1. Fitness Deterioration

This chapter contains detailed description of the deterioration algorithm which is the central point of the *Sequential niching* algorithm described in chapter 2. Before diving into the details, here is the formal definition of the fitness deterioration:

Definition. Fitness Deterioration is a process of degrading the fitness function in areas occupied by groups of individuals obtained from clustering. This goal is achieved by crating a linear combination of the current fitness function and *crunching functions* which approximate the fitness in subsets of problem domain occupied by clusters.

Let F_k be the fitness function in k-th iteration and $C_1, \ldots, C_M$ be M clusters found in k-th iteration of our sequential niching algorithm described in chapter 2. For each cluster C_i we create the *crunching function g_i* and then we construct deteriorated fitness F_{k+1} (which will be used in the $(k+1)$-th iteration) as follows:

$$F_{k+1} = F_k + \sum_{i=1}^{M} \alpha_i g_i \tag{1.1}$$

the selection of coefficients $\alpha_1, \ldots, \alpha_M$ depends on the type of deterioration used and will be described later in this chapter.

Definition. Crunching function. For a given cluster of individuals crunching function is a real-valued non-negative function K which is created by estimating the density function of individuals in the cluster and then by adapting it in order to approximate the basin of attraction occupied by the cluster.

Section 4.1.1 describes the family of functions which are suitable to be used as crunching functions in deterioration process.

Based on the assumption that clusters lie inside basins of attraction and that the distribution of individuals inside the cluster is a good approximation of the shape of the basin occupied by the cluster, the deterioration algorithm tries to exploit information provided by the clustering algorithm and based on that information it augments the fitness function in order to minimize the probability of finding already explored basins of attraction in further iterations.

We may ask ourselves why we do not prevent exploration of basins we found in previous iterations simply by remembering the regions occupied by clusters and ignoring individuals which fall in this regions. The answer is probably the most important reason why we have chosen fitness deterioration for this task. We can not prevent individuals to explore neighborhood of the solutions found in previous iterations because such approach would cause our metaheuristic to *completeness* in terms of search operations.

Definition. A set Op of search operations searchOp is *complete* if and only if every point g_1 in the search space G can be reached from every other point $g_2 \in \mathbb{G}$ by applying only operations $searchOp \in Op$.

$$\forall g_1, g_2 \in \mathbb{G} \Rightarrow \exists k \in \mathbb{N} : P(g_1 = Op^k(g_2)) > 0 \tag{1.2}$$

If the set of search operations is not complete, there are points in the search space which cannot be reached. Then, we are probably not able to explore the problem space adequately and possibly will not find satisfyingly good solution. That is why it is better to use fitness deterioration as a way to discourage individuals from sinking to the same basins of attraction twice.

Here I would like to emphasize the fact that the deterioration process does not try to accurately interpolate the fitness function in the neighborhood of the solution, because it would be very expensive in a high dimensional spaces. Instead it tries to find a simple crunching functions which would degrade the fitness landscape in the areas occupied by the clusters.

We now can define what are the characteristics of a good deterioration algorithm:

– **It has to be chep** in terms of space and time. We are looking for methods which can be successfully applied for solving high dimensional functions' optimization. After every iteration the resulting fitness function is the linear combination of crunching functions, therefor we must ensure that computation of a single crunching function is $O(1)$ and does not depend no the number of individuals inside a cluster or the number of clusters. Resulting fitness must be $O(k)$ where k is the number of solutions (clusters) found in previous iterations.

– **It has to discourage the population of EA from visiting the neighborhood of solutions found in previous iterations twice**. We do not want these regions to be inaccessible as this would break the **completeness** of our algorithm, we just want an individual to be given a low fitness when it finds itself in these regions.

– **It has to be easy to extend for subsequent solutions**. The way we construct fitness function in subsequent iterations already fulfills this requirement. To extend the deterioration for subsequent solutions we simply add new crunching functions multiplied by the scaling factors we define later in this chapter.

Definition. Premature Convergence [**?**] An optimization process has *prematurely converged* to a local optimum if it is no longer able to explore other parts of the search space than the area currently being examined and there exists another region that contains a superior solution.

The premature convergence is not a problem in our algorithm, because once the population finds itself in the local optima trap it is very likely that the population will be avoiding this region in the next iterations. Actually we encourage an optimization process performed by the used EA (e.g. SGA) to converge quickly to local solution in order to find new solutions faster in the next iterations.

