<http://goldberg.berkeley.edu/pubs/fogel-CASE07_0166_FI.pdf>

## One-dimension beacon navigation problem

Simplifications:

1. In one dimension case we can reduce the problem by using linear segments:



1. We could discard dealing with partition size and concentrate only on the total number of distinctly identifiable partitions (DIPs) because by adjusting the beacon strength (segment sizes) we can get DIPs of equal size.

Theorem:

Given *n* beacons in one dimension the best solution will have *2n* distinctly identifiable partitions (DIP).

Proof:

We will prove this by induction on the number of beacons.

1. Basis: given one beacon it is obvious that the maximal number of DIPs is 2.
2. Let’s assume that for given *n* beacons we have *2n* DIPs and now we’ll check for *n+1* beacons:  
   * Using the *n* beacons we get the *2n* DIP – by the assumption.

b

a

c

* + Let’s select 2 adjacent DIPs (*A* and *B*):
  + Let’s place the new (n+1) segment *C* such that it will intersect both *A* and *B* DIPs starting from the middle of *A* and ending at the middle of *B*.
  + The first part of *A* (not intersecting with *C*) is described by the same beacons as before; the same applies to the last part of B.
  + The part of A which intersects with C is described by the **new** set of beacons (the “old” *A* set + *C* beacon) which gives us 1 new DIP.
  + The same is valid for *B* from the symmetry reasons, which gives us another **new** DIP.
  + Now we used *n+1* beacons and get *2n+2* DIPs.

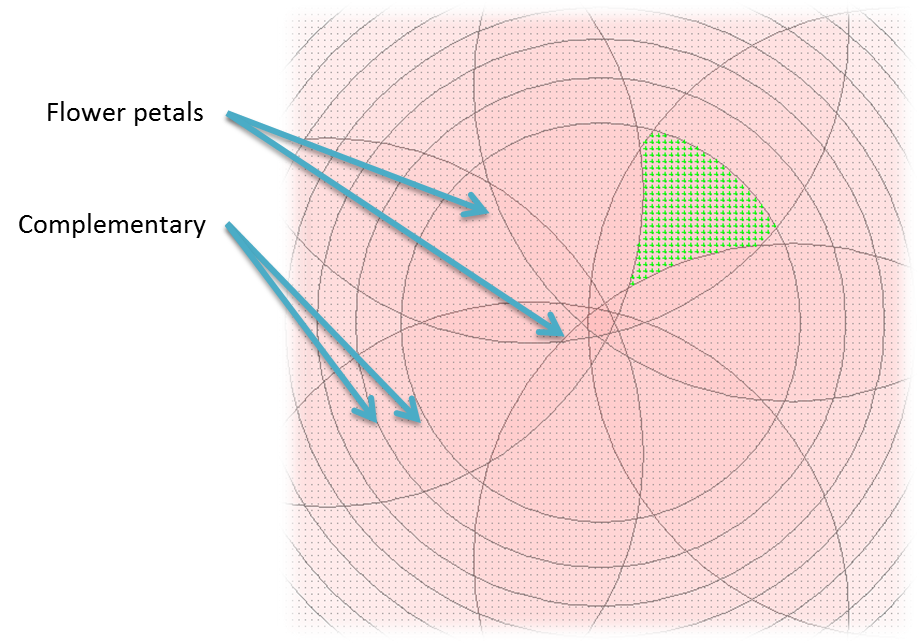
Our solution for one-dimension:



## Two-dimension beacon navigation problem

Proposed Heuristics:

After examining different approaches (see below) we came up with an algorithm called “Intersected Flower”. This algorithm draws a flower-like figure using circles (beacons) in the middle of the solution area and then with additional complementary circles intersects the flower petals:

