**AGH**

**University of Science and Technology in Krakow**

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

DEPARTMENT OF COMPUTER SCIENCE


MASTER OF SCIENCE THESIS

ADRIAN WOLNY

# IMPROVING POPULATION-BASED ALGORITHMS USED IN GLOBAL OPTIMIZATION WITH FITNESS DETERIORATION TECHNIQUES

SUPERVISOR:
Robert Schaefer, Prof., PhD, DSc

Krakow 2011

...

# Contents

# 1. Introduction

> It is impossible for any optimization algorithm to outperform random walks on all possible problems.
>
> ... a conclusion from No Free Lunch Theorem

## 1.1. A statement of a problem

A Global Optimization Algorithm is defined as optimization algorithm that employs measures that prevent convergence to local optima and increase the probability of finding a global optimum.

Evolutionary algorithms are known as a generic population-based metaheuristics which often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape; this generality is shown to be a great successes in many real-life problems. Evolutionary algorithms have the tendency to lose diversity within their population of feasible solutions and to converge into a single solution. However, there are domains where the global solution may not suffice. Such problems require the location and maintenance of multiple robust local solutions, i.e. local solutions whose basins of attraction are properly wide and deep.

The most common technique in evolutionary algorithm which is used to achieve this goal is to incorporate some sort of niching method like crowding or fitness sharing which promote diversity of population, which in turn delay premature convergence and likely enable the algorithm to find multiple optimal solutions in single population.

Standard niching methods are often ineffective and hard to introduce in existing evolutionary algorithms. In this paper we adopt a different approach to multimodal function optimization. Instead of embedding a niching method in the evolutionary algorithm itself we use a hybrid approach in which we perform several runs of a evolutionary algorithm and alter the fitness function in every subsequent run in a way that prevents exploration of basins of attraction which were found in previous runs of the algorithm.

In each iteration we run EA, then cluster received population and based on the assumption that clusters of individuals obtained from the clustering algorithm are located in basins of attraction we interpolate each basin by multidimensional Gaussian function. By combining these functions with current objective function in a proper way we create deteriorated fitness function which will discourages future runs from revisiting the same area.

This work tries to find an effective fitness deterioration technique in high-dimensional domain spaces. We have implemented a general-purpose framework which can be used to test our fitness deterioration techniques in conjunction with various evolutionary algorithms. While our algorithm may be used with many types of EAs it would be the most efficient when used with algorithms which are capable of finding many local solutions in single run. This is why for tests we choose so called Hierarchical Genetic Strategy which performs efficient concurrent search in the optimization landscape by many small populations.

The quality of the deterioration process strongly depends on clustering results. We choose density-based algorithm called OPTICS as with this method we can extract clusters of different densities very efficiently and choose clusters which give the best accuracy of fitness deterioration process.

## 1.2. Related Work

As mentioned before this work focus on finding solutions for multi-modal optimization tasks. There are many publications which describes how to extend EAs to multi-modal optimization. Most of them focuses on *Niching methods* [8, 9, 10] which address this issue by maintaining a population of diverse solutions throughout the time and this way they allow parallel convergence into multiple good solutions in multimodal domains.

Our solution works by running an evolutionary algorithm (here we may choose SGA, Evolution Strategy, etc.) multiple times and after each run it tries to detect basins of attraction where local extremum reside and degenerates the fitness landscape. This process is what we call *Sequential niching* algorithm. The rest of this paper describes the *Sequential niching* algorithm in a top-down manner, starting from the high-level description of the algorithm in chapter 2.

At this point it is worth mentioning some of the works of Prof. A. Obuchowicz especially the publication [3] which is the only one I found which use the term *fitness deterioration* explicitly. In [3] he describes ESSS-DOF algorithm (Evolutionary Search with Soft Selection with Deterioration of Objective Function) as an extension to the ESSS method which maintaining population diversity by the following schema:

> When the population converges to local optimum we degenerate the objective function which cause the rapid migration of individuals and enable the population to escape for the local optimum.

The algorithm degenerate the objective function by composing it with Gaussian function which approximate the local optimum. Our deterioration algorithm described in detail in chapter 4 uses Gaussian functions as well (Gaussian function has got many useful properties which makes it well-suited to the fitness deterioration. We describe these characteristics in chapter 4).

Mentioned methods are incorporated directly into the basic cycle of evolutionary algorithm which differs from our *Sequential niching* technique. The sequential niching approach has several advantages:

– it is simple to incorporate in existing optimization methods

– it efficiently finds many local solutions

– it provides reasonable stop criterion which in this case is based on the quality of clusters returned by the clustering algorithm

# 2. Sequential niching algorithm

**Definition. Sequential niching** - the process of iterative degradation of fitness landscape in the areas occupied by clusters of individuals which are assumed to agglomerate inside basins of attraction.

## 2.1. Hybrid Approach

Our deterioration algorithm may be easily incorporated in existing optimization methods. This can be expressed as a general hybrid approach to global optimization in which we do not change the implementation of the used EA but treat it as an integral element of our *sequential niching* process. The following pseudo code shows the general idea behind the *sequential niching*:

1: **while** $i < getIterationCount()$ **do**
2: $execute(evolutionaryAlgorithm)$
3: $population = getPopulation(evolutionaryAlgorithm)$
4: $clusters = cluster(population)$
5: **if** $clusters.isEmpty()$ **then**
6: $break$
7: **end if**
8: $detFitness = performCrunching(clusters, currentFitness)$
9: $saveClusters(clusters)$
10: $updateFitness(detFitness)$
11: **end while**
12: $execute(evolutionaryAlgorithm)$
13: $extractBestClusters()$

The condition in while statement should be treated as control statement rather than the real termination criterion. It may be useful in cases where the clustering algorithm is misconfigured and always returns some clusters in which case the condition would prevent our algorithm to run endlessly. Our actual stop criterion is based on the condition inside the loop (line 5): if the clustering algorithm did not find any group of similar individuals or the group has low quality measures it jumps out of the loop and then we perform the last invocation of the EA in order to increase the probability of finding the local optimum which has not been explored during iterations.

Using this general scheme has one important advantage: we do not have to change existing implementation of a given EA in contrast to standard niching methods which must be incorporated directly in

the evolutionary algorithm. But what is more important, it provides a reasonable stop criterion which in this case is based on the quality of clusters returned by the clustering algorithm.

In the subsequent chapters we will describe components of the *sequential niching* algorithm in more details. In chapter 3 we will discuss the clustering method used in our algorithm and how does it influence the deterioration process. Chapter 4 describes the deterioration algorithm in more detail and chapter 5 shows the results of the algorithm applied to simple multimodal problems.

# 3. Clustering

Clustering algorithms divide a dataset into several disjoint subsets. All elements in such a subset share common features like, for example, spatial proximity. Clustering is used as a stand-alone tool to get insight into the distribution of a data set or as a preprocessing step for other algorithms operating on the detected clusters. The former can be used to determine stop criteria (see section 3.2) and the latter usecase is used in our deterioration schema (see chapter 4).

## 3.1. Cluster Extension

Basin of attraction is a term used in dynamic systems defined as the set of initial conditions leading to long-time behavior that approaches the particular attractor. In the field of global optimization, basin of attraction may be defined in the following way.

**Definition. Basin of attraction** is a subset A of the problem space which contains local solution and for each point which belongs to A the gradient of the objective function at that point will lead towards the solution.

Clusters may be seen as an approximation of the basin of attraction, because the distribution of individuals which flooded to the basins provides useful information about its size and shape. So the clustering algorithm may be used to detect the set of individuals which belongs to the same basin of attraction. Such a set may be later described by extracting some statistical information from that set, e.g the center point, the radius of the set, covariance matrix etc. This is what we called a **Cluster extension**. Not only does our algorithm efficiently solve multimodal problems, it also provides information about detected sets (basins of attraction) in the from of cluster extensions.