An ideal EA to be used in our hybrid algorithm defined on page 10 would be the one which produces many small populations in problem space and each population would use strong selection pressure with small mutation rate with high recombination rate to increase exploitation. This is why Hierarchical

Genetical Search (HGS) algorithm would be perfect for our needs as it possess all of the listed characteristics. However we may also choose some simple SGA or Evolution Strategy with small population size and strong exploitation characteristics.

1.1. Fitness deterioration and clustering

To increase the accuracy of the fitness deterioration process we want to use the maximum amount of information provided by the clustering algorithm. As we mentioned earlier (see the definition of *Fitness deterioration*) we crate one crunching function per cluster which, depending on the shape of the cluster may not be very accurate. To increase that accuracy we use the following property of the *OPTICS ordering*:

While creating the *ordering* OPTICS constructs density-based clusters with respect to different densities simultaneously. *OPTICS ordering* actually contains the information about the intrinsic clustering structure of the input data set (up to the generating distance ϵ) [**?**].

TODO: plot

Having the *OPTICS ordering* of the population returned by the EA our method iteratively extract clusters with higher densities by decreasing the *neighborhood radius (ϵ)*, construct crunching functions for extracted clusters and check if the resulting crunching functions is more accurate than the best found in previous iterations.

```
1: $\epsilon' = \epsilon$
2: while $\epsilon' > treshold$ do
3:     $clusters = optics.extractDBSCANClustering(\epsilon')$
4:     $crunchingFunctions = crateCrunchingFunctions(clusters)$
5:     $distance = getDistance(crunchingFunctions, currentFitness, population)$
6:     if $distance < minDistance$ then
7:         $saveBestCrunching(distance, crunchingFunctions)$
8:     end if
9:     $\epsilon' = \epsilon' * 0.75$
10: end while
```

1.1.1. Crunching functions

When degenerating a single basin of attraction represented by a cluster of individuals, we are looking for *crunching functions* with the following properties:

1. cheap (in multi-dimensional spaces). Finding the optimal solution to complex high dimensional, multimodal problems often requires very expensive fitness function evaluations. It is crucial we increase the cost of fitness evaluation very slightly by introducing the *crunching functions*
2. has low impact on the areas of fitness landscape which are distant from the cluster so that we do not introduce unnecesary noise to the fitness landscape in further iterations

A class of functions which which are suitable for the fitness deterioration are so called *(*kernel functions) [**?**]. Examples of kernels are:

– Triangular $K(u) = (1 - |u|)\,\mathbf{1}_{\{|u| \leq 1\}}$

– Epanechnikov $K(u) = \frac{3}{4}(1 - u^2)\,\mathbf{1}_{\{|u| \leq 1\}}$

– Quartic $K(u) = \frac{15}{16}(1 - u^2)^2\,\mathbf{1}_{\{|u| \leq 1\}}$

– Gaussian $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$

From the above list we choose Gaussian kernel which meets all requirements and has one great advantage over the other kernels which is the fact that Gaussain function may be defined on the whole domain, whereas the rest kernels are defined on some finite intervals, which makes them computationally inefficient in high-dimensional spaces).

1.2. Basic Scheme

The basic version of our deterioration algorithm is as follows:
For each cluster generate a multi-dimensional Gaussian function:

$$g(x) = -F_k(x_{max}) exp(\frac{-1}{2}(x - \mu)'\Sigma^{-1}(x - \mu)) \tag{1.3}$$

where F_k is a fitness function in kth iteration of the algorithm, Σ is an **unbiased sample covariance matrix** [**?**] estimated from the cluster population:

$$\Sigma = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T \tag{1.4}$$

Fitness function in $k + 1$th iteration is of the form:

$$F_{k+1} = F_k + \sum_{i=1}^{M} g_i \tag{1.5}$$

where M is the number of generated Gaussian functions. In this case all α-coefficients from equation 4.1 are equal 1.

This version of the algorithm cause strong degradation of the fitness function in the neighborhood of the solution, so there is a risk that deterioration of one solution affect others which are close to it. To overcome this issue we developed so called *weighted scheme* described in the next section. Disadvantages of the weighted approach is that it may take a long time to find new solutions, because of the slow degradation process.

