Internal Report 2012–B-2012-02 April 2012

Universiteit Leiden
Opleiding Informatica

Niching for finding robust optima

Frank van Rijn

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Niching for finding robust optima.

Frank van Rijn

March 9, 2012

Preface

In this paper a robustness scheme with niching is presented for extending an evolution strategy to an evolution strategy with a niching technique and robustness scheme. This gives the algorithm more explorative power and better chances of finding robust optima. The proposed schemes are implemented and experiments are run on nine different test functions. The conclusion is that, for the majority of the test functions the extended algorithm perform wells and outperform the benchmark algorithms. However, interestingly enough, the extended algorithm did not outperform the benchmark algorithms on the multipeak functions, although that was the hypothesis.

Chapter 1

Introduction

Evolution strategies are optimization techniques to find a high quality solutions for complex non-linear optimization problems. Given an objective function $f(\mathbf{x}) \rightarrow min, \mathbf{x} \in \mathbb{R}^N$ the evolution strategy (ES), will try to minimize this function and find the best solution. However, the input variables are not always fully controllable. Practical realizations of solutions can deviate, requiring a robust solution that performs well under these deviations.

The normal fitness function does not take the possible deviations into account. Therefore, a robust individual will not be rewarded with a higher fitness. A new fitness function is created that estimates the fitness under deviations. The new fitness function is called the *effective fitness function* [5] and is an expected fitness function. $f_{\text{eff}} = \boldsymbol{E}[f(\mathbf{x}+\mathbf{z})]$, where $\mathbf{z} \sim \mathbf{pdf}(\boldsymbol{\delta})$ is some continuous probability variable.

Robustness schemes proposed in literature [4,9] perform well on locally zooming into the robust optimum, but they are not able to consistently determine the more robust parts of the search area. In this paper it is investigated whether niching can help with identifying the more robust parts of the search area.

Niching is a technique that enforces spatial diversity within a population. In the approach of Shir [7], at every generation niche leaders are chosen that are at least a distance ρ from each other. These spatiality separated niche leaders are thereafter used for generating several separated sub-populations. This prevents the whole population from converging to one point or area, therewith giving the algorithm more exploratory power. In this paper, niching will be combined with sampling schemes for finding robust solutions. The intuition is that this will improve the algorithm's ability to find the more robust parts of the search space, which will result in better robust optima. The studies performed in this paper serve to underpin this hypothesis.

The remainder of the paper is structured as follows: Chapter 2 explains the chosen evolution strategy. Chapter 3 shows how the niching is implemented. Chapter 4 explains the robustness scheme that was used. In Chapter 5 niching and robustness are combined. Chapter 6 presents the experiments and their results, ending with a conclusion and outlook in Chapter 7.

Chapter 2

Evolution Strategy

The Evolution Strategy (ES) that is used for the implementation is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [3]. It is a state-of-the-art algorithm for unconstrained continuous optimization that is suitable for extending it with niching techniques and robustness schemes.

Algorithm 1 CMA-ES

Input parameters(λ, μ)
1: **Initialize** internal parameters: $\sigma, \mathbf{C}$ and $\mathbf{x}_{mean}$
2: **while** not terminate **do**
3: **for** $i = 1 : \lambda$ **do**
4: $\mathbf{z_i} \sim N(\mathbf{0}, \mathbf{I})$
5: $\mathbf{x_i} \leftarrow \mathbf{x}_{mean} + \sigma \cdot \sqrt{\mathbf{C}} \cdot \mathbf{z_i}$
6: $\mathbf{f_i} \leftarrow$ **evaluate**($\mathbf{x_i}$)
7: **end for**
8: $\mathbf{x}_{mean} \leftarrow$weighted average of the μ best individuals
9: **update** σ
10: **update** $\mathbf{C}$
11: **end while**
Output (best $\mathbf{x}$)

Algorithm 1 subsumed the working mechanism of the CMA-ES. After initializing the global stepsize σ, the covariance matrix $\mathbf{C}$ and the recombinant $\mathbf{x}_{mean}$ the main evolution loop begins. The λ offspring are created by adding a random vector $\mathbf{z}, \mathbf{z} \sim N(\mathbf{0}, \sigma\mathbf{C})$ to $\mathbf{x}_{mean}$. The new offspring are evaluated by the fitness function and the μ best are recombined into a new recombinant, $\mathbf{x}_{mean}$, for the next generation. The step size σ and the covariance matrix $\mathbf{C}$ are updated and a new iteration of the loop is started until the stopping conditions are met, then the best found solution is returned. For a detailed descriptions of the working mechanism of the algorithm see [3].

Chapter 3

Niching

A property of evolution strategies is that populations converge with a high probability to a (local) optimum. When that happens diversity of the population is lost. As explained earlier, when looking for robust optima it could be very useful to maintain diversity in the population. A technique to keep diversity in the population is niching.

Niching as defined by [7], works by evolving q different subpopulations parallel to each other, and forcing these subpopulations to stay separated. This creates a $(\{\mu_1, \ldots, \mu_q\}, q \cdot \lambda)$-strategy, where $\mu_1, \ldots, \mu_q$ denotes the number of parents of the subpopulations. Algorithm 2 describes the identification of the best individuals per niche (see appendix A for a matlab implementation of the algorithm).

Algorithm 2 works as follows: It assumes that the population Pop is sorted by fitness (the fittest individual is Pop_1). The population is an array that consists of $\lambda \cdot q$ vectors. Each vector is an candidate solution $\{\mathbf{x_1}, \ldots, \mathbf{x}_{\lambda \cdot \mathbf{q}}\}$. The algorithm calculates the DPS. The DPS is a set of vectors, which are candidate solutions. Initially the DPS is set to $\emptyset$. Then, for every individual i in the population, it is checked, if its distance to all the elements in the DPS is bigger then the niching radius ρ. If this is the case and the number of elements in DPS is smaller than q, then ith individual is also a niche leader and is added to the DPS.

In Figure 3.1, the working mechanism of the algorithm is illustrated. Figure 3.1a shows the population. Then in, Figure 3.1b, the best individual is checked. At this point the DPS is still empty, so its distance to all elements in DPS is larger than ρ. This makes the best individual a new niche leader. It is colored blue. The blue circle around it shows the niching radius and is added to the DPS. In Figure 3.1c, the second best individual is checked and it is within the niching radius of one of the members of the DPS. Hence, it is not a new niche leader and it is not added to the DPS. After that, the next best individual is checked in Figure 3.1d. It lies outside the niching radius of the member in the DPS, so the individual becomes a niche leader. It is colored red and the niching radius is shown around it with a circle, then it is added to the DPS.

Algorithm 2 Dynamic Peak Identification

Input (Pop, λ, q, ρ) {assume that Pop is sorted on fitness}
1: $i \leftarrow 1$
2: $numpeaks \leftarrow 0$
3: DPS $\leftarrow \emptyset$
4: **while** ($numpeaks < q$ **and** $i \leq \lambda \cdot q$) **do**
5: $is\_niche\_leader \leftarrow$ **true**
6: **for** $k = 1 : numpeaks$ **do**
7: **if** $distance(\mathrm{Pop_i}, \mathrm{DPS_k}) \leq \rho$ **then**
8: $is\_niche\_leader \leftarrow$ **false**
9: **end if**
10: **end for**
11: **if** $is\_niche\_leader ==$ **true then**
12: DPS $\leftarrow$ DPS $\cup$ $\mathrm{Pop_i}$
13: $numpeaks \leftarrow numpeaks + 1$
14: **end if**
15: $i \leftarrow i + 1$
16: **end while**
Output DPS

This process continues in the other Figures 3.1e and 3.1f.

3.1 Niching radius

An important parameter for niching is the niching radius ρ. In [7], a suggestion is made how to set this parameter. Given the number of peaks q in the search space, every peak is considered to have a n-dimensional hypersphere surrounding the peaks with radius ρ, which encloses $\frac{1}{q}$ of the volume of the search space. Two assumptions are made: 1) the number of peaks q can be estimated or is given, and 2) the peaks lie at least a distance of 2ρ from each other.

The volume of a hypersphere is

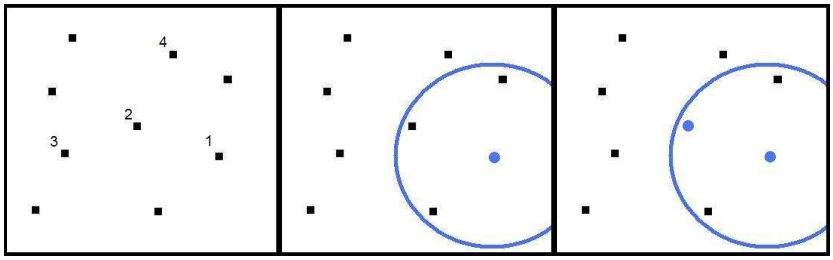
$$V = c(n)r^n. \tag{3.1}$$

where $c(n)$ is a positive constant depending on the dimensions and r is given by:

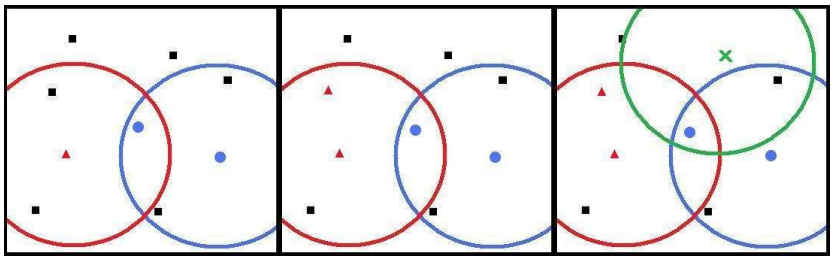
$$r = \frac{1}{2}\sqrt{\sum_{i=1}^{n}(\mathbf{xl_i} - \mathbf{xu_i})^2}, \tag{3.2}$$

where **xu** denotes the upper bound and **xl** denotes the lower bound. V is divided into q spheres

$$c(n)\rho^n = \frac{1}{q}c(n)r^n. \tag{3.3}$$



(a) Population (b) First niche found (c) Second best individual is inside niching radius



(d) New niche found (e) Inside niche (f) New niche found

Figure 3.1: Illustration of the dynamic peak identification procedure with three niches found

To calculate ρ:

$$\rho = \frac{r}{\sqrt[n]{q}}. \tag{3.4}$$

When dealing with a search space that is a cube the volume of the circumscribed sphere is taken as the volume of the search space, however, the volume of the circumscribed sphere is much larger then the volume of the cube. This can result in a ρ that is too large.

