# Real-Valued Optimization Using Evolution Strategies

*Date: 6-12-2013*

*Authors:*

*Brian Weeteling* [*brian\_weeteling@hotmail.com*](mailto:brian_weeteling@hotmail.com) *1276417*

*Ricardo Blikman* [*ricardo.blikman@yahoo.com*](mailto:ricardo.blikman@yahoo.com) *1184164*

# *Universiteit Leiden*

# 

# Introduction

In this research assignment we must implement an evolutionary strategy in MATLAB and benchmark it against five black-boxoptimization problems that are given to us. The algorithm should also be able to operate on arbitrary values of µ and . Each black-box problem consists of a function with real-valued input parameters that is to be minimized:

Subject to for all in

with = 30 (i.e., a 30-dimensional search space).

# Problem Description

Our research question is: How well does our implementation of an evolutionary algorithm perform based on five black-box benchmarks for real-valued optimization?

# Method

We will implement both 1+1 and evolution strategies in MATLAB and benchmark both implementations and compare them based on a budget of 10,000 function evaluations per run averaged over 20 runs over each of the five black-box problems to answer our research question. We will also set different values for both and .

# Implementation

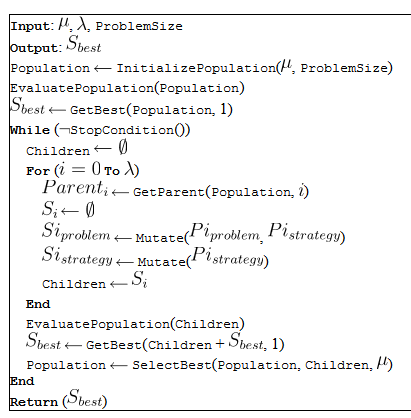
We have implemented a Genetic Algorithm in MATLAB with the following signature:

function [aopt, fopt] = weeteling\_blikman\_es(cnf\_file, eval\_budget)

The following parameters must be supplied:

cnf\_file (Benchmark), eval\_budget (evauluations to be made)

## Pseudo code Implementation



Input: Objective function, dimensions, lower bound, upper bound.

Output: population, Best, statistics.

Set

Set

Population 🡨 (dimensions, , lower bound, upper bound)

Initialize sigma (lower bound + (upperboud - lowerbound) \* random()) /6

EvaluatePopulation(Population)

Best = Getbest(Population)

While evaluations < stopval

‘

For I = 1 to

## Variables

Our algorithm starts with initializing the following variables:

= 3 (Parents)

= 12 (Offspring)

And in another run:

= 15 (Parents)

= 100 (Offspring)

## Initialization

We start with initializing a random vector of bits based on the size provided by the parameter: cnf\_file (benchmark file) and initialize the population of the vector for mutations. We Generate random chromosome, decode to phenotype, and evaluate these using the feval function in MATLAB.

## Evolution

For evolutions we generate new population. We start with evaluating P1 (parent one) using the feval function based on the tournament parameters. Then we evaluate the chance of to occur by using a random number. If occurs then we create a second parent P2 using the same function and parameters as P1 and create a child based on P1|P2 and complete our crossover. If does not occur we copy the parent into our new generation.

mutationString = rand(n,1) < pm;

mutatedIndividual = xor(Pnew(:,i),mutationString);

Pnew(:,i) = mutatedIndividual;

% Decode and evaluate

for i = 1:mu

g(:,i) = feval(decodefct, P(:,i));

f(i) = feval(fitnessfct, g(:,i));

end

%save best from last generation

[~, worstindex] = min(f);

P(:,worstindex) = aopt;

f(worstindex) = fopt;

% Statistics administration

[fopt, optindex] = max(f);

aopt = P(:,optindex);

for i = 1:mu

evalcount = evalcount + 1;

% histf(evalcount) = fopt;

end

after each evaluation we set the permutation rate (pm) to: -.985+exp((0.5+(1-currentEvaluation)/(functionEvaluations \*2.5))^4).

# Experiments

We benchmarked our implementations based on a budget of 10,000 function evaluations per run averaged over 20 runs over each of the five black-box problems given to us.

We benchmarked our implementation against 10 benchmarks problems for a MAX-3SAT problem with a maximum of 10,000 function evaluations per run averaged over 20 runs. The following benchmarks where used:

|  |  |
| --- | --- |
| Benchmark 1 | bbf1.m |
| Benchmark 2 | bbf2.m |
| Benchmark 3 | bbf3.m |
| Benchmark 4 | bbf4.m |
| Benchmark 5 | bbf5.m |

We have set two different values for both and . Our initial run with the was:

= 3 and =12

In our second run for the strategy we set the values to:

= 15 and = 100.