1.3. Weighted Scheme

This type of fitness deterioration is more accurate and may produce more stable fitness (*Basic scheme* may produce some sharp peaks in fitness landscape, because of very aggressive fitness degeneration) than

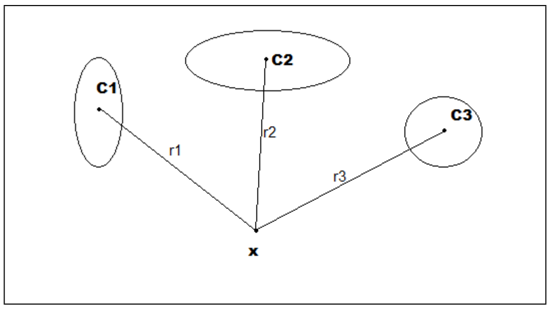

Figure 1.1: The coefficient α_i is inversely proportional to the distance from the centers of clusters (C_1, C_2, C_3). α_i may be seen as the impact C_i has on x

the *Basic scheme*. Is also more complex as it is resulting in slightly more computionally intensive fitness functions.

Initial steps are the same as in *Basic scheme* (we create multi-dimensional Gaussian function for each cluster). What is different is how we compute the deteriorated fitness for a given individual x:

$$F_{k+1}(x) = F_k(x) + \sum_{i=1}^{M} \alpha_i(x) g_i(x) \tag{1.6}$$

where the α-coefficients are given by the following equations:

$$\alpha_1 + \ldots + \alpha_M = 1 \tag{1.7}$$

$$\alpha_i = \xi \frac{1}{r_i} \tag{1.8}$$

which gives:

$$\frac{1}{\xi} = \frac{1}{r_1} + \ldots + \frac{1}{r_M} \tag{1.9}$$

where r_i is the distance from the center of cluster i. This coefficients are computed separately for each new individual and this is why this method is more costly than the previous one.

The *Weighted Scheme* is more smooth and gentle and more costly. It should do better than *Basic scheme* for problems with many local solutions.

1.4. Covariance Matrix Adjustment

Because of the fast convergence of populations generated by the EA algorithm to the local solutions and the algorithm we use to increase accuracy of the deterioration process (see section 4.1), clusters sometimes becomes very dense in areas of local optima, therefore Gaussians created for such clusters does not approximate a basin of attraction well, speaking informally: Gaussian functions created for such clusters consist of high and thin peaks which deteriorate only the area inside the cluster, not the basin of attraction in which the cluster resides (see Figure 4.3 and 4.4). To overcome this issue we developed so called *Covariance Matrix Adjustment (CMA)* algorithm described below.

We use sample covariance matrix as an estimator [**?**], which is extremely sensitive to outliers. However we may take this property as our advantage and incorporate it CMA algorithm. Having given a cluster of points the CMA algorithms works as follows:

- we create covariance matrix using *sample covariance matrix* [**?**] and perform eigenvalue decomposition, which gives use the d orthogonal directions (eigenvectors), which define the orientation of the Gaussian 'bell' (d is the dimension of the problem space)
- for each eigenvector v_i we generate two points (so called *outliers*)

$$p_{i1} = \overline{m} + \sqrt{\lambda_i}\overline{v_i}, p_{i2} = \overline{m} - \sqrt{\lambda_i}\overline{v_i} \tag{1.10}$$

  where $\overline{m}$ is the center of the cluster (mean value), λ_i is an eigenvalue of the eigenvector v_i
- then we add these $2d$ generated *outliers* to the initial population which constitute a cluster, and compute new covariance matrix.
- because *sample convariance matrix* is very sensitive to outliers the resulting covariance matrix produces a Gaussian function whose 'bell curve' is more stretched in directions of eigenvectors.

1.4.1. Results

The figures below shows the result of our sequential niching algorithm for two simple functions from $f : \mathbb{R}^2 \to \mathbb{R}$, specifically:

- $f(X) = 2e^{-(x^2+y^2)}$, where $X \in \mathbb{R}^2$
- $f(X) = e^{-(x^2+y^2)} + 1.4e^{-((x-1.7)^2+(y-1.7)^2)}$, where $X \in \mathbb{R}^2$

Figures 4.3 and 4.4 shows how much the algorithm benefit from using the CMA algorithm. The plots show deterioration results applied to functions from 4.1 and 4.2 without the CMA algorithm.