For example the 10-dimensional problem with the lower bound for every dimension being -5 and the upper bound for every dimension 5 and dividing that into 4 volumes, $q = 4$.

$$\rho = \frac{\frac{1}{2}\sqrt{\sum_{i=1}^{n}(\mathbf{xl_i} - \mathbf{xu_i})^2}}{\sqrt[n]{q}} = \frac{15.8114}{\sqrt[10]{4}} = 13.7646. \tag{3.5}$$

Since the maximum distance for two points the search space is 31.6228 and the suggested niching radius for $q = 4$ is 13.7646, the chances that the peaks lie a distance of at least 2ρ from each other are not so high. The sphere function only has one peak, but the ρ is not depended on the function only on the bounds of the search space and the number of niches.

As an alternative we propose another procedure to calculate ρ, by taking the incircle of the search space instead of the circumscribed circle. The r is now be given by:

$$r = \frac{1}{2} \cdot \frac{1}{n} \sum_{i=1}^{n} |\mathbf{xu_i} - \mathbf{xl_i}|. \tag{3.6}$$

For the 10-dimensional problem with the same bounds this results into a ρ of:

$$\rho = \frac{\frac{1}{2} \cdot \frac{1}{n} \sum_{i=1}^{n} |\mathbf{xu_i} - \mathbf{xl_i}|}{\sqrt[n]{q}} = \frac{5}{\sqrt[10]{4}} = 4.3528, \tag{3.7}$$

which appears to be a better value for ρ.

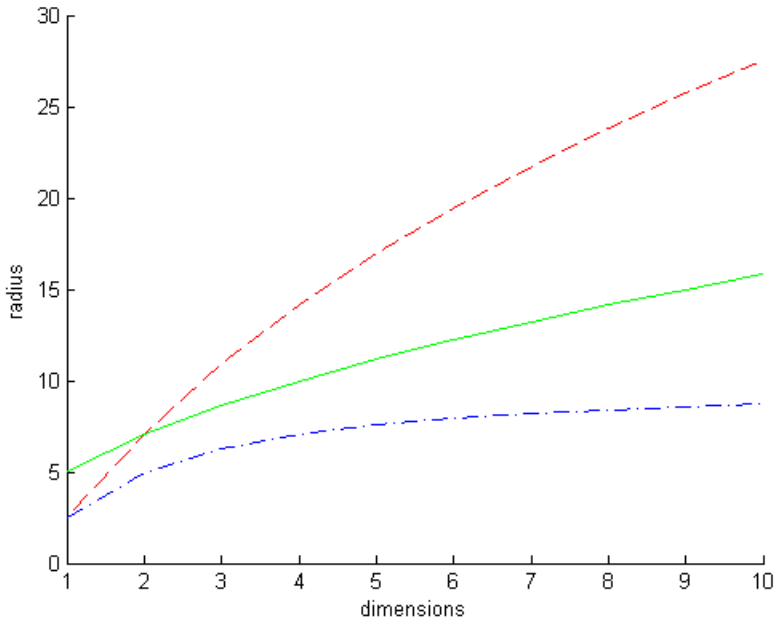


Figure 3.2: A plot of two different niching radii versus the diagonal of the search space. The blue slasheddotted line shows the niching radius derived from the inner circle. The red slashed line indicates th niching radius derived from the circumscribed circle. The green line indicates $\frac{1}{q}$ times the length of the diagonal of the search space, $q = 4$ the lowerbounds are -10 and the upperbounds are 10.

Figure 3.2, shows that ρ taken from the circumscribed circle grows faster for higher dimensions than ρ derived from the incircle. The green constant line shows the growth of the diagonal of the search space. The blue line stays closer to the green line than the red line.

3.2 CMA-ES with niching

Algorithm 3 shows the niching technique implemented into the CMA-ES. To add the niching technique to the CMA-ES, some changes have to be made to the CMA-ES as presented in Algoritm 1. Instead of having one $\sigma, \mathbf{x}_{mean}$ and $\mathbf{C}$, every niche has its own $\sigma, \mathbf{x}_{mean}$ and $\mathbf{C}$. When creating new offspring the $\sigma, \mathbf{x}_{mean}$ and $\mathbf{C}$ of the niche leader are used.

Algorithm 3 CMA-ES with niching

Input parameters(λ, μ, q)
1: **Initialize** internal parameters: $\sigma_1, \ldots, \sigma_q, \mathbf{C}_1, \ldots, \mathbf{C}_q, \mathbf{x}_{mean_1}, \ldots, \mathbf{x}_{mean_q}, \rho$
2: **while** not terminate **do**
3: **for** $j = 1 : q$ **do**
4: **for** $i = 1 : \lambda$ **do**
5: $\mathbf{z}_{j,i} \sim N(\mathbf{0}, \mathbf{I})$
6: $\mathbf{x}_{j,i} \leftarrow \mathbf{x}_{mean_j} + \sigma_j \cdot \sqrt{\mathbf{C}_j} \cdot \mathbf{z}_{j,i}$
7: $\mathbf{f}_{j,i} \leftarrow \mathbf{evaluate}(\mathbf{x}_{j,i})$
8: **end for**
9: **end for**
10: $(\mathbf{x_{1:1}}, \ldots, \mathbf{x_{q:\lambda}}) \leftarrow \mathbf{sort}(\mathbf{x_{1,1}}, \mathbf{f_{1,1}}), \cdots, (\mathbf{x_{q,\lambda}}, \mathbf{f_{q,\lambda}})$ {In ascending order}
11: DPS = **dpi(x)**
12: **for** $j = 1 : q$ **do**
13: $\mathbf{x}_{mean,j} \leftarrow \mathrm{DPS_j}$
14: **update** σ_j
15: **update** $\mathbf{C_j}$
16: **end for**
17: **end while**
Output (best individual)

Two problems occur when updating $\mathbf{x}_{mean}$ and creating offspring. How to deal with the case when there are less then μ individuals in a niche? This problem can be overcome by taking the mean of the individuals that exist in the niche, instead of taking the mean of μ individuals.

The other problem is more difficult. When the new offspring are created and regrouped in q niches, it is possible that two individuals that are now in the same niche have different parents. Updating σ and $\mathbf{C}$ becomes a problem, because it is not reasonable to combine two (or multiple) covariance matrices and two step sizes. The solution is to allow for only one parent per niche, $\mu = 1$. Changing the CMA-ES from a (μ, λ)-strategy into the CMA-ES with niching, which is a $(\{\mu_1, \ldots, \mu_q\}, q \cdot \lambda)$-strategy, where every $\mu = 1$.

Chapter 4

Robustness

A robust solution is a solution that performs well when the variables are effected by a disturbance $\boldsymbol{\delta}$. In Figure 4.1, two peaks are shown, the left peak has a better optimum. The right peak is more robust, and might be a better robust solution depending on the disturbance $\boldsymbol{\delta}$. Note that is assumed that $\boldsymbol{\delta}$ is known, but that assumption holds for many of real life problems, without a known $\boldsymbol{\delta}$ it would not be possible to have an effective fitness function.

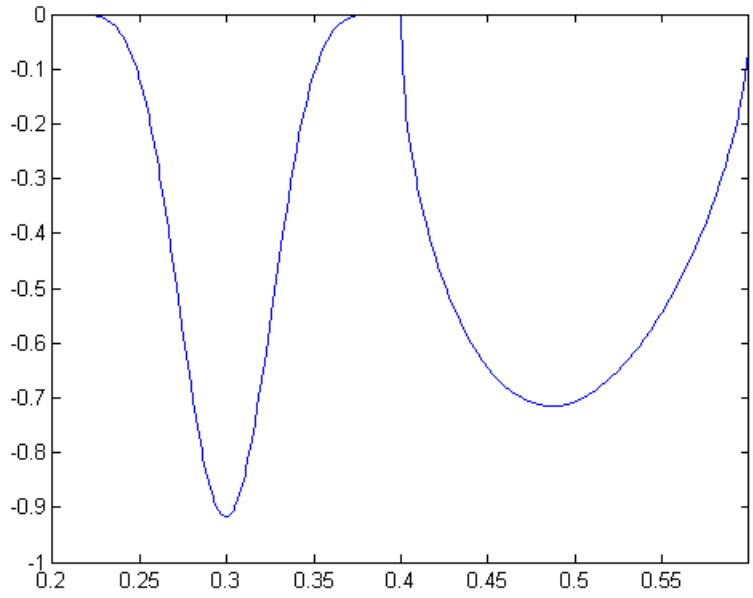


Figure 4.1: Two peaks: The left peak has the better optimum for minimization. The right peak is the more robust peak.