## 3.2. Clustering as a Stop Criterion

The termination criterion in classic evolutionary algorithms is hard to define and very often problem dependent, as we do not have any global information about the fitness landscape and therefore we can only compare one solution to another previously founded. Some of the common termination criteria such as:

- maximum computation time
- total number of iterations

– no improvement for a specified number of iterations

are only applicable to a specific problems and none of them can be used as a general stop criterion. The clustering, which is performed in every iteration, on the other hand may give us some clues about the global characteristics of the fitness landscape i.e. when the clustering algorithm performed on the final population finds nothing it is very likely that in previous iterations we have deteriorated the fitness landscape in places where the most desirable solutions reside. This is considered to be true because we are looking for robust solutions which are resistant to noise and lie in basins of attraction which are significantly wide and deep. The population of EA is likely to converge to such solutions, so having found no clusters of individuals after performing the EA in a given iteration shows that the population does not converge to any robust solution. In such circumstances we terminate the main loop of the algorithm.

## 3.3. OPTICS

We have choosen density-base clustering algorihtm called *OPTICS: Ordering Points To Identify the Clustering Structure* [2]. In density clustering clusters are regarded as regions in the data space in which the objects are dense and which are separated by regions of low object density. These regions may have an arbitrary shape and the points inside a region may be arbitrarily distributed.

*OPTICS* is an extendsion to a well-known density clustering algorithm called *DBSCAN*. The basic idea for *DBSCAN* is that for each point of a cluster the neighborhood of a given radius $\epsilon$ has to contain at least a minimum number of points $minPts$.

*OPTICS* works like *DBSCAN* but for an infinite number of distance paramters $\epsilon_i$ which are smaller than a *generating distance* $\epsilon$. The only difference is that we do not assing cluster memberships. Instead, we store the **order** in which the objects are processed (the main priniple is that we always have to select an object which is density-reachable with respect to the lowest $\epsilon$ value to guarantee that clusters with higher density are finished first) and the information which would be used by *DBSCAN* algorithm to assing cluster memberships. This information consists of only two values for each object:

**Definition. core-distance** - the core-distance of an object $p$ is simply the smallest distance $\epsilon'$ between $p$ and an object in its $\epsilon$-neighborhood such that $p$ would be a core object with respect to $\epsilon'$ if this neighbor is contained in $N_\epsilon(p)$. Otherwise, the core-distance is *UNDEFINED*

**Definition. reachability-distance** - the reachability-distance of an object $p$ with respect to another object $o$ is the smallest distance such that $p$ is directly density-reachable from $o$ if $o$ is a core object

This information is sufficient to extract all density-based clusterings with respect to any distance $\epsilon'$ which is smaller that the generating distance $\epsilon$

An advantage of cluster-ordering a data set compared to other clustering methods is that the ordering which might be visualized by *reachability-plot* of ordered points is rather insensitive to the input parameters of the method i.e. the *generating distance* $\epsilon$ and the value for $minPts$. Roughly speaking, the values have just to be *large* enough to yield a good result. The concrete values are not crucial because there is a broad range of possible values for which we always can see the clustering structure of a data set when

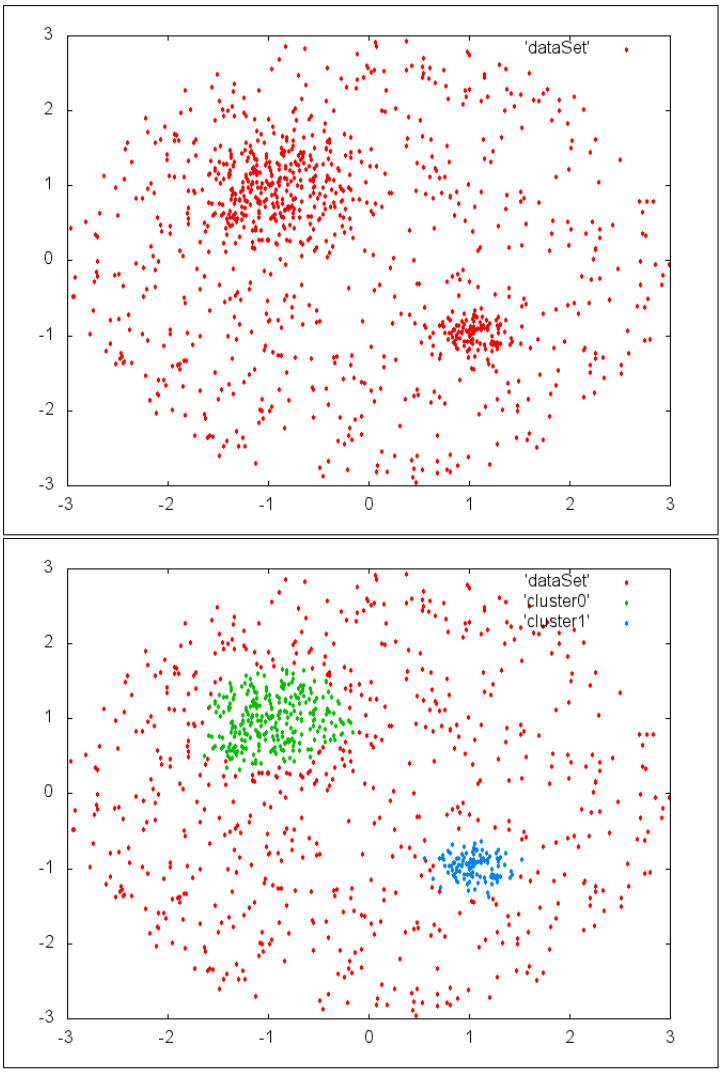


Figure 3.1: Visualization of the DBSCAN algorithm applied to Optics ordering of simple 2-dimensional data set which consists of 1000 points. Optics paramters: $minPts = 20, \epsilon = 1.2$, the two clusters was found using DBSCAN paramters: $\epsilon' = 0.2$

looking at the corresponding *reachability-plot*. Figure 3.1 shows the result of *OPTICS* clustering for a sample set of points. Figure 3.2 shows reachablity plot for various *generating distances* - $\epsilon$.