Our experiment yielded the following results:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BBF1 |  |  |  |  |  |  |  |  |
| Function | At 1000 eval | | | | At 10000 eval | | | |
|  | min | max | mean | std | min | max | mean | std |
| 3,12 | 55,1034 | 2569,994 | 447,013 | 707,4673 | 34,5575 | 34,5575 | 34,5575 | 7,11E-15 |
| 15,100 | 1646,152 | 18386,47 | 6640,775 | 4102,271 | 34,5576 | 34,5786 | 34,5608 | 0,005647 |
| 1+1 | 34,8757 | 77,1629 | 43,0728 | 11,0562 | 34,5575 | 34,5575 | 34,5575 | 1,41E-14 |
|  |  |  |  |  |  |  |  |  |
| BBF2 |  |  |  |  |  |  |  |  |
|  | At 1000 eval | | | | At 10000 eval | | | |
| Function | min | max | mean | std | min | max | mean | std |
| 3,12 | 1418,459 | 16474,18 | 6727,592 | 4623,319 | 220,7705 | 4753,914 | 868,8151 | 1042,451 |
| 15,100 | 82291,32 | 302314,3 | 181840,9 | 54791,05 | 316,3063 | 580,4581 | 428,2092 | 80,0053 |
| 1+1 | 447,1914 | 3094,793 | 917,385 | 550,1389 | 297,3813 | 1004,764 | 585,5591 | 209,0616 |
|  |  |  |  |  |  |  |  |  |
| BBF3 |  |  |  |  |  |  |  |  |
| Function | At 1000 eval | | | | At 10000 eval | | | |
|  | min | max | mean | std | min | max | mean | std |
| 3,12 | 4728,607 | 637261,6 | 123439,7 | 192594,4 | 40,8964 | 52,5223 | 42,8068 | 3,095 |
| 15,100 | 686003,4 | 14680687 | 4916420 | 4260836 | 65,1672 | 2163,664 | 763,8674 | 678,6048 |
| 1+1 | 89,9708 | 7931,828 | 1108,556 | 1811,72 | 40,8407 | 311,5456 | 120,6188 | 92,5931 |
|  |  |  |  |  |  |  |  |  |
| BBF4 |  |  |  |  |  |  |  |  |
| Function | At 1000 eval | | | | At 10000 eval | | | |
|  | min | max | mean | std | min | max | mean | std |
| 3,12 | 45,0284 | 46,5566 | 45,3208 | 0,36001 | 43,9823 | 44,0193 | 43,9964 | 0,012371 |
| 15,100 | 46,6346 | 55,7886 | 49,9422 | 2,7921 | 44,1291 | 44,5433 | 44,3026 | 0,11315 |
| 1+1 | 44,1427 | 45,0376 | 44,7143 | 0,27313 | 43,9823 | 44,0289 | 43,9926 | 0,012536 |
|  |  |  |  |  |  |  |  |  |
| BBF5 |  |  |  |  |  |  |  |  |
| Function | At 1000 eval | | | | At 10000 eval | | | |
|  | min | max | mean | std | min | max | mean | std |
| 3,12 | 68,3066 | 68,7698 | 68,6043 | 0,10064 | 68,2853 | 68,7565 | 68,6155 | 0,10908 |
| 15,100 | 67,4522 | 68,5608 | 68,334 | 0,28681 | 50,941 | 68,5782 | 64,7729 | 6,2686 |
| 1+1 | 67,1249 | 67,3662 | 67,2137 | 0,074606 | 67,1236 | 67,1239 | 67,1238 | 7,94E-05 |

Table 1: Final solution(s) quality after 10,000 function evaluations, averaged over 20 runs

Benchmark results:

# C:\Studie\Computer Science\EA\Assignment2\Final matlab\Result images\bbf1.png

Figure 1: bbf1 benchmark.

# C:\Studie\Computer Science\EA\Assignment2\Final matlab\Result images\bbf2.png

Figure 2: bbf2 benchmark.

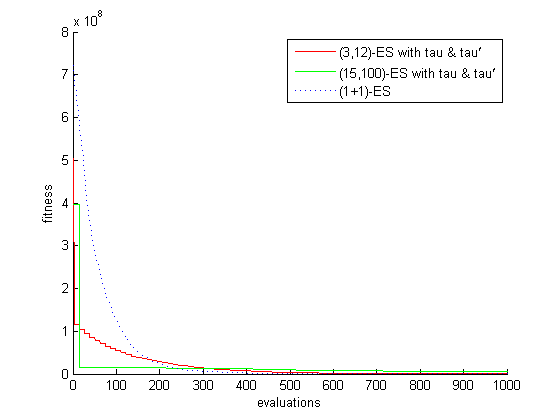


Figure 3: bbf1 benchmark.

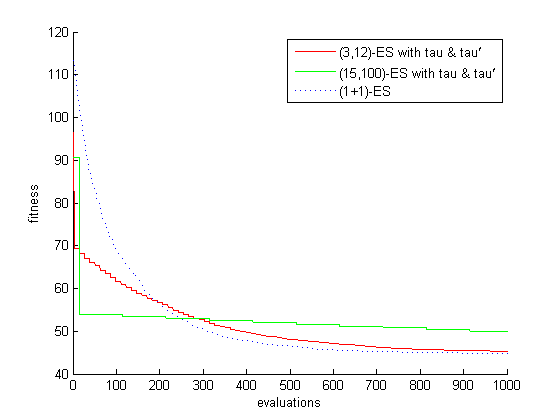


Figure 4: bbf4 benchmark.

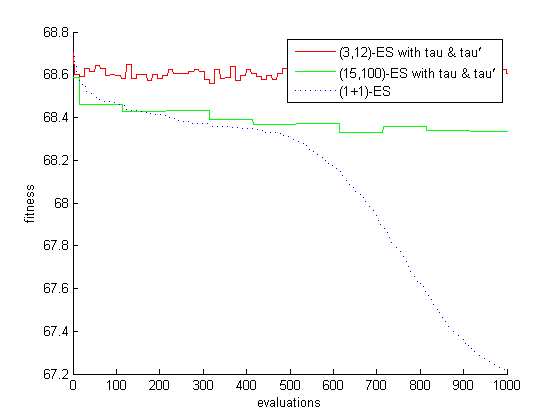


Figure 5: bbf5 benchmark.

# Discussion and Conclusion

The goal