Robustness is important because it is not always possible to control the input variables. Especially in industrial applications where candidate solutions cannot be realized with arbitrary precision.

When there are a lot of peaks it is very hard to find the more robust peaks. Robustness schemes perform well on zooming in locally into the robust optimum. But they perform poorly in identifying the more robust parts of the search space.

What niching can add to this is, is that keeping the population spread out over the search space it finds more peaks. This should result in better chances of finding more robust peaks.

The robustness scheme that was chosen is the multi-evaluation-model (MEM). It does not just calculate the fitness from an individual $\mathbf{x}$, but it takes m different random samples of the disturbance $\boldsymbol{\delta}$ from an individual. MEM calculates the fitness of those m points and takes the mean of those values as the fitness for the individual $\mathbf{x}$, i.e.,

$$f_{\text{eff}}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x} + \boldsymbol{\delta_i}), \boldsymbol{\delta_i} \sim \boldsymbol{\delta}. \tag{4.1}$$

The evaluation can be done with the recycling of the disturbance samples or for every individual take a new disturbance sample. In the first case, denoted MEM^{+}, the same disturbance is added to every individual and its fitness is then calculated. In the second case, denoted MEM^{-}, every individual has its own disturbance which is evaluated. For this algorithm the recycling method, MEM^{+}, is chosen.

Chapter 5

Niching and finding robust optima

In Algorithm 4, the result of combining the MEM scheme with Algorithm 3 is shown (see appendix B for a matlab implementation of the algorithm).

In the MEM scheme, the parameter m, the number of samples, needs to be set. To find the best setting for m the algorithm, with $q = 4$, is tested on a 10 dimensional instance of Branke's multipeak problem. It's a modification of the Branke function [1] and it has many peaks. The peaks are positioned like a grid. All the peaks are positioned at the -1 and 1 coordinates. The more -1 coordinates the more robust the peak is. Hence, the global optimum is $\mathbf{x} = (-1, -1, -1, -1, -1, -1, -1, -1, -1, -1)$. So each found peak has a score for the number of dimensions the coordinate is smaller then zero. The most robust peak has a score of 10.

The algorithm is a run 100 times for each setting of m. The most robust niche was identified by counting in how many dimensions $\mathbf{x} < 0$. For how many dimensions that was true is plotted in Figure 5.1

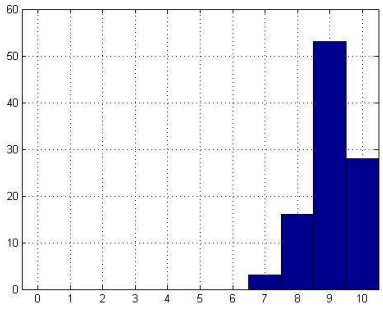
As can be seen in Figure 5.1, $m = 1$ is the best. It finds the most robust peak the most often and has a lower average fitness. MEM with $m = 1$ is a special case of the MEM-scheme, called single evaluation method (SEM).

Algorithm 4 CMA-ES with niching and the MEM evaluation scheme for finding robust optima

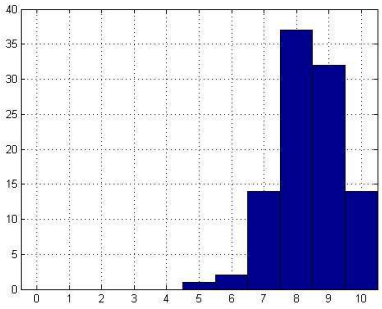
Input parameters(λ, μ, q, m)

Initialize internal parameters: $\sigma_1, \ldots, \sigma_q, \mathbf{C}_1, \ldots, \mathbf{C}_q, \mathbf{x}_{mean_1}, \ldots, \mathbf{x}_{mean_q}, \rho$
while not terminate **do**
  $\boldsymbol{\delta_l} \sim \boldsymbol{\delta}, l = 1, \cdots, m$
  for $j = 1 : q$ **do**
    for $i = 1 : \lambda$ **do**
      $sumf = 0$
      $\mathbf{z}_{j,i} \sim N(0, \mathbf{I})$
      $\mathbf{x}_{j,i} \leftarrow \mathbf{x}_{mean_j} + \sigma_j \cdot \sqrt{\mathbf{C}_j} \cdot \mathbf{z}_{j,i}$
      for $l = 1 : m$ **do**
        $sumf = sumf +$ **evaluate**$(\mathbf{x}_{j,i} + \boldsymbol{\delta_l})$
      end for
    end for
    $\mathbf{fo}_{j,i} \leftarrow \frac{sumf}{m}$
  end for
  $(\mathbf{x}_{1:1}, \ldots, \mathbf{x}_{q:\lambda}) \leftarrow$ **sort**$(\mathbf{x}, \mathbf{fo})$ {In ascending order}
  $DPS =$ **dpi(x)**
  for $j = 1 : q$ **do**
    update $\mathbf{x}_{mean,j}$
    update σ_j
    update $\mathbf{C_j}$
  end for
end while

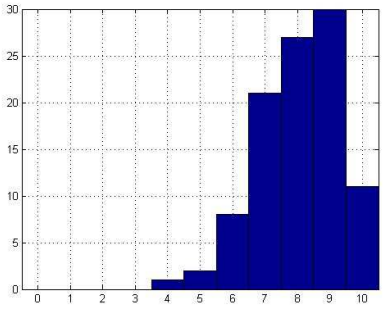
Output (best individual)



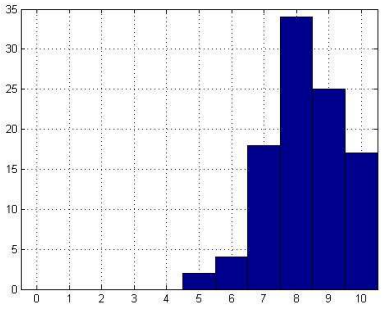
(a) $m = 1$, average fitness = 0.4601



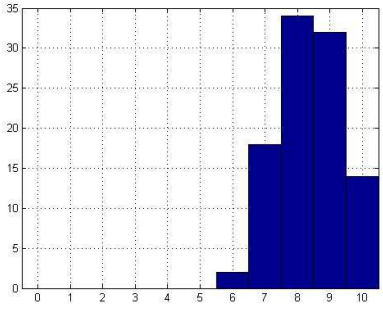
(b) $m = 2$, average fitness = 0.4791



(c) $m = 3$, average fitness = 0.4845



(d) $m = 4$, average fitness = 0.4730



(e) $m = 5$, average fitness = 0.4649

Figure 5.1: Number of times peak found versus robustness score of the peaks for different instances of the CMA-ES with niching and the MEM$^+$ for finding robust optima.

Chapter 6

Experimental results

6.1 Proof of concept

An experiment on a two-dimensional problem is done to see if the CMA-ES with niching and MEM has potential to find the more robust parts of the search space. Three algorithms are compared namely, the CMA-ES, the CMA-ES with niching and the CMA-ES with niching and the MEM$^+$. The three algorithms are run on a two-dimensional instances of Branke's multipeak function, and again by counting in how many dimensions the solution is smaller then zero it is calculated which peak is found. The more dimensions smaller than zero, the more robust the optimum is. The two dimensional instance of Branke's multipeak function has four peaks: $(1,1),(-1,1),(1,-1),(-1,-1)$. Each peak is given a score equal to the number of -1 they have. The three algorithms need to find as often as possible the peak with score two. The settings for the algorithms are: $q = 2$ and $m = 1$. For the niching algorithms the peak with the higher score is chosen. The results are plotted in Figure 6.1.

The results shown in Figure 6.1, indicate that niching alone is not enough to identify the more robust parts of the search area. The algorithm with niching and MEM however, performs better then the normal CMA-ES in finding the more robust parts of the search area. The differences are marginal, and a more systematic testing approach is needed and will be conducted in Chapter 6.2.

6.2 Experiments

The CMA-es with niching and the MEM evaluation scheme is compared with a normal CMA-ES, a CMA-ES with niching and a CMA-ES with the MEM scheme (without niching). The four algorithms are compared on nine artificial test functions.

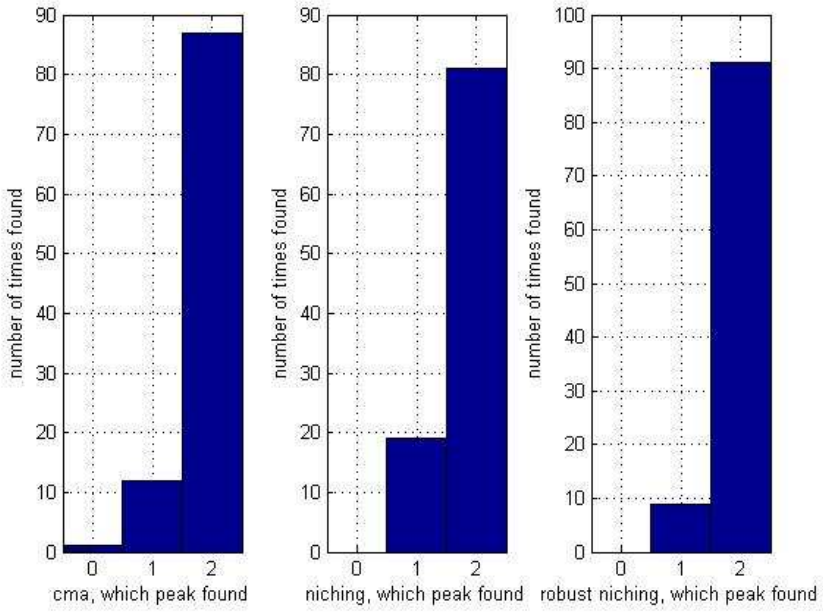


Figure 6.1: Number of times peak found vs robustness score of the peaks.