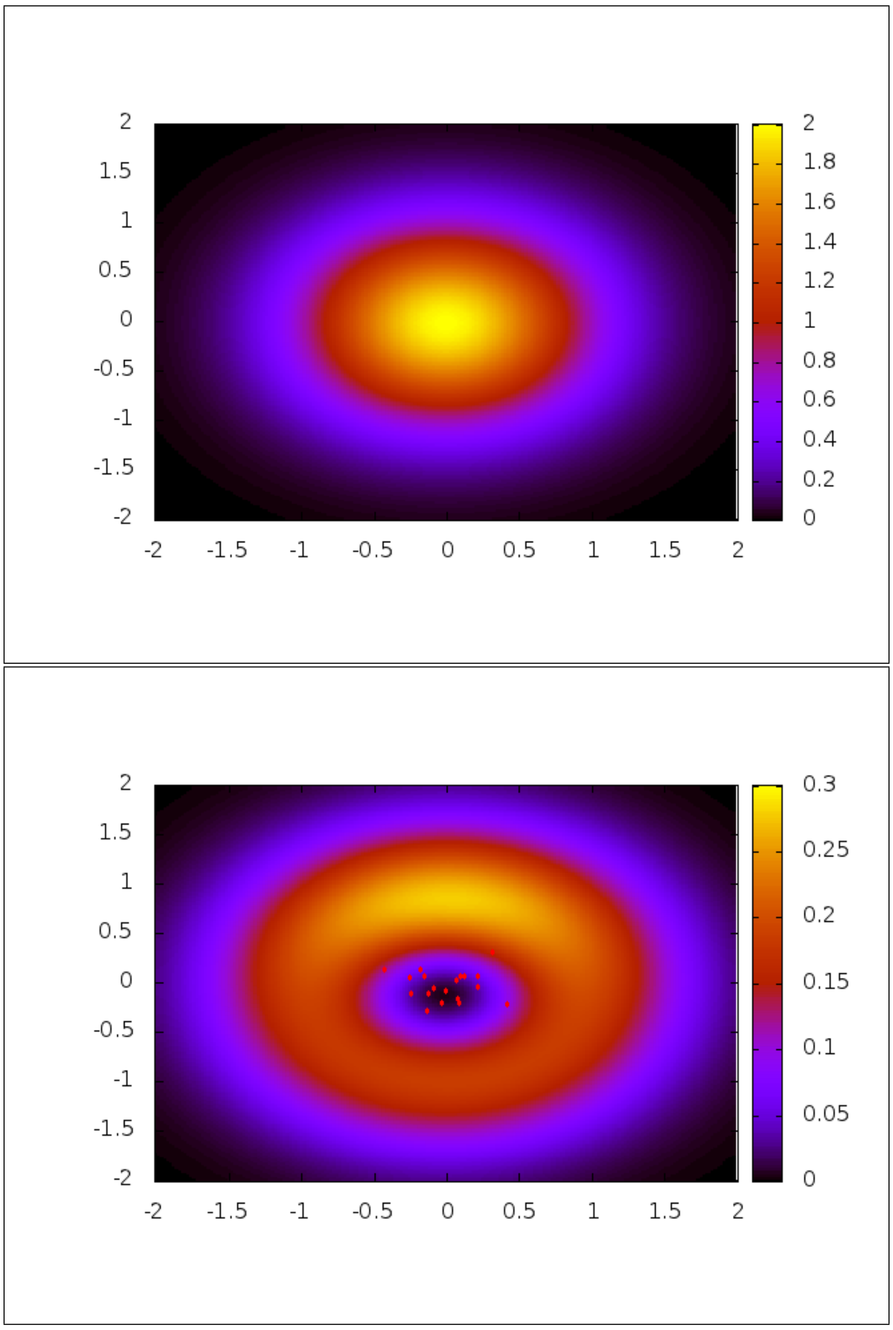

Figure 1.2: The result of basic deterioration scheme with CMA applied to unimodal function: $f(X) = 2e^{-(x^2+y^2)}$. Optics paramters: $minPts = 20, \epsilon = 0.4$, algorithm: SGA, iterationCount=1