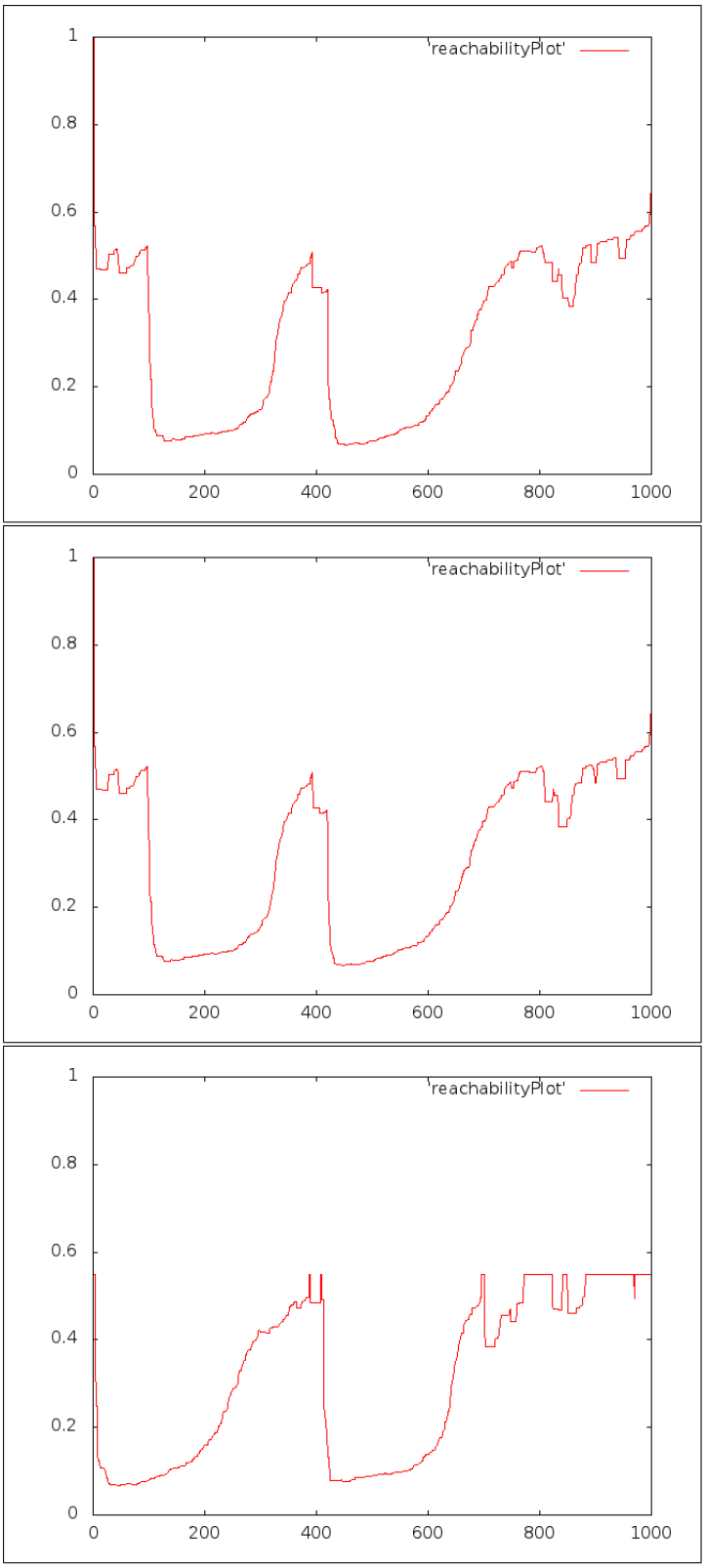


Figure 3.2: Reachability plot for data set presented on figure 3.1. Optics paramters (minPts, $\epsilon$) are: (20, 1.5), (20, 1.0), (20, 0.5) respectively. The two cavities which are visible in each plot depict the two of the clusters on figure 3.1. This proves that there is a large range of values for $\epsilon$ for which the appearance of the reachability plot will not change significantly. (the flat shape of function from the last plot results from the fact that we truncate the reachability-distance to the generating distance $\epsilon$, when the former is greater than the latter)

# 4. 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:

$$F_{k+1} = F_k + \sum_{i=1}^{M} \alpha_i g_i \tag{4.1}$$

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 $searchOp \in Op$.

$$\forall g_1, g_2 \in \mathbb{G} \Rightarrow \exists k \in \mathbb{N} : P(g_1 = Op^k(g_2)) > 0 \tag{4.2}$$

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** [1] 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.

## 4.1. 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 $\epsilon$) [2].

Having the *OPTICS ordering* of the population returned by the EA our method iteratively extract clusters with higher densities by decreasing the *neighborhood radius ($\epsilon$)* (see Figure 4.1), 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**

### 4.1.1. 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

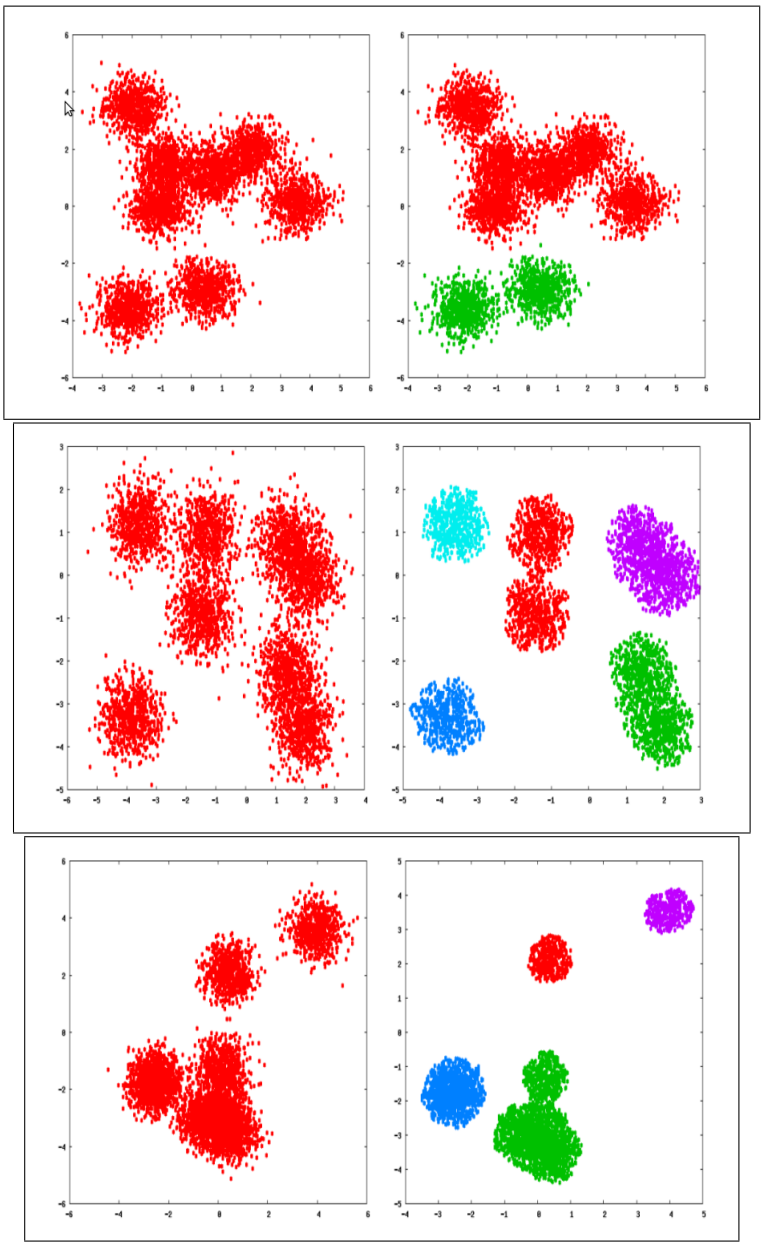

Figure 4.1: The result of extracting clusters with different densities from a random data set. OPITCS ordering parameters: $\epsilon = 1.0$ $minPts = 15$. The values of density radius are: $\epsilon = 0.5$, $\epsilon = 0.25$, $\epsilon = 0.1$ respectively (the second picture in each plot shows extracted clusters). Assuming that clusters are dense in basins of attraction, it would be very inaccurate to create Gaussian *crunching function* for clusters from the first diagram (bear in mind that we create one *crunching function* per cluster)

A class of functions which which are suitable for the fitness deterioration are so called (kernel functions) [20]. 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).

## 4.2. Basic Scheme

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

$$g(x) = -F_k(x_{max}) exp(\frac{-1}{2}(x - \mu)'\Sigma^{-1}(x - \mu)) \tag{4.3}$$

where $F_k$ is a fitness function in $k$th iteration of the algorithm, $\Sigma$ is an **unbiased sample covariance matrix** [19] estimated from the cluster population:

$$\Sigma = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T \tag{4.4}$$

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

$$F_{k+1} = F_k + \sum_{i=1}^{M} g_i \tag{4.5}$$

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

The greate advantage of this algorithm are it's simplicity and it's speed. Howerver, this version of the algorithm may cause strong deformation of the fitness landscape in areas which are distant from the already found clusters, which is unacceptable. To overcome this issue we developed so called *weighted scheme* described in the next section. Disadvantages of the weighted approach is that it is a great deal slower than the *Basic scheme*.

## 4.3. Weighted Scheme

This type of fitness deterioration is more accurate and is likely to produce more stable fitness that *Basic scheme*, cause the latter, as we mentioned in the previous section, may produce sharp peaks in

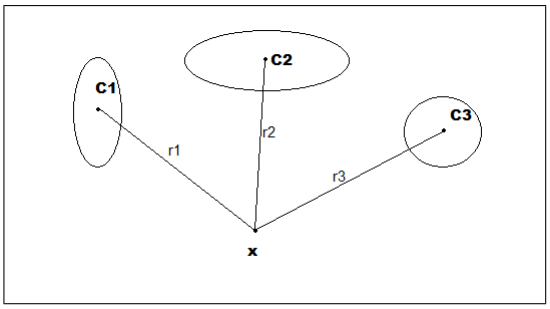


Figure 4.2: The coefficient $\alpha_i$ is inversely proportional to the (a power of) distance from the center of cluster $C_i$ to the $x$. $\alpha_i$ may be seen as the impact $C_i$ has on $x$

the fitness landscape, because of the very aggressive fitness degeneration. *Weighted Scheme* is also more complex couse it generates 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 new (deteriorated) fitness for a given individual $x$:

$$F_{k+1}(x) = F_k(x) + \sum_{i=1}^{M} \alpha_i(x) g_i(x) \tag{4.6}$$

where the $\alpha$-coefficients are given by the following equations:

$$\alpha_1 + \ldots + \alpha_M = 1 \tag{4.7}$$

$$\alpha_i = \xi(\frac{1}{r_i})^p \tag{4.8}$$

which gives:

$$\frac{1}{\xi} = \frac{1}{r_1^p} + \ldots + \frac{1}{r_M^p} \tag{4.9}$$

where $r_i$ is the distance between $x$ and the center of cluster $i$ and $p$ is a power to which the distance is rised. Parameter $p$ is used when an individual finds themselves near one cluster to strengthen the impact of this cluster and to weaken the impact of other clusters. So to compute fitness for a given individual $x$ we have to compute distances between $x$ and clusters. Then we compute $\xi$, which is just a proportionality factor, from the equation $(4.9)$ and lastly we compute $\alpha$-coefficients from $(4.8)$ and the fitness value. Experiments show that values $p \in [1; 1.5)$ yield good results. $\alpha$-coefficients are computed separately for each new individual and this is why this method is more costly than the previous one.