6.3 Test functions

f1: Sphere problem (from [6])

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2,$$
$$\mathbf{x} \in [-5,5]^N, \boldsymbol{\delta} \sim U(-\mathbf{1},\mathbf{1}), \text{ and } \mathbf{x_{ro}} = \mathbf{0}.$$

f2: Heavyside sphere problem (from [4])

$$f_2(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1+e^{(\frac{2}{5}(x_i+3))}} + \frac{1}{1+e^{10(x_i-4)}},$$
$$\mathbf{x} \in [-10,10]^N, \boldsymbol{\delta} \sim U(-\mathbf{1},\mathbf{1}), \text{ and } \mathbf{x_{ro}} = [1,1,0,\cdots,0].$$

f3: Sawtooth problem (from [2])

$$f_3(\mathbf{x}) = 1 - \frac{1}{n}\sum_{i=1}^{n} \begin{cases} (x_i + 0.8) & \text{if } x_i < 0.2 \wedge x_i \geq -0.8 \\ 0 & \text{else} \end{cases},$$
$$\mathbf{x} \in [-1,1]^N, \boldsymbol{\delta} \sim U(-\mathbf{0.2},\mathbf{0.2}), \text{ and } \mathbf{x_{ro}} = \mathbf{0}.$$

f4: Volcano problem (from [5])

$$f_4(\mathbf{x}) = \begin{cases} \sqrt{||\mathbf{x}||} - 1 & \text{if } ||\mathbf{x}|| > 1 \\ 0 & \text{else} \end{cases},$$
$$\mathbf{x} \in [-10, 10]^N, \boldsymbol{\delta} \sim U(-\mathbf{1.5}, \mathbf{1.5}), \text{ and } \mathbf{x_{ro}} = \mathbf{0}.$$

f5: Modded branke multipeak problem (from [1])

$$f_5(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} c - \begin{cases} 1 - (x+1)^2 & \text{if } x_i < 0 \wedge x_i \geq -2 \\ c \cdot 2^{(-8 \cdot |x_i - 1|)} & \text{if } x_i \geq 0 \wedge 2 \geq x_i \end{cases},$$
$$c_1 = 1.3, \mathbf{x} \in [-2, 2]^N, \boldsymbol{\delta} \sim U(-\mathbf{0.5}, \mathbf{0.5}), \text{ and } \mathbf{x_{ro}} = -\mathbf{1}.$$

f6: Pickelhaube problem (from [5])

$$f_6(\mathbf{x}) = c_{1a} - max(f_{base}, f_{1a}, f_{1b}, f_2),$$
$$f_{1a} = c_{1a} \cdot \left(1 - \frac{||\mathbf{x}+5||}{5 \cdot \sqrt[4]{N}}\right),$$
$$f_{1b} = c_{1b} \cdot \left(1 - \frac{||\mathbf{x}+5||}{5 \cdot N^2}\right),$$
$$f_2 = c_2 \cdot \left(1 - \frac{||\mathbf{x}-5||}{5 \cdot (\sqrt{N})^{d_2}}\right),$$
$$fbase = 0.1 \cdot \exp(-\tfrac{1}{2} \cdot ||\mathbf{x}||),$$
$$c_{1a} = \tfrac{5}{5-\sqrt{5}}, c_{1b} = \tfrac{625}{624},$$
$$c_2 = 1.5975528761621545, d_2 = 1.1513175769876054,$$
$$\mathbf{x} \in [-10, 10]^N, \boldsymbol{\delta} \sim U(-\mathbf{1}, \mathbf{1}), \text{ and } \mathbf{x_{ro}} = \mathbf{5}.$$

f7: FNIM f2 problem

$$f_7(\mathbf{x}) = -\frac{(a - x_2^2) + \frac{1}{n} \sum_{i=3}^{n} x_i^2}{(x_i^2 + b) - x_1^2},$$
$$a = 5, b = 1, \mathbf{x} \in [-5, 5]^N, \boldsymbol{\delta} \sim U(-\mathbf{1}, \mathbf{1}), \text{ and } \mathbf{x_{ro}} = \mathbf{5}.$$

f8: Multipeak f1 problem (from [8])

$$f_8(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} e^{(-2 \ln 2 (\frac{x-0.1}{0.8})^2} \sqrt{|\sin(5\pi x_i)|} & \text{if } 0.4 < x_i \geq 0.6 \\ e^{(-2 \ln 2 (\frac{x-0.1}{0.8})^2} \sin^6(5\pi x_i) & \text{else} \end{cases},$$
$$\mathbf{x} \in [0, 1]^N, \boldsymbol{\delta} \sim U(-\mathbf{0.0625}, \mathbf{0.0625}), \text{ and } \mathbf{x_{ro}} \approx \mathbf{0.4911}.$$

f9: Multipeak f2 problem

$$f_9(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} 2 \sin(10 \exp(-0.2 x_i) \cdot x_i) \cdot \exp(-0.25 x_i).$$
$$\mathbf{x} \in [-10, 10]^N, \boldsymbol{\delta} \sim U(-\mathbf{1}, \mathbf{1}), \text{ and } \mathbf{x_{ro}} = \mathbf{5}.$$

In Figure 6.2 the nine test problems are shown in two dimensions.

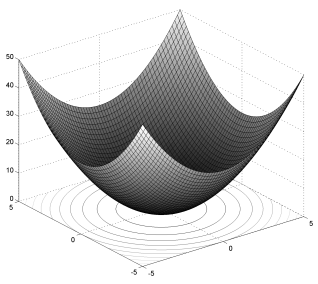
6.4 Results

Each of the four algorithms is run 25 times on each of the nine test problems. Each run has 10,000 function evaluations. The parameters q and m are respectively set to 4 and 3. In Figures 6.3 to 6.11, the average fitness versus the function evaluations are shown.

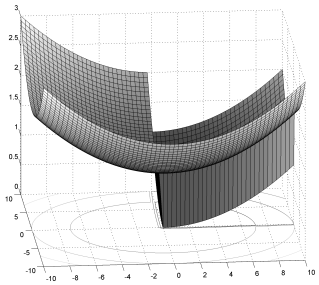
6.5 Discussion of the results

In Figures 6.3 to 6.11, it can be seen that the CMA-ES with niching and the MEM evaluation scheme performs the best on four out of the nine test functions, namely the heavy side sphere problem, the volcano problem, the FNIM problem and Multipeak f2. The CMA-ES with the MEM evaluation scheme, without niching, only is the best for two test problems, namely the sawtooth problem and the multipeak f1 problem. The CMA-ES with niching and MEM scheme, outperforms the CMA-ES with the MEM scheme in a total of 6 out of 9. Adding niching give the algorithm an improvement for the majority of the test problems.

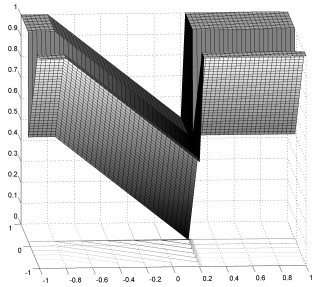
What is very interesting to see is that the CMA-ES with niching and the MEM evaluation scheme does not perform well on Branke's multipeak function and the Multipeak f1 function. While the intuition was that niching would help in a multipeak problems it seems like that is not always the case. The reason for this is not clear. It is possible that a niche leader gets stuck at a local minimum that has a distance to the global minimum that is smaller then ρ. In that case the niching radius prevents other niches from finding the global minimum. The niche which has the global minimum in its niching radius might have zoomed in too far at a local minimum to be able to jump to the global minimum. This presumption gets reinforced by the fact that the CMA-ES with niching and MEM does perform well on the Multipeak f2 function, where the global minimum is located further away from other peaks.



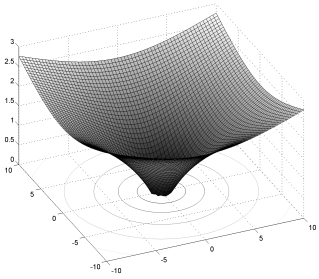
(a) Sphere problem, f_1.



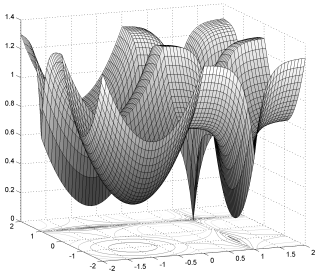
(b) Heavy side sphere problem f_2.



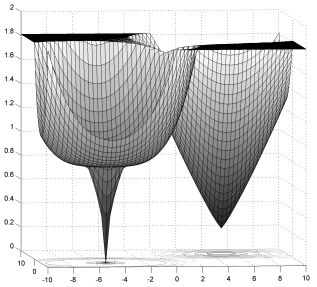
(c) Sawtooth problem, f_3.