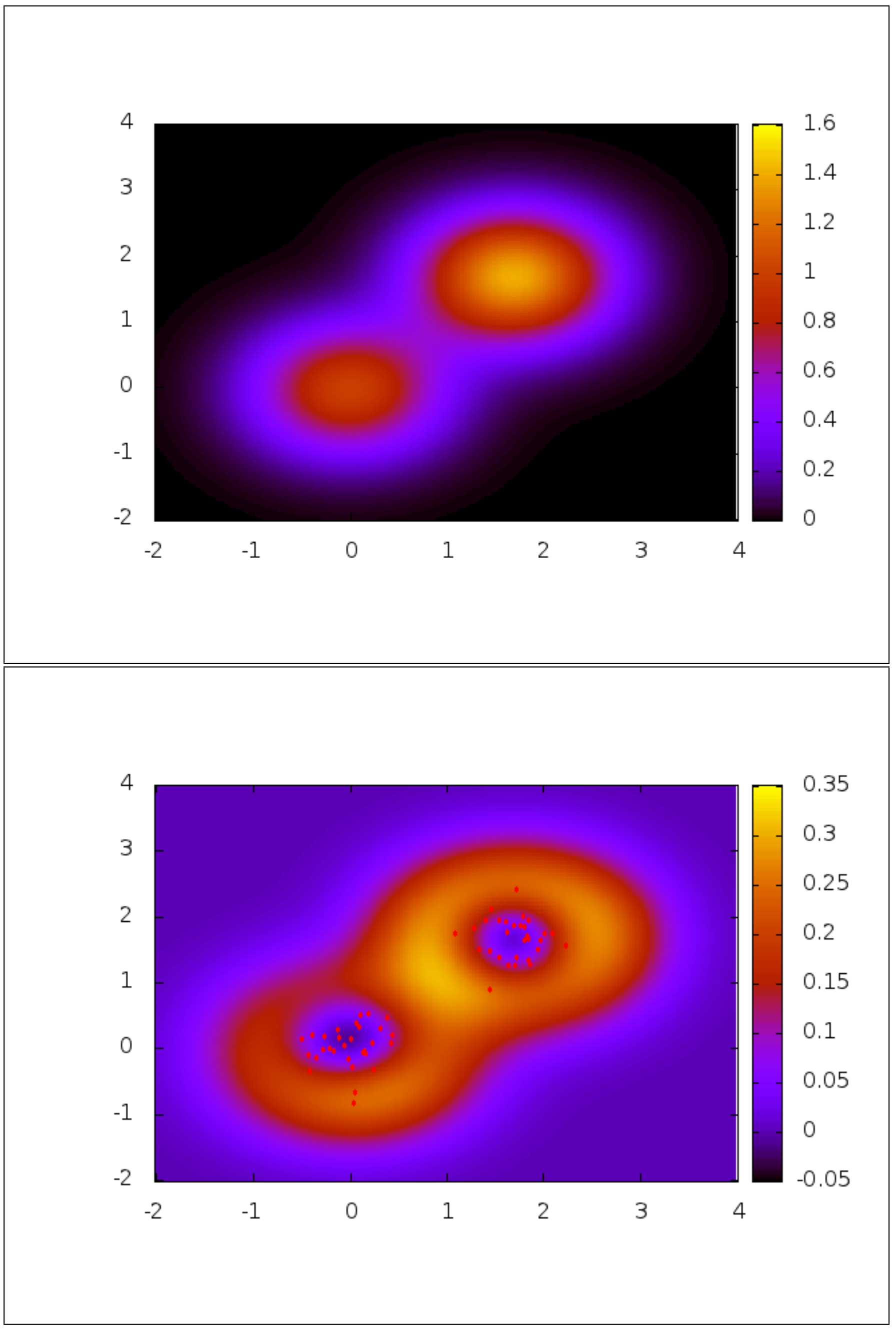

Figure 1.3: The result of basic deterioration scheme with CMA applied to bimodal function: $f(X) = e^{-(x^2+y^2)} + 1.4e^{-((x-1.7)^2+(y-1.7)^2)}$. Optics paramters: $minPts = 20, \epsilon = 0.4$, algorithm: SGA, iterationCount=2