If we look at the equation $(4.6)$ it is clear that regions of domain which are distant from clusters found in previous iterations of the algorithm are very little affected by the crunching functions which is a big advantage over the *Basic scheme*. The *Weighted Scheme* is more smooth and precise, but on the other hand more costly. In general it should do better than *Basic scheme*.

## 4.4. 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 [19], 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* [19] 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*)

$$p_{i1} = \overline{m} + \sqrt{\lambda_i}\overline{v_i} \tag{4.10}$$

$$p_{i2} = \overline{m} - \sqrt{\lambda_i}\overline{v_i} \tag{4.11}$$

  where $\overline{m}$ is the center of the cluster (mean value), $\lambda_i$ 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.

### 4.4.1. 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.

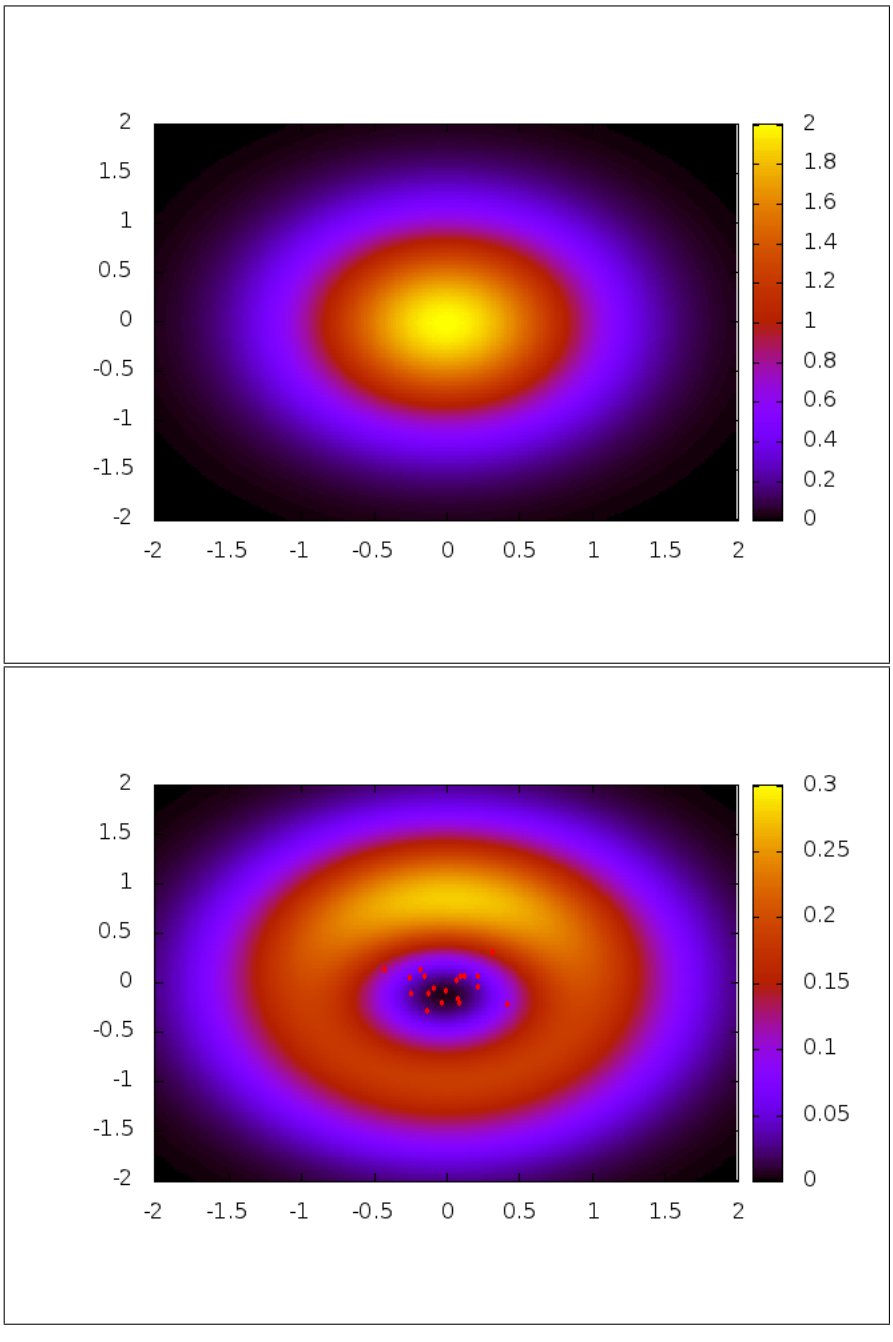


Figure 4.3: The result of basic deterioration scheme with CMA applied to unimodal function: $f(X) = 2e^{-(x^2+y^2)}$. Optics paramters: $minPts = 20, \epsilon = 0.4$, algorithm: SGA, iterationCount=1