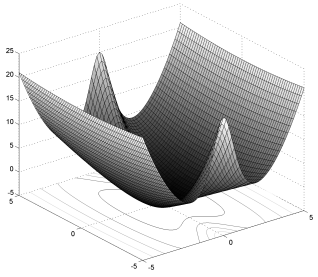
(d) Volcano problem, f_4.



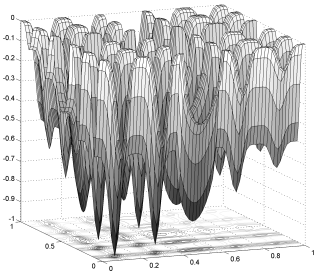
(e) Branke multipeak problem, f_5.



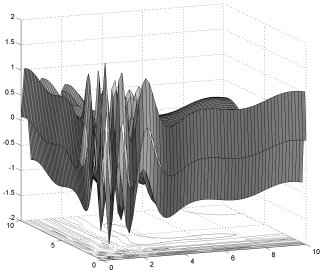
(f) Pickelhaube problem, f_6.



(g) FNIM f2 problem, f_7.



(h) Multipeak f1 problem, f_8.



(i) Multipeak f2 problem, f_9.

Figure 6.2: Two dimensional representations of the nine test problems.

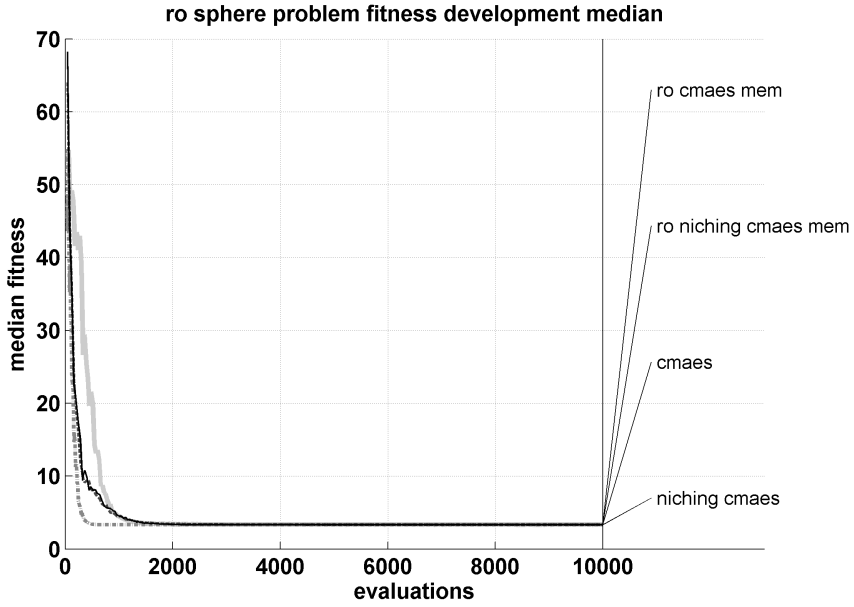


Figure 6.3: Sphere problem, f_1.

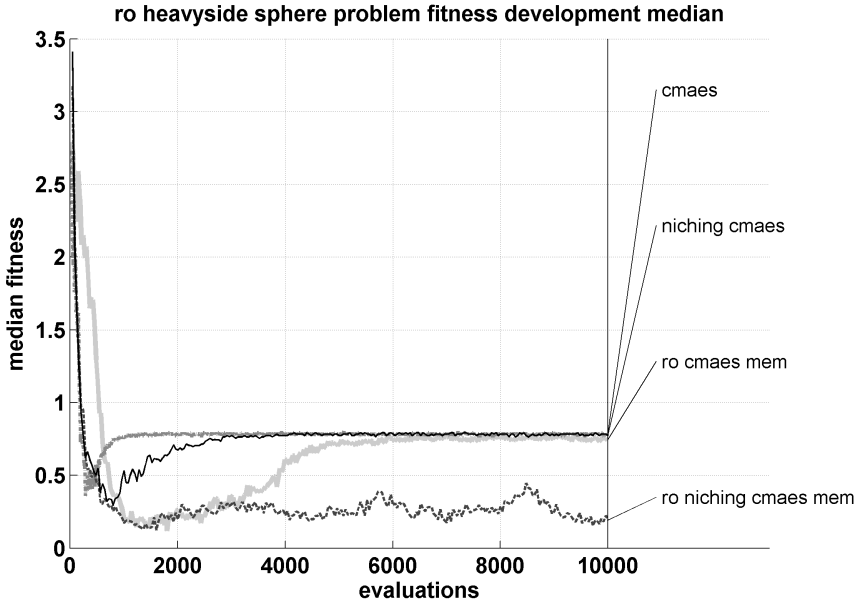


Figure 6.4: Heavy side sphere problem f_2.

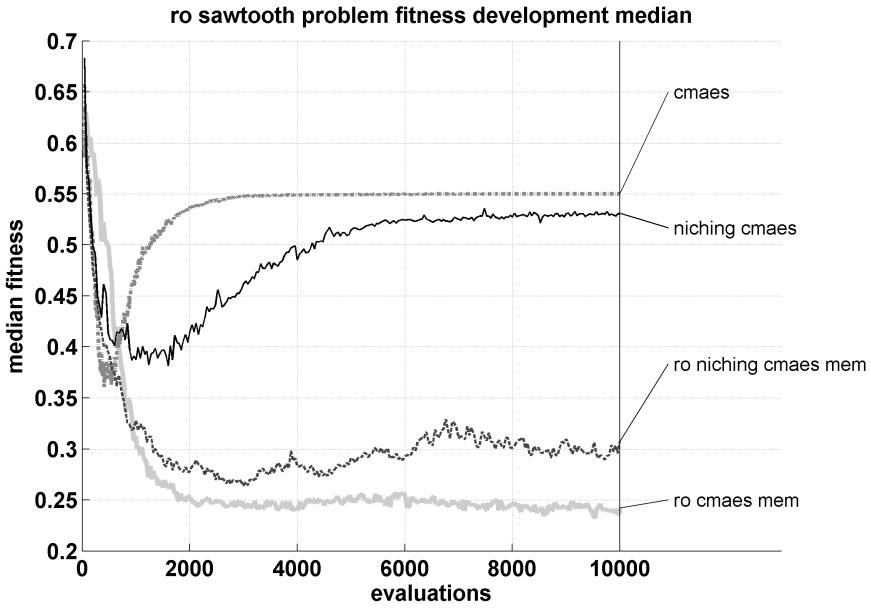


Figure 6.5: Sawtooth problem, f_3.

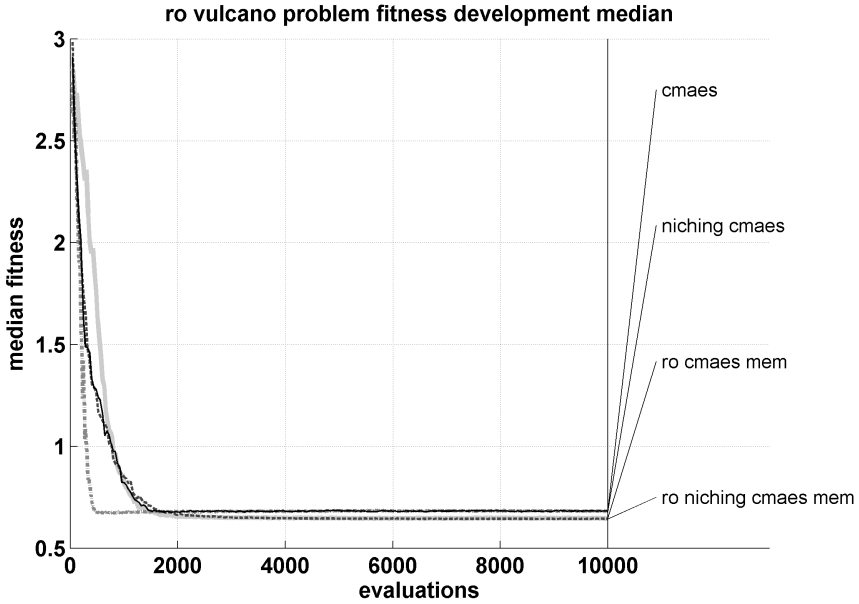


Figure 6.6: Volcano problem, f_4.

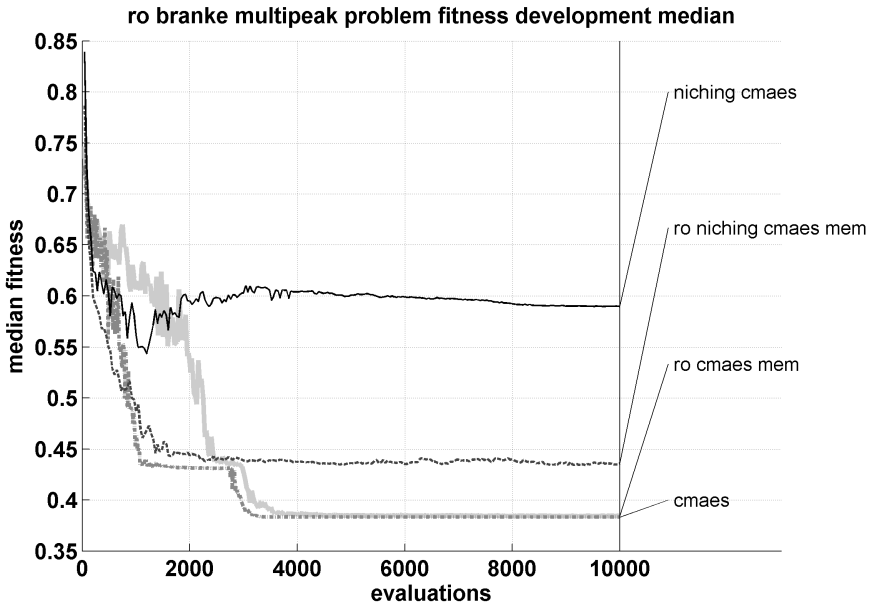


Figure 6.7: Branke multipeak problem, f_5.