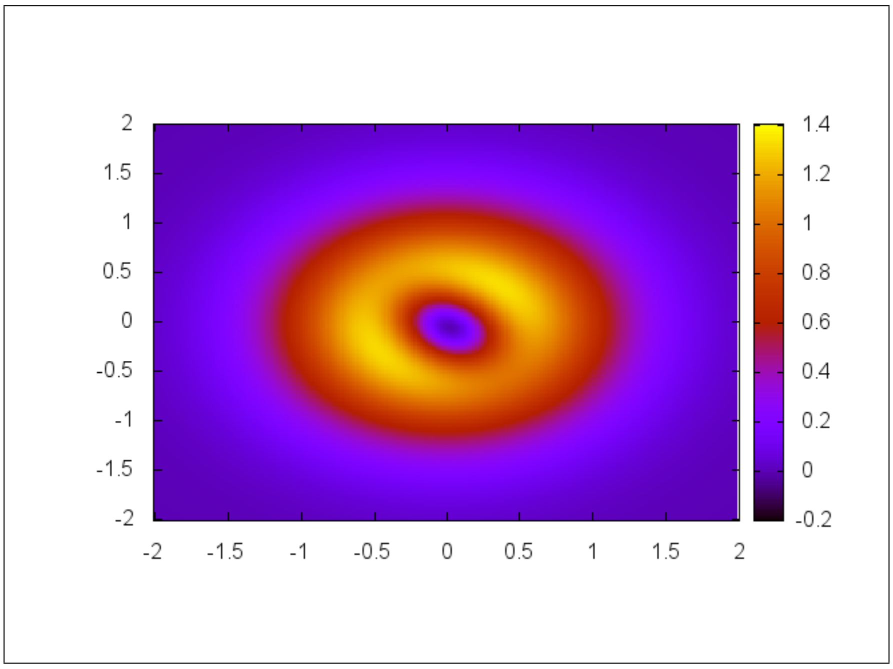

Figure 1.4: The result of basic deterioration scheme without CMA applied to unimodal function from 4.1. Optics paramters: $minPts = 20, \epsilon = 0.4$, algorithm: SGA, iterationCount=2. We may see that the overall landscape decreases only by 30 percent, while using CMA gives us 85 percent of decline.

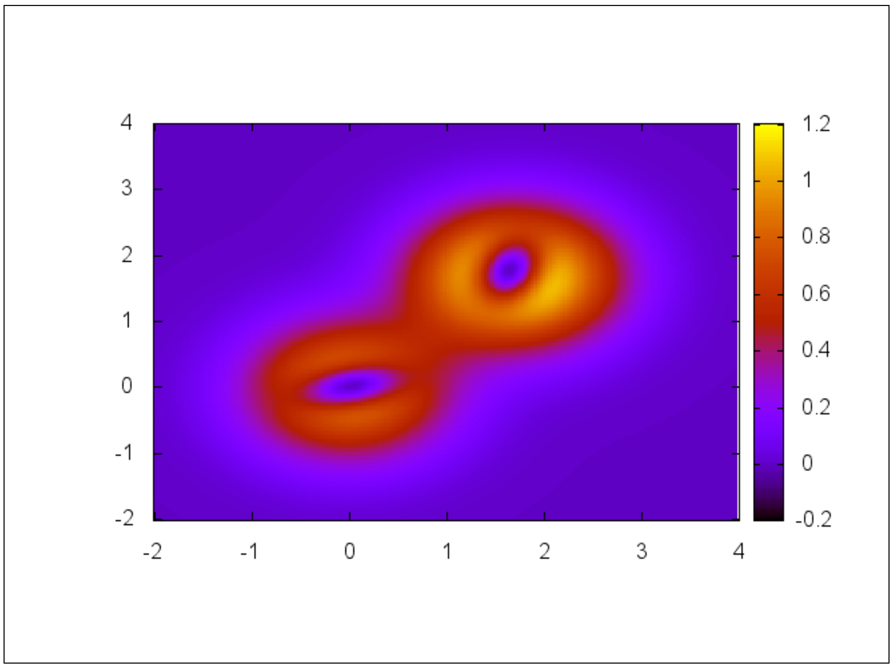

Figure 1.5: The result of basic deterioration scheme without CMA applied to bimodal function from 4.2. Optics paramters: $minPts = 20, \epsilon = 0.4$, algorithm: SGA, iterationCount=2. We see 25 percent of deterioration, while CMA gives us 78 percent when applied to the same case.