in qsort(3) form that will do that. In the case below we make a random choice of order if the order is equal so that qsort does not get to decide, in a **biased** way, the order. IMPORTANT: This example compare function assumes the elements of the population are pointers to Genes. Read your sort routine documentation.

```
// compare fitness in decreasing order
int fitnessCompare(const void *a, const void *b)
{
    double fitnessa = (*((Gene **)a))->fitness;  // cast to Gene *
    double fitnessb = (*((Gene **)b))->fitness;  // cast to Gene *

    if ( fitnessa > fitnessb) return -1;
    if ( fitnessa < fitnessb) return 1;
    return (choose(.5) ? 1 : -1);         // choose a random order if equal
}
```

3 Some Possible Results

I have a script that takes the numbers in the above format. Here are statistics over all the distances in configurations I found for various numbers of points. The minimum distance (column 4) in the table is what is being maximized. The answers that generated these stats might not be right but they are what my algorithm got. It used a simple ES(μ+λ). This is a table averaged over 30 trials for each number of points. $\mu = 32, \lambda = 128$ and 20K generations.

NumPts	Mean	Stddev	Max
2	2.0	0.0	2.0
3	1.732	0.0	1.732
4	1.413	0.001	1.414
5	1.149	0.059	1.175
6	1.0	0.0	1.0
7	0.995	0.003	1.0
8	0.856	0.008	0.866
9	0.742	0.014	0.762
10	0.688	0.018	0.706
11	0.656	0.022	0.677
12	0.617	0.038	0.653
13	0.586	0.027	0.613
14	0.554	0.029	0.594
15	0.523	0.017	0.556
16	0.512	0.009	0.529
17	0.501	0.009	0.516
18	0.488	0.022	0.506
19	0.465	0.018	0.493

4 Submission

Homework will be submitted as an uncompressed tar file to the homework submission page linked from the main class page. You can submit as many times as you like. The LAST file you submit BEFORE the deadline will be the one graded. For all submissions you will receive email giving you some automated feedback on the unpacking and compiling and running of code and possibly some other things that can be autotested.

Have fun.