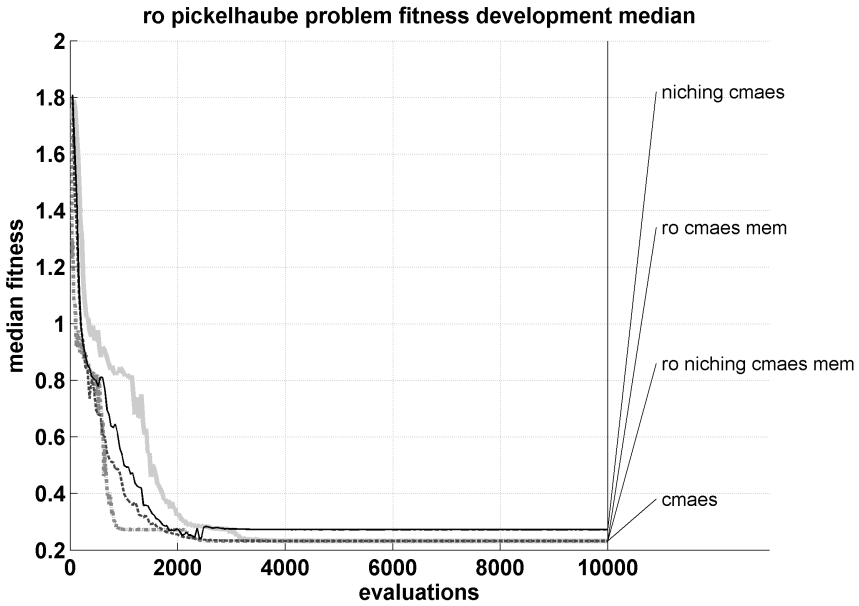


Figure 6.8: Pickelhaube problem, f_6.

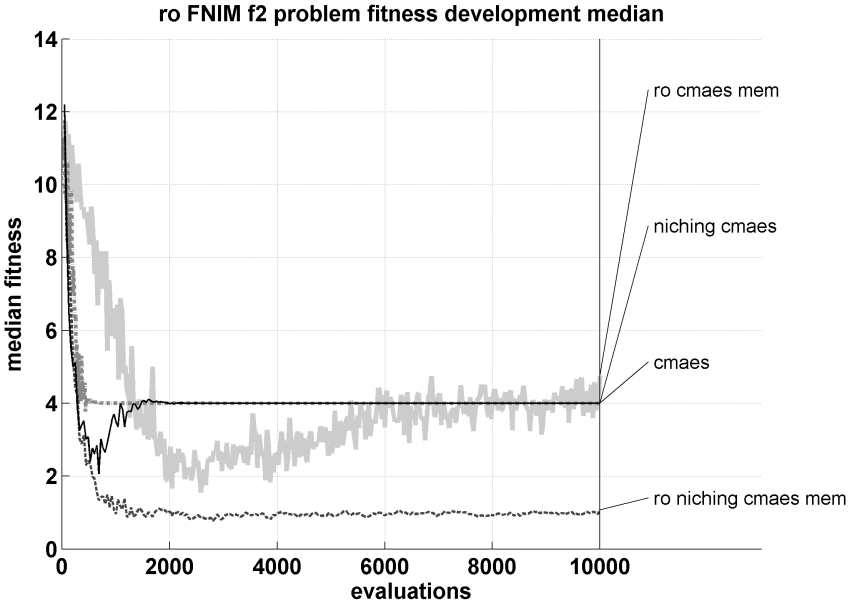


Figure 6.9: FNIM f2 problem, f_7.

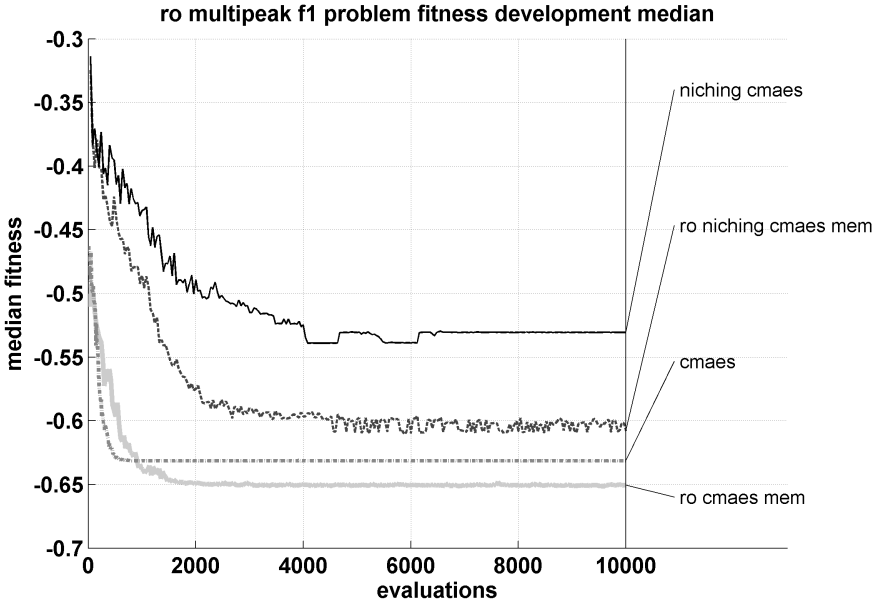


Figure 6.10: Multipeak f1 problem, f_8.

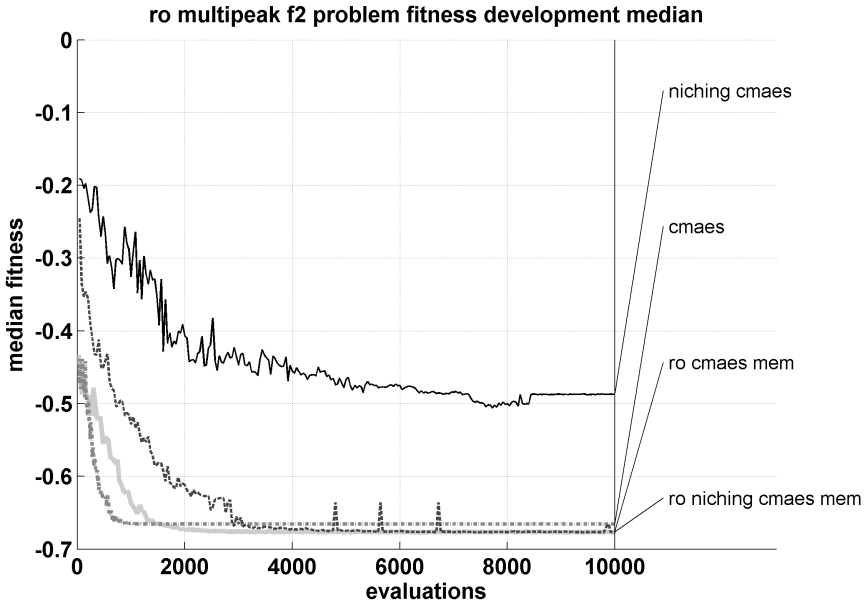


Figure 6.11: Multipeak f2 problem, f_9.

Chapter 7

Conclusion and Outlook

In this paper we have proposed an CMA-ES with niching and the MEM evaluation scheme. The proposed strategy has been tested against three other versions of the CMA-ES algorithm on nine 10 dimensional test problems. It has shown that the CMA-ES with niching and the MEM evaluation scheme can outperform on the majority of the test problems, but the original hypothesis is not completely confirmed. Because the original idea was that niching would help with finding the more robust parts of the search area, and thus perform better on the multipeak problems, but it does not perform better on the multipeak problems. The reason for this could not be found.

For future work it would be interesting to find the exact reason why the CMA-ES with niching and MEM does perform well on functions with sharp edges, but not so well on multipeak functions, while the intuition is the opposite.

Other future work could consist of exploring the possibilities of increasing the size of μ per niche. The problem of having multiple $\sigma's, \mathbf{C}'s$ can be overcome by different ways. For example, taking the $\mathbf{C}$ and σ of the best individual of each niche, or taking the average of them. It is possible that this could improve the algorithm.

Another thing that is be worth looking into, is the role of q. How to set this parameter to have the most chance of finding the robust peak, but not wasting too many function evaluations. Having a variable q sounds like viable option. Starting with many niches to have big explorative power and then towards the end of the budget decreasing the size to continue with the best niches.

Bibliography

[1] J. Branke. Creating Robust Solutions by Means of Evolutionary Algorithms. In *Parallel Problem Solving from Nature (PPSN V)*, LNCS, pages 119–128. Springer-Verlag, 1998.

[2] J. Branke. Reducing the Sampling Variance when Searching for Robust Solutions. In *Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 235–242. Morgan Kaufmann, 2001.