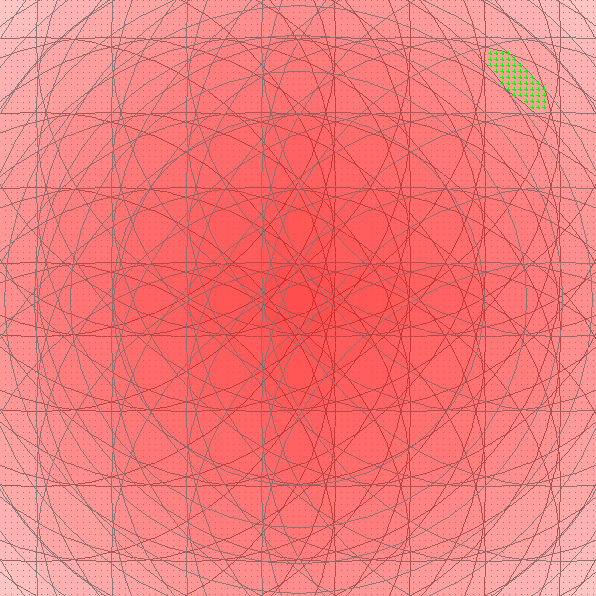
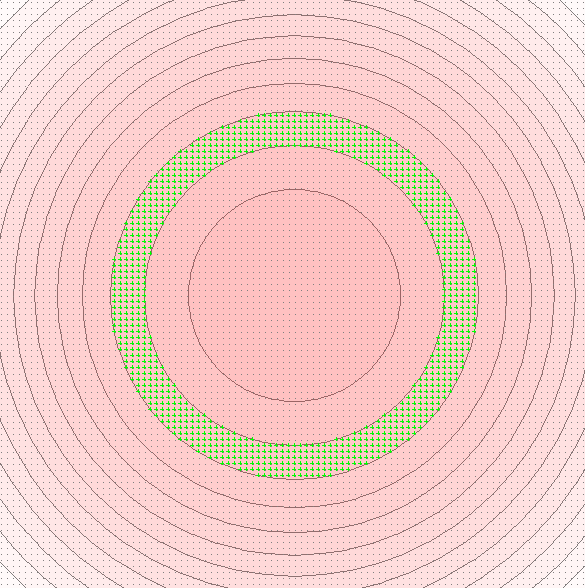
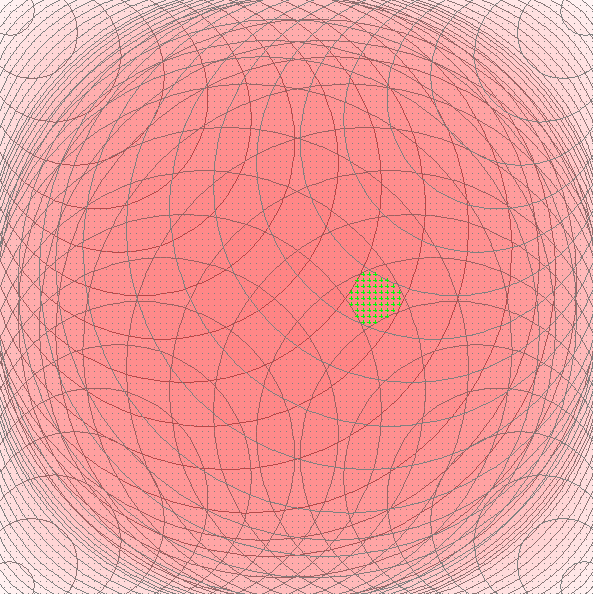
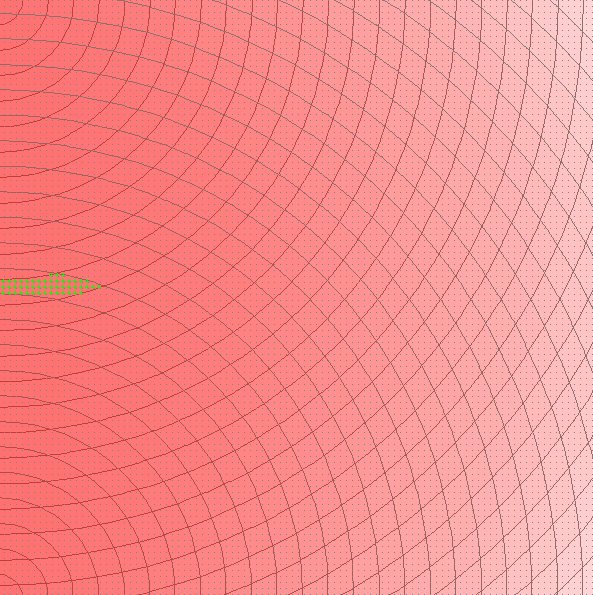


The parameters for our algorithm are:

1. flower petals circles center as a distance from the plot center
2. flower petals radius
3. number of complementary circles
4. complementary circles radius

Our implementation doesn’t make any smart choice for the parameters values. Some possible improvements we were thinking about are based on finding optimal values (local minima) using conjugate gradient descent (should be feasible for the small inputs we’re dealing with).

Alternative Approaches (discontinued):

1. Grid based approach – making a grid-like placement ( beacons for each axe):  
   
2. Centered growing circles (based on the fact that energy function is not about the distance between the sampling points):  
   
3. Growing corner circles:  
   
4. Double-spectrum approach:  
   

Benchmark table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File name** | **N=4** | | **N=8** | | **N=16** | | **N=32** | | **N=64** | |
| **D** | **T** | **D** | **T** | **D** | **T** | **D** | **T** | **D** | **T** |
| bm\_grid10000\_.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_grid1000\_.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_grid100\_.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_10000\_0.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_10000\_1.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_10000\_2.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_10000\_3.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_1000\_0.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_1000\_1.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_1000\_2.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_1000\_3.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_100\_0.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_100\_1.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_100\_2.txt |  |  |  |  |  |  |  |  |  |  |
| bm\_random\_100\_3.txt |  |  |  |  |  |  |  |  |  |  |

Implementation details:

We implemented our solution using Java programming languages (language level complies with version 1.6). We do use JRE classes in java.awt package for the points (Point2D class) and beacons (Ellipse2D class). We also developed a GUI which allows us to visually represent our ideas and play with these (see usage details). Our solution (as well the alternatives we chose to discontinue) are located under idc.edu.ex2.solution package. Additional libraries used:

* Google-guava (<http://code.google.com/p/guava-libraries/>)
* Apache Commons Lang (<http://commons.apache.org/lang/>)

Submission & usage:

Our submission zip archive contains:

* ex2.docx (this file)