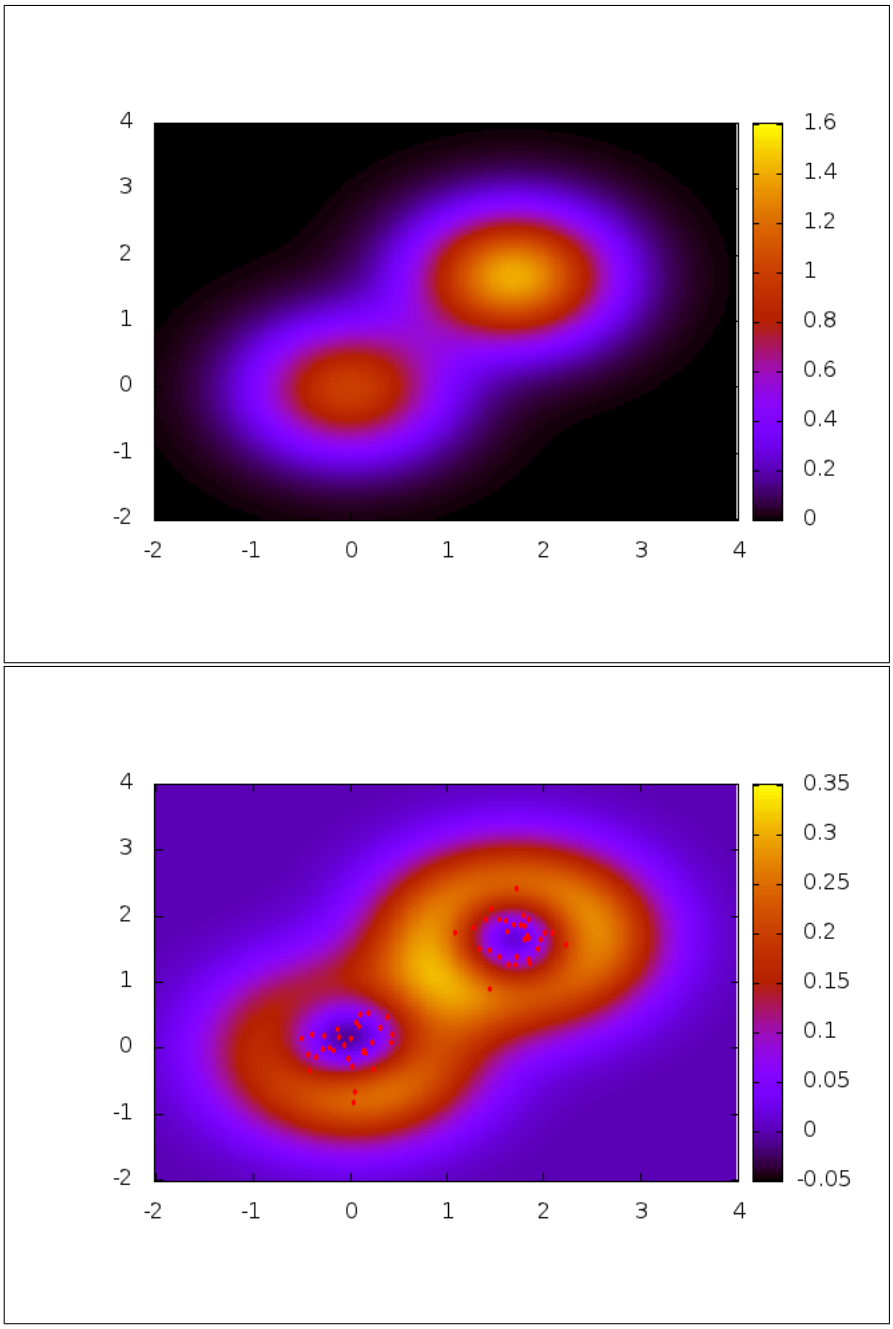


Figure 4.4: The result of basic deterioration scheme with CMA applied to bimodal function: $f(X) = e^{-(x^2+y^2)} + 1.4e^{-((x-1.7)^2+(y-1.7)^2)}$. Optics paramters: $minPts = 20, \epsilon = 0.4$, algorithm: SGA, iterationCount=2

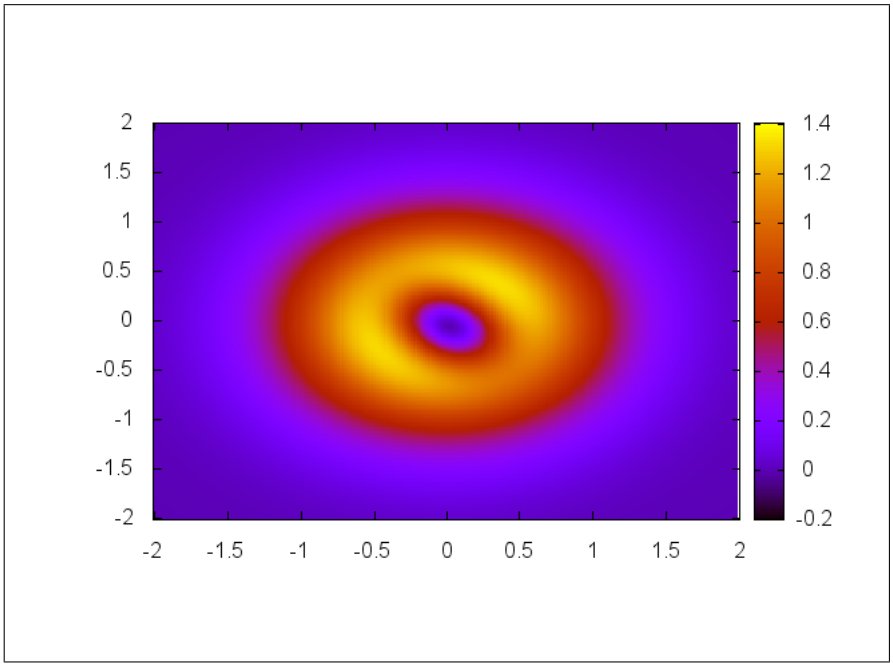


Figure 4.5: The result of basic deterioration scheme without CMA applied to unimodal function from 4.1. Optics paramters: $minPts = 20, \epsilon = 0.4$, algorithm: SGA, iterationCount=2. We may see that the overall landscape decreases only by 30 percent, while using CMA gives us 85 percent of decline.

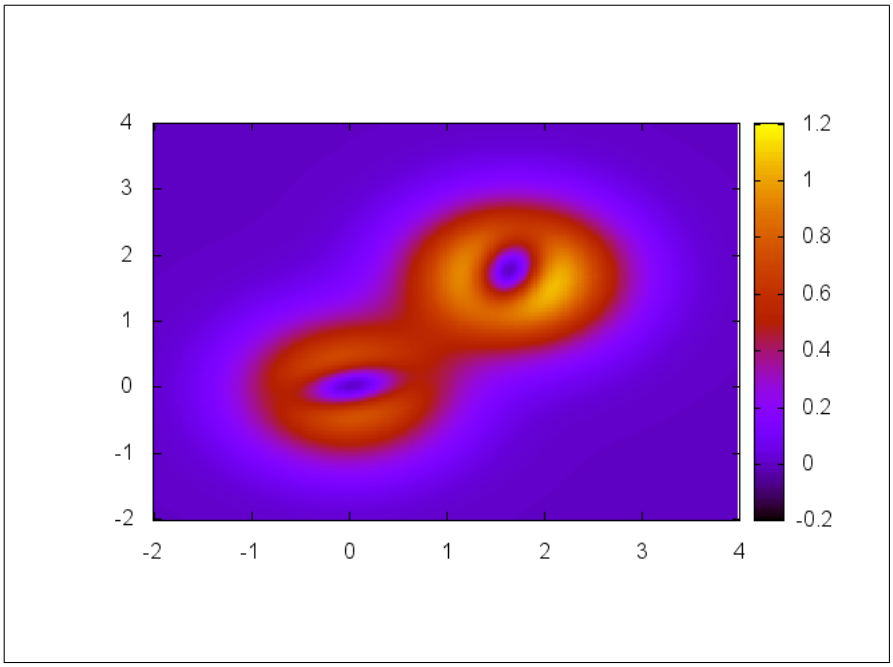


Figure 4.6: The result of basic deterioration scheme without CMA applied to bimodal function from 4.2. Optics paramters: $minPts = 20, \epsilon = 0.4$, algorithm: SGA, iterationCount=2. We see 25 percent of deterioration, while CMA gives us 78 percent when applied to the same case.

# 5. Tested Algorithms

## 5.1. HGS

why HGS is well suited to our algorithm (suitable for clustering, fast convergence in leaves)

## 5.2. Tests

### 5.2.1. Benchmark functions

uni, bi and multimodal functions

### 5.2.2. Accuracy measures

how many optimas have been found, diagrams

### 5.2.3. Efficiency measures

# 6. Implementation

To verify our algorithm we have created a simple, global optimization framework which enabled us to execute and test all of the algorithms described throughout this article.

We decided to use Java [11] as a programming language and runtime environment for this project.

Using the framework we can find optima of real, mulimodal, multidimensional and continuous functions. We just need to provide the framework with the concrete fitness function implementation, and by specifying the problem domain.

The framework principle is to separate the interfaces from implementation. Our API provides interfaces for evolutionary algorithms, fitness assignment and fitness deterioration, genetic operators like selection and reproduction, phenotype, clustering algorithms and many more. This makes it compact, extensible and easy to introduce new implementations. For the detailed description of the classes and interfaces see subsection 'Implementation in Java'.

## 6.1. Architecture

The project consists of six main packages, which communicates with the others through clearly specified interfaces and together constitute a complete, lightweight framework for testing and execution of the global optimization algorithms especially the evolutionary algorithms.

- *Algorithm module* - central module of the framework. Contains implementation of the *Sequential Niching* algorithm as described in chapter 2. The main class in the module *ki.edu.agh.algorithm.SequentialNichingWithOpticsClustering* is responsible for creation of the Spring context and execution of the algorithm.

- *Clustering module* - contains interfaces for clustering algorithms and implementation of OPTICS.

- *Fitness deterioration module* - contains implementation of *FitnessDeterioration* interface. Most important are: *ki.edu.agh.deterioration.SimpleGaussianFitnessDeterioration* and *ki.edu.agh.deterioration.WeightedGaussianFitnessDeterioration*.

- *Evolutionary Algorithms module* - contains all the necessary interfaces for EA, selection and reproduction operators, individuals and phenotypes. Implementations of SGA and Evolution Strategy.

- *Printing Modul* - used to print populations and fitness landscape during the program execution

– *Statistics Utils* - based on JAMA [12]. This package contains many useful statistical functions, e.g. CMA algorithm, covariance matrix estimation, multidimensional Gaussian function, fitness deterioration untilities.

## 6.2. Implementation in Java

The framework was written to be elegant and extensible. The system's modules are loosely coupled and each of system's components has little knowledge of the definitions of other separate components (see previous section). This allows for extensibility in further design. To implement the system we used Spring [14] as an application framework, Maven [15] for project management and build automation, JUnit [18] and Mockito [17] for testing and Git [13] as revision control system. The source code, configuration files and sample results may be found at GitHub [16]: https://github.com/wolny/Fitness-Deterioration

### 6.2.1. Technologies

– Spring [14] - application framework

– Maven [15] - project management and build automation

– Mockito [17] - testing framework

– JAMA [12] - linear algebra package

### 6.2.2. Diagrams

Below you may find class diagrams for each of the module implemented in our framework. It shows a general overview of the structure of a system: used classes, their attributes, operations and the relationships between the classes.

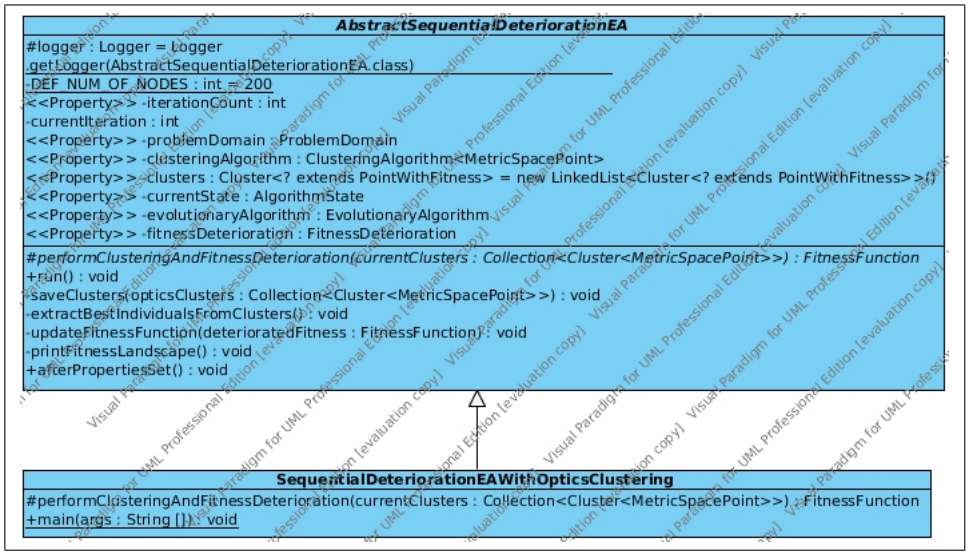


Figure 6.1: Main algorithm package

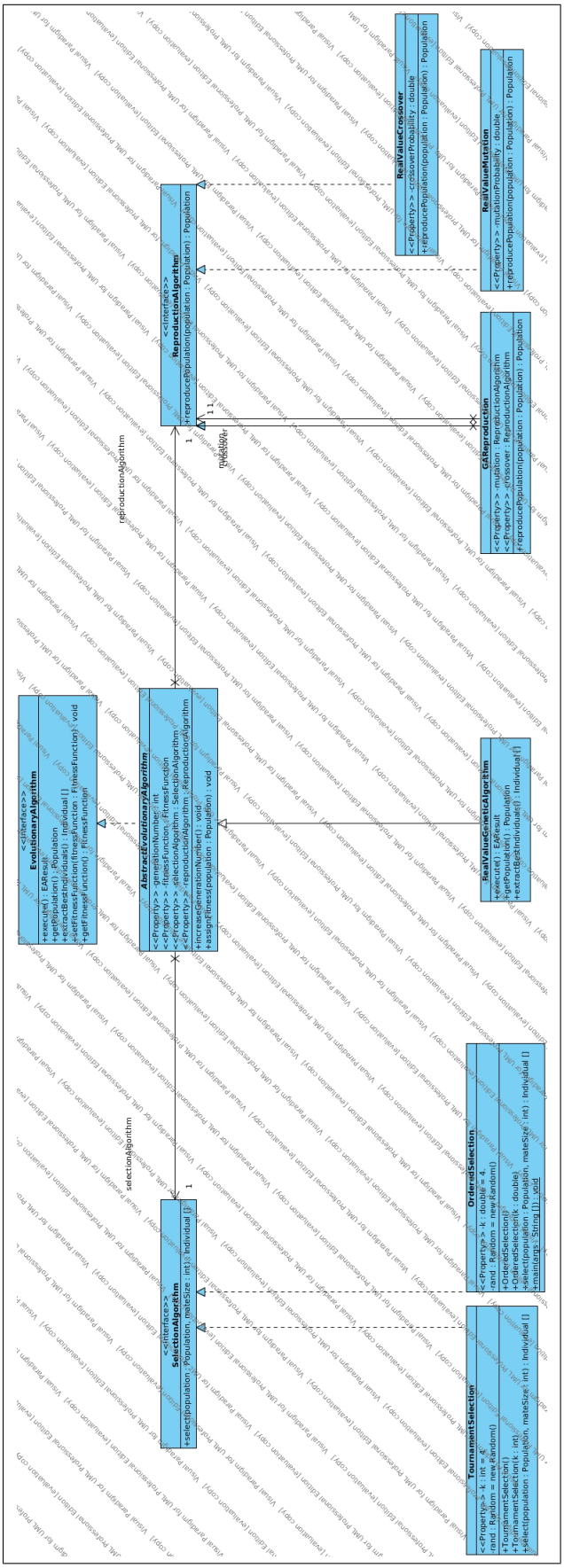


Figure 6.2: Evolutionary algorithms package

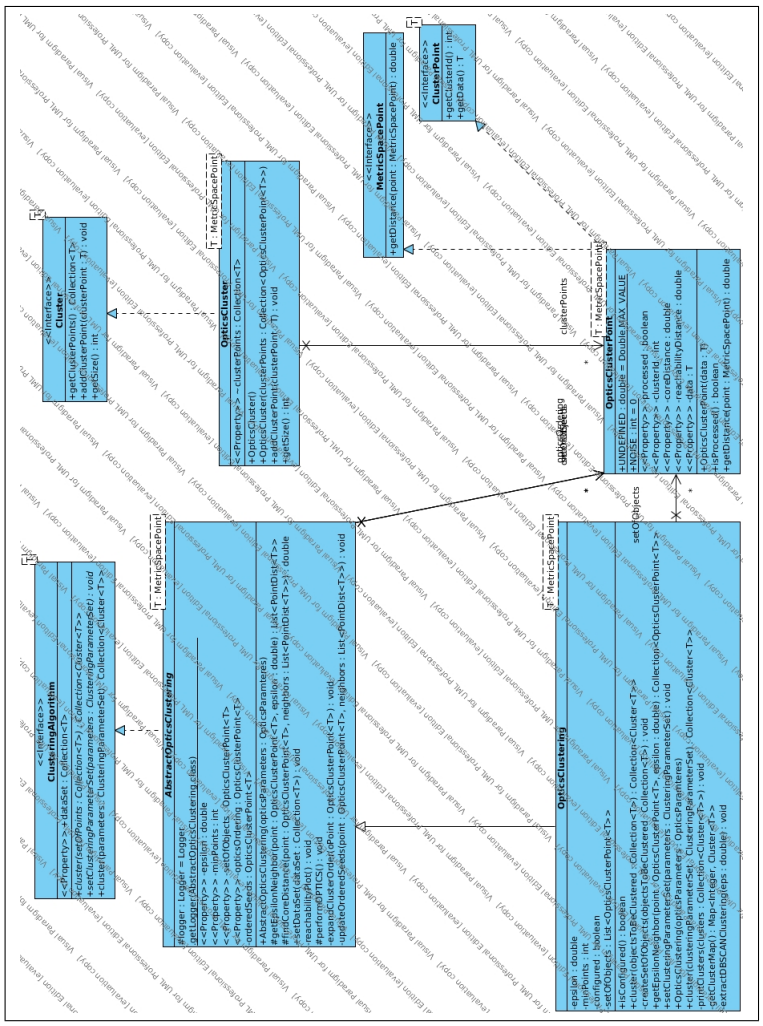

Figure 6.3: Clustering package

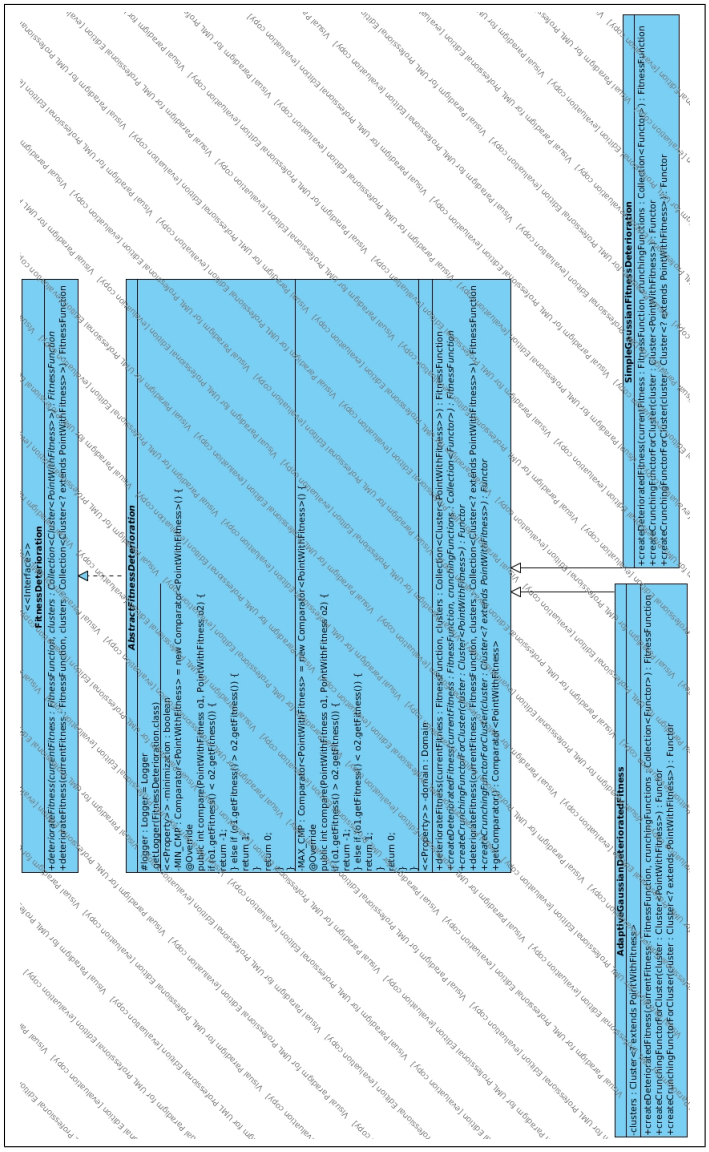

Figure 6.4: Fitness deterioration package

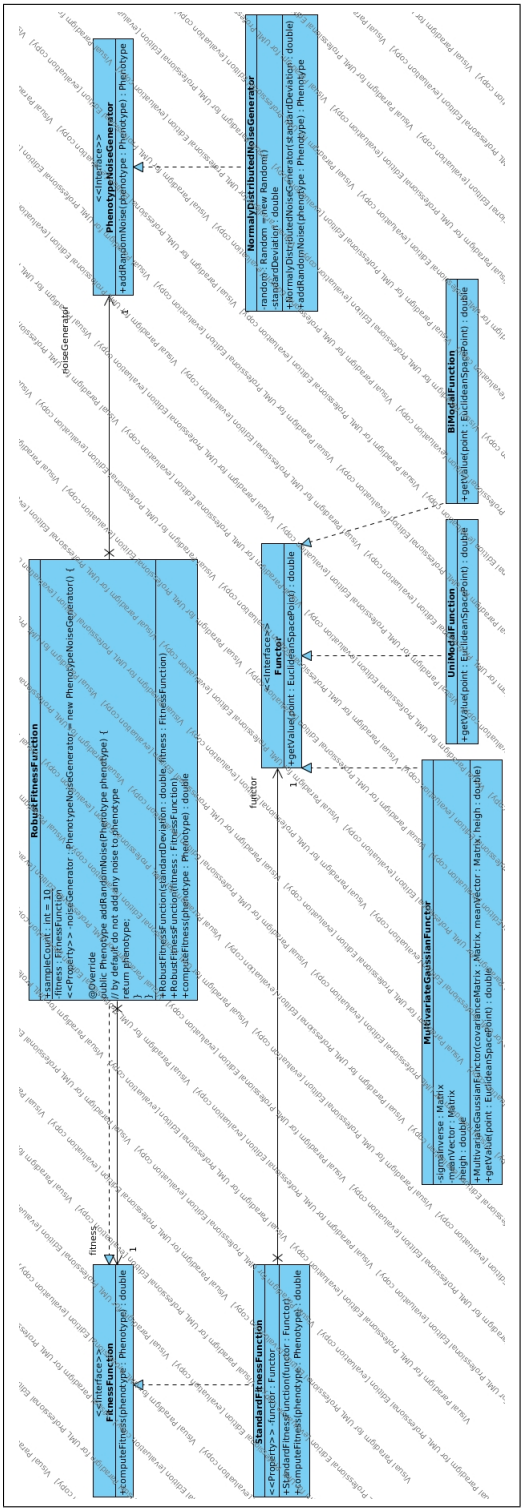


Figure 6.5: Fitness and functors package

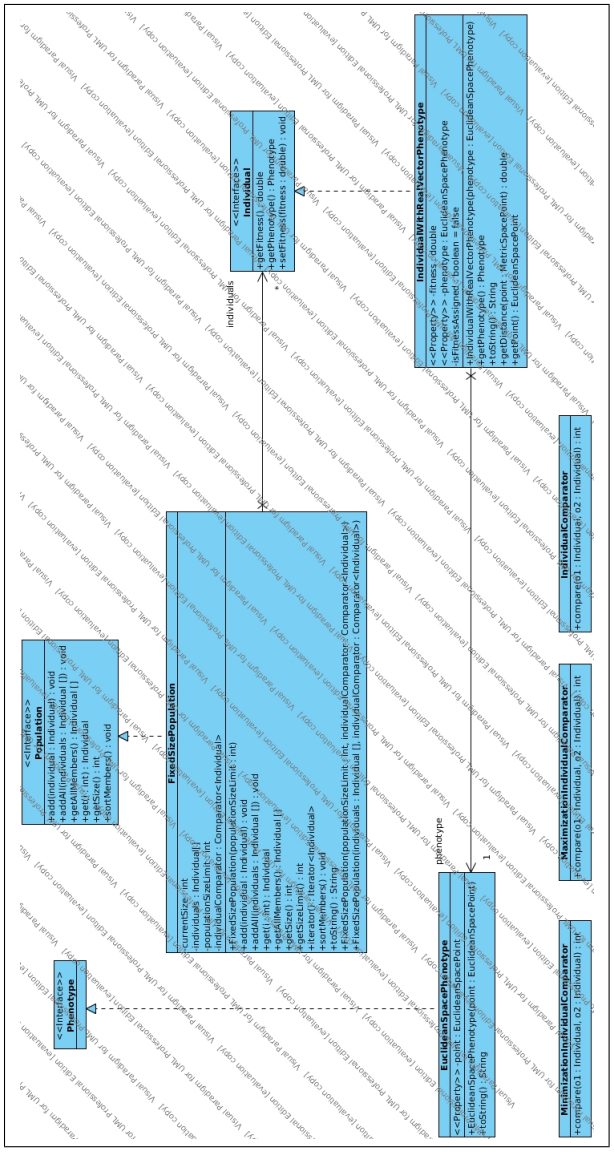


Figure 6.6: Population package

# 7. Conclusions

## 7.1. Summary

The aim of my Master Thesis was to find an effective fitness deterioration algorithm as defined in chapter 4, which minimizes the chance of multiple exploration of the same solution during the course of the *Sequential niching* algorithm. Taking into account the results of tests described in chapter 5 we may come to the conclusion that the goal was reached. However, tests was performed only for simple multimodal problems in which the obtained clusters were convex and provides a lot of information about underlying basins of attraction. Applying the algorithm for more demanding functions should be the next step in the further development of the algorithm.