[3] N. Hansen. The CMA Evolution Strategy: A Tutorial. 2011.

[4] Johannes Kruisselbrink, Michael Emmerich, and Thomas Bäck. An archive maintenance scheme for finding robust solutions. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature – PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 214–223. Springer Berlin / Heidelberg, 2011.

[5] Johannes W. Kruisselbrink, Edgar Reehuis, André Deutz, Thomas Bäck, and Michael Emmerich. Using the uncertainty handling cma-es for finding robust optima. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 877–884, New York, NY, USA, 2011. ACM.

[6] J.W. Kruisselbrink, M.T.M. Emmerich, A.H. Deutz, and T. Bäck. A Robust Optimization Approach using Kriging Metamodels for Robustness Approximation in the CMA-ES. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1–8, 2010.

[7] Ofer M. Shir and Thomas Bäck. Niching with derandomized evolution strategies in artificial and real-world landscapes. 8:171–196, March 2009.

[8] O.M. Shir, M.T.M Emmerich, and T. Bäck. Adaptive Niche Radii and Niche Shapes Approaches for Niching with the CMA-ES. *Evolutionary Computation*, 18(1):97–126, 2010.

[9] Shigeyoshi Tsutsui and Ashish Ghosh. *Effects of adding perturbations to phenotypic parameters in genetic algorithms for searching robust solutions*, pages 351–365. Springer-Verlag New York, Inc., New York, NY, USA, 2003.

Appendix A

Dynamic peak identification

```
function [num_peaks, dps] = dynamic_peak_identification(n, q, rho, xo, sortindex)
% [] = dynamic_peak_identification()
% Input:
%  - N                                  - Number of dimensions
%  - n                                  - Size of the population lambda * q
%  - q                                  - Number of peaks to identify
%  - rho                                - The niching radius
%  - xo                                 - The current population
%  - sortf                              - Order of the indices
%
% Output:
%  - num_peaks                          - Number of peaks found
%  - niches                             - Niches found

  i = 1;
  num_peaks = 0;
  dps = -1 * ones(q,1);
  while (num_peaks < q && i <= n)
    is_niche_leader = true;
    k = 1;
    while (k <= num_peaks && is_niche_leader == true)
      if (norm(xo(:,dps(k)) - xo(:,sortindex(i))) < rho)
        is_niche_leader = false;
      end
      k = k+1;
    end
    if (is_niche_leader)
      num_peaks = num_peaks + 1;
      dps(num_peaks) = sortindex(i);
    end
    i = i + 1;
  end
end
```