The assumption that distribution of individuals inside a cluster provides information about its shape oversimplifies the real nature of the evolutionary algorithms, where the population distribution inside basins of attraction is very hard to predict and depends on many factors, including recombination versus mutation rate, selection algorithm, definition of genetic operators such as mutation and crossover. We can expect satisfactory results when choosing proportionate selection and genetic operators which are based on normal distribution. *Fitness Proportionate Selection* usually causes faster convergence to local solutions and normal distribution based reproduction operators tends to produce populations with useful information about fitness landscape.

Choosing weighted crunching functions for *Fitness deterioration* (as described in section 4.3) is always more accurate than the *Basic scheme* described in section 4.2. You can sacrifice accuracy for speed when fitness function evaluation is very costly by using the *Basic scheme*.

The most valuable outcomes of this work are:

– definition and implementation of a hybrid metaheuristic called *Sequential niching* which may be used for solving multimodal optimization problems.

– applying clustering algorithms as a reasonable stop criterion for real-valued evolutionary algorithms

– using properties of *OPTICS ordering* [2] to improve the deterioration process

– definition and implementation of two variants of the fitness deterioration algorithm which might be used in different scenarios depending on the expected speed and accuracy of the global optimization

– definition and implementation of the *Covariance Matrix Adaptation* algorithm which improves the efficiency of the fitness deterioration

– creation of an extensible, lightweight framework which implements all of the algorithms and ideas presented in this work and which can be used for further research

## 7.2. Future Research

This work focuses mainly on the *Fitness Deterioration* algorithm and shows the result of Basic and Weighted variants of the algorithm applied to simple multimodal functions (see chapter 4). Given the *Sequential niching* algorithm described at the beginning of this paper (chapter 2), the main direction of the future research should be to test this algorithm using many different evolutionary algorithms and evolution strategy. The most promising algorithm to choose would be *Hierarchical Genetic Strategy* (HGS) [21].

The Hierarchical Genetic Strategy (HGS) performs efficient concurrent search in the optimization landscape by many small populations. The creation of these populations is governed by dependent genetic processes with low complexity [21]. HGS is an example of parallel genetic algorithms and it performs very well especially in case of problems with many local extrema. Taking into account the fact that HGS is likely to find many solutions in a single run of the algorithm and that the hierarchy of populations generated by the algorithm are rapidly convergent it would be very efficient from the standpoint of the deterioration process.

There are many aspects of the implementation of *Sequential niching* algorithm, described in chapter 6, which can be improved as well. The framework uses very simple concurrency model which may be further developed to improve the overall performance of the algorithm. The most costly stages of the algorithm include: fitness function computations during EA algorithm and creation of *OPTICS ordering* [2]. The former problem might be solved by introducing some of the known parallel genetic algorithms and the latter might be tackled by proper domain decomposition. If we want to introduce a highly concurrent EA model again it would be best to use HGS because of its natural concurrent character.

# A. Installation and Configuration

## A.1. Installation and Execution Instructions

– Configuration file: *application-context.xml* - here you specify used algorithms, problem domain, fitness function and other aspects of the system (see sample configuration below)

– To build the project invoke: *mvn clean install*.

– Program execution: *mvn exec:java* By default the main class of the project is: *ki.edu.agh.algorithm.SequentialDeteriorationEAWithOpticsClustering* (see *pom.xml* for more details)

## A.2. Program Results

Program saves the results in 'diagrams' catalog:

– cluster's members are stored in files of the name 'clusterN', where N is the number of cluster

– populations for each run are stored in files 'populationN', where N is the number of EA run

– fitness landscape is stored in file 'originalFitnessLand'

– reachablity plot for OPTICS algorithm is stored in "reachabilityPlot'

– fitness landscape after each run is stored in 'fintessLand_iterN' where N is the number of EA run

## A.3. Sample Spring configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.s

  <bean
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigu
```

```
    <property name="locations" value="classpath:algorithm.properties" />
  </bean>

  <bean id="algorithm"
    class="ki.edu.agh.algorithm.SequentialNichingWithOpticsClustering">
    <property name="problemDomain">
      <bean class="ki.edu.agh.problem.MultimodalRealSpaceProblem">
        <!-- set domain and functor -->
        <property name="domain">
          <bean class="ki.edu.agh.problem.Domain">
            <property name="multidimensionalCube">
              <list>
                <!-- first dimension interval -->
                <bean class="ki.edu.agh.problem.Interval">
                  <constructor-arg value="-2." />
                  <constructor-arg value="4." />
                </bean>
                <!-- second dimension interval -->
                <bean class="ki.edu.agh.problem.Interval">
                  <constructor-arg value="-2." />
                  <constructor-arg value="4." />
                </bean>
              </list>
            </property>
          </bean>
        </property>
        <property name="fitnessFunction">
          <bean class="ki.edu.agh.fintess.StandardFitnessFunction">
            <constructor-arg>
              <bean class="ki.edu.agh.functors.BiModalFunction" />
            </constructor-arg>
          </bean>
        </property>
        <!-- specify if the the problem is minimization or maximization -->
        <property name="minimization" value="false" />
      </bean>
    </property>

    <property name="iterationCount" value="${algorithm.iterationCount}" />
```

```
    <!-- clustering algorithm -->
    <property name="clusteringAlgorithm">
      <bean class="ki.edu.agh.clustering.optics.OpticsClustering">
        <constructor-arg>
          <bean
            class="ki.edu.agh.clustering.optics.OpticsParamteres">
            <property name="minPoints" value="${optics.min_pts}" />
            <!-- problem's domain dependent -->
            <property name="epsilon" value="${optics.eps}" />
          </bean>
        </constructor-arg>
      </bean>
    </property>

    <!-- specifies evolutionary algorithm used -->
    <property name="evolutionaryAlgorithm" ref="sga" />

    <!-- fitness deterioration algorithm -->
    <property name="fitnessDeterioration">
      <bean
        class="ki.edu.agh.deterioration.SimpleGaussianFitnessDeterioration" />
    </property>
  </bean>

  <bean id="sga" class="ki.edu.agh.evolutionary.algorithm.SGA">
    <!-- SGA configuration -->
  </bean>
</beans>
```

# Bibliography


[1] *Global Optimization Algorithms - Theory and Application* by Thomas Weise

[2] *OPTICS: Ordering Points To Identify the Clustering Structure* by Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jrg Sander

[3] *Evolutionary search with soft selection* by A. Obuchowicz

[4] *Foundations of global genetic optimization* by Robert Schaefer, Henryk Telega

[5] *Introduction to Data Mining* by Tan, Steinbach, Kumar

[6] *Foundations of modern probability* by Kallenberg

[7] *No Free Lunch Theorems for Optimization* by David Wolpert, William Macready

[8] *Niching Methods for Genetic Algorithms* by Samir W. Mahfound

[9] *Genetic algorithms with sharing for multimodal function optimization* by D. E. Goldberg and J. Richardson

[10] *Genetic algorithms with dynamic niche sharing for multimodal function optimization* by B. Miller and M. Shaw

[11] http://www.oracle.com/technetwork/java/index.html

[12] http://math.nist.gov/javanumerics/jama/

[13] http://git-scm.com/

[14] http://www.springsource.org/

[15] http://maven.apache.org/

[16] http://github.com/

[17] http://mockito.googlecode.com/svn/tags/1.8.0/javadoc/org/mockito/Mockito.html

[18] http://www.junit.org/

[19] http://en.wikipedia.org/wiki/Estimation_of_covariance_matrices

[20] http://en.wikipedia.org/wiki/Kernel_statistics

[21] *Hierarchical Genetic Strategy with real number encoding* by Robert Schaefer and Joanna Koodziej