Appendix B

CMA-ES with niching and MEM

```
function [stat] = ro_niching_cmaes_mem(problem_specification,
                                       algorithm_parameters, run_parameters)
% [stat] = cmaes(problem_specification,
%                                       run_parameters, algorithm_parameters)
%
% Implementation of the mem scheme and niching into CMA-ES algorithm
%
% Input:
% - problem_specification            - A struct containing the problem
%                                                 definition with:
%   - problem_type                   - A string holding description of
%                                                 the problem type
%   - problem_name                   - A string holding the name of
%                                                 the test function
%   - objective_function             - A function handle to the
%                                                 objective function
%   - N                              - The number of dimensions
%                                                 of the search space
%   - lb                             - A vector of the upper bounds of
%                                                 the search interval
%   - ub                             - A vector of the lower bounds of
%                                                 the search interval
%
% - run_parameters                   - A struct containing the run
%                                                 parameters with:
%   - max_generations_termination    - The maximum number of generations
%   - max_evaluations_termination    - The maximum number of evaluations
%   - min_fitness_termination        - Stop when a solution is found below
%                                                 this value
%   - history_statistics             - Maintain history statistics
%   - internal_parameters_statistics - Maintain statistics of the internal
```

```
32 %                                                        parameters
33 %   - report_intermediate            - Report statistics while running
34 %   - report_after_termination       - Report after termination
35 %   - report_fct                     - A handle to the report function
36 %   - restart_log_intermediate       - Maintain a logfile to allow for
37 %                                                        restarts
38 %   - restart_log_after_termination  - Store a logfile to allow for
39 %                             restarts after completing an optimization run
40 %   - restart_log_logfile            - Filename of the restart logfile
41 %
42 % - algorithm_parameters             - A struct containing algorithm
43 %                                                      specific parameters
44 %   - bch_fct                        - A handle to the box constraint
45 %                                                      handling function
46 %   - sampling_fct                   - The sampling function used for
47 %                  obtaining the samples for the robustness approximations
48 %   - m                              - The number of samples used for the
49 %                                                 robustness approximations
50 %   - reuse_disturbances             - Specify whether or not to use the
51 %                  same disturbances for all individuals in the population
52 %
53 % Output:
54 % - stat                    -
55 %
56 % Last modified: December 9, 2011
57
58   % Problem specification
59   fitnessfct = problem_specification.objective_fct;
60   N = problem_specification.N;
61   lb = problem_specification.lb;
62   ub = problem_specification.ub;
63   ur_sigma = problem_specification.ur_sigma;
64
65   % Run parameters
66   stopeval = run_parameters.max_evaluations_termination;
67   stopgen = run_parameters.max_generations_termination;
68   minfitness = run_parameters.min_fitness_termination;
69   history_statistics = run_parameters.history_statistics;
70   internal_parameter_statistics =
71                             run_parameters.internal_parameter_statistics;
72   report_intermediate = run_parameters.report_intermediate;
73   report_after_termination = run_parameters.report_after_termination;
74   report_fct = run_parameters.report_fct;
75   restart_log_intermediate = run_parameters.restart_log_intermediate;
76   restart_log_after_termination =
77                             run_parameters.restart_log_after_termination;
78   if (restart_log_intermediate || restart_log_after_termination)
79     restart_log_logfile = run_parameters.restart_log_logfile;
80   end
81
```

```
82   % Algorithm parameters
83   bch_fct = algorithm_parameters.bch_fct;
84   sampling_fct = algorithm_parameters.sampling_fct;
85   m = algorithm_parameters.m;
86   q = algorithm_parameters.q;
87
88   %count times of niche restart
89   restart = 0;
90   % Set lambda, mu and the weights for recombination
91   r = 0.5 * norm(ub - lb);
92   rho = r / q^(1/N);
93   lambda = 4 + floor(3 * log(N));
94   mu = 1; %only works for mu = 1
95   weights = log(mu + 1) - log(1:mu)';
96   weights = weights / sum(weights);
97   mueff = sum(weights)^2 / sum(weights.^2);
98
99   % Set parameters
100  cc = 4 / (N + 4);
101  cs = (mueff + 2) / (N + mueff + 3);
102  mucov = mueff;
103  ccov = (1 / mucov) * 2 / (N + 1.4)^2 + (1 - 1 / mucov)
104                          * ((2 * mueff - 1) / ((N + 2)^2 + 2 * mueff));
105  damps = 1 + 2 * max(0, sqrt((mueff - 1) / (N + 1)) - 1) + cs;
106  chiN = N^0.5 * (1 - 1 / (4 * N) + 1 / (21 * N^2));
107
108  % Initialize xmean and step size
109  if (isfield(algorithm_parameters, 'xmean_init'))
110    xmean = algorithm_parameters.xmean_init;
111  else
112    xmean = repmat(lb, 1, q) + repmat(ub - lb, 1, q) .* rand(N,q);
113  end
114
115  if (isfield(algorithm_parameters, 'sigma_init'))
116    sigma = algorithm_parameters.sigma_init;
117  else
118    sigma = repmat((norm(ub - lb)) / (3 * q * sqrt(N)), 1, q);
119  end
120
121  % Test reuse_disturbances parameter
122  if (isfield(algorithm_parameters, 'reuse_disturbances'))
123    reuse_disturbances = algorithm_parameters.reuse_disturbances;
124  else
125    reuse_disturbances = true;
126  end
127
128  % Initialize matrices and vectors of the CMA-ES
129  pc = zeros(N,q);
130  ps = zeros(N,q);
131  B = ones(N,N,q);
```

```
132   D = ones(N,N,q);
133   C = ones(N,N,q);
134   for i=1:q,
135     pc(:,i) = zeros(N,1);
136     ps(:,i) = zeros(N,1);
137     B(:,:,i) = eye(N,N);
138     D(:,:,i) = eye(N,N);
139     C(:,:,i) = eye(N,N);
140   end
141
142   % Initialize counters
143   evalcount = 0;
144   gencount = 0;
145
146   % Statistics administration parameters
147   stat.optimizer_name = 'ro_niching_CMA-ES';
148   stat.run_status = 'Incomplete';
149   estimated_stopeval = max(stopeval, stopgen * lambda);
150   estimated_stopgen = max(stopgen, ceil(stopeval / lambda));
151   stat.gencount = 0;
152   stat.evalcount = 0;
153   stat.x_opt = zeros(N,q);
154   stat.f_opt = zeros(1,q);
155   stat.rho = rho;
156   if (history_statistics)
157     stat.evalvsgen = zeros(1, estimated_stopgen);
158     stat.hist_x_opt = zeros(N, q, estimated_stopgen);
159     stat.hist_f_opt = zeros(q, estimated_stopgen);
160     stat.hist_x = zeros(N, estimated_stopeval);
161     stat.hist_f = zeros(1, estimated_stopeval);
162     stat.hist_xmean = zeros(N, estimated_stopgen);
163   end
164   if (internal_parameter_statistics)
165     stat.hist_sigma = zeros(1, q, estimated_stopgen);
166     stat.hist_ps = zeros(N, estimated_stopgen);
167     stat.hist_pc = zeros(N, estimated_stopgen);
168     stat.hist_C = zeros(N, N, estimated_stopgen);
169     stat.hist_B = zeros(N, N, estimated_stopgen);
170     stat.hist_D = zeros(N, N, estimated_stopgen);
171     stat.hist_dps = zeros(q, estimated_stopgen);
172     stat.hist_q = q;
173     stat.hist_N = N;
174   end
175
176   % If restart then load restart log logfile
177   if (run_parameters.do_restart && exist(restart_log_logfile, 'file'))
178     load(sprintf('%s',restart_log_logfile));
179     stat.run_status = 'Incomplete';
180   end
181
```

```
182   % Initialization for speedup
183   xo = zeros(N, q * lambda);
184   zo = zeros(N, q * lambda);
185   fo = zeros(1, q * lambda);
186   zmean = zeros(N,q);
187
188   % Evolution loop
189   while ((stopeval == -1 || evalcount < stopeval) ...
190     && (stopgen == -1 || gencount < stopgen) ...
191     && stat.f_opt(1) > minfitness)
192
193     % Statistics administration
194     gencount = gencount + 1;
195     stat.gencount = gencount;
196     if (internal_parameter_statistics)
197       for i=1:q,
198         stat.hist_sigma(:,i,gencount) = sigma(:,i);
199       end
200         stat.hist_ps(:,gencount) = ps(:,1);
201         stat.hist_pc(:,gencount) = pc(:,1);
202         stat.hist_C(:,:,gencount) = C(:,:,1);
203         stat.hist_B(:,:,gencount) = B(:,:,1);
204         stat.hist_D(:,:,gencount) = D(:,:,1);
205     end
206
207   % Generate a sample set of input parameter disturbances if they are reused
208     if (reuse_disturbances)
209       x_dists = sampling_fct(m, N, -ur_sigma', ur_sigma')';
210     end
211     fo_dists = zeros(1,m);
212     xo_dists = zeros(N,m);
213
214     % Generate and evaluate lambda offspring
215     for j=1:q
216       i = (j - 1) * lambda;
217       for k=1:lambda,
218         % Generate a sample set for every individual
219         if(~reuse_disturbances)
220           x_dists = sampling_fct(m, N, -ur_sigma', ur_sigma')';
221         end
222         zo(:,k+i) = randn(N,1);
223         xo(:,k+i) = xmean(:,j) + sigma(:,j) * (B(:,:,j)
224                                                * D(:,:,j) * zo(:,k+i));
225         xo(:,k+i) = feval(bch_fct, xo(:,k+i), lb, ub);
226
227         for l=1:m,
228           xo_dists(:,l) = xo(:,k+i) + x_dists(:,l);
229           fo_dists(l) = feval(fitnessfct, xo_dists(:,l)');
230         end
231         fo(k+i) = mean(fo_dists);
```

```
232
233         % Statistics administration
234         evalcount = evalcount + 1;
235         stat.evalcount = evalcount;
236         if (history_statistics)
237           stat.hist_x(:,evalcount) = xo(:,k+i);
238           stat.hist_f(evalcount) = fo(k+i);
239         end
240       end
241     end
242
243     % Sort by fitness and compute weighted mean into xmean
244     [~, sortindex] = sort(fo); % M I N I M I Z A T I O N
245     [num_peaks, niches] = dynamic_peak_identification(lambda*q, q,
246                                                       rho, xo, sortindex);
247
248     parxmean = xmean;
249     parsigma = sigma;
250     parC = C;
251     parB = B;
252     parD = D;
253     parps = ps;
254     parpc = pc;
255
256     for i=1:num_peaks,
257       dps_parents = niches(i,:);
258       par = ceil(dps_parents / lambda);
259
260       zmean(:,i) = zo(:,dps_parents) * weights;
261
262       Bn = squeeze(parB(:,:,par));
263       Cn = squeeze(parC(:,:,par));
264       Dn = squeeze(parD(:,:,par));
265
266       xmean(:,i) = feval(bch_fct, parxmean(:,par)
267                          + parsigma(par) * (Bn * Dn * zmean(:,i)), lb, ub);
268
269       % Cumulation: Update evolution paths
270       ps(:,i) = (1 - cs) * parps(:,par) + sqrt(cs * (2 - cs)
271                                                * mueff) * (Bn * zmean(:,i));
272       hsig = norm(ps(:,i)) / sqrt(1 - (1 - cs)^(2 * evalcount / lambda))
273                                                       / chiN < 1.4 + 2/(N + 1);
274       pc(:,i) = (1 - cc) * parpc(:,par) + hsig * sqrt(cc * (2 - cc)
275                                             * mueff) * (Bn * Dn * zmean(:,i));
276
277       % Adapt covariance matrix C
278       Cn = (1 - ccov) * Cn + ccov * (1 / mucov) *...
279       (pc(:,i) * pc(:,i)' + (1-hsig) * cc * (2 - cc) * Cn) +
280                                                        ccov * (1-1/mucov) *...
281       (Bn * Dn * zo(:,dps_parents)) * diag(weights)
```

```
282                                         * (Bn * Dn * zo(:,dps_parents))';
283
284       % Adapt step size sigma
285       sigma(i) = parsigma(par) * exp((cs / damps)
286                                      * (norm(ps(:,i)) / chiN - 1));
287       % Update B and D from C
288       Cn = triu(Cn) + triu(Cn,1)';
289       [Bn,Dn] = eig(Cn);
290       D(:,:,i) = diag(sqrt(diag(Dn)));
291       C(:,:,i) = Cn;
292       B(:,:,i) = Bn;
293     end
294
295     while (num_peaks < q)
296       num_peaks = num_peaks + 1;
297       restart = restart + 1;
298       xmean(:,num_peaks) = lb + (ub - lb) .* rand(N,1);
299       if (isfield(algorithm_parameters, 'sigma_init'))
300         sigma(num_numpeaks) = algorithm_parameters.sigma_init;
301       else
302         sigma(num_peaks) = (norm(ub - lb)) / (3 * q * sqrt(N));
303       end
304       pc(:,num_peaks) = zeros(N,1);
305       ps(:,num_peaks) = zeros(N,1);
306       B(:,:,num_peaks) = eye(N,N);
307       D(:,:,num_peaks) = eye(N,N);
308       C(:,:,num_peaks) = eye(N,N);
309     end
310
311     % Statistics administration
312     for i=1:q,
313       if(niches(i) < 0)
314         %stat.x_opt(:,i) = [inf, inf];
315       else
316         stat.x_opt(:,i) = xo(:,niches(i));
317       end
318     end
319
320     for i=1:q,
321       if (niches(i,1) < 0)
322         stat.f_opt(i) = Inf;
323       else
324         stat.f_opt(i) = fo(sortindex(niches(i,1)));
325       end
326     end
327
328     if (history_statistics)
329       stat.evalvsgen(stat.gencount) = evalcount;
330       stat.hist_x_opt(:,:,gencount) = stat.x_opt;
331       stat.hist_f_opt(:,gencount) = stat.f_opt;
```

```
332       stat.hist_xmean(:,gencount) = xmean(:,1);
333       stat.hist_dps(:,gencount) = niches(:,1);
334     end
335
336     % Store log for restart
337     if (restart_log_intermediate)
338       save(sprintf('%s',restart_log_logfile), 'xmean', 'sigma', 'pc',
339                    'ps', 'B', 'D', 'C', 'evalcount', 'gencount', 'stat');
340     end
341
342     % Report statistics
343     if (report_intermediate)
344       report_fct(stat, problem_specification,
345                                       algorithm_parameters, run_parameters)
346     end
347   end
348
349   % Complete statistics struct
350   if (history_statistics)
351     stat.evalvsgen = stat.evalvsgen(1:gencount);
352     stat.hist_x = stat.hist_x(:,1:evalcount);
353     stat.hist_f = stat.hist_f(:,1:evalcount);
354     stat.hist_x_opt = stat.hist_x_opt(:,:,1:gencount);
355     stat.hist_f_opt = stat.hist_f_opt(:,1:gencount);
356     stat.hist_xmean = stat.hist_xmean(:,1:gencount);
357   end
358   if (internal_parameter_statistics)
359     stat.hist_sigma = stat.hist_sigma(:,:,1:gencount);
360     stat.hist_ps = stat.hist_ps(:,1:gencount,:);
361     stat.hist_pc = stat.hist_pc(:,1:gencount,:);
362     stat.hist_C = stat.hist_C(:,:,1:gencount,:);
363     stat.hist_B = stat.hist_B(:,:,1:gencount,:);
364     stat.hist_D = stat.hist_D(:,:,1:gencount,:);
365   end
366   stat.run_status = 'Complete';
367
368   % Store log
369   if (restart_log_intermediate || restart_log_after_termination)
370     save(sprintf('%s',restart_log_logfile), 'xmean', 'sigma', 'pc', 'ps',
371                             'B', 'D', 'C', 'evalcount', 'gencount', 'stat');
372   end
373
374   % Plot statistics
375   if (report_after_termination)
376     report_fct(stat, problem_specification, algorithm_parameters,
377                                                           run_parameters)
378   end
379   stat.hist_restart = restart;
380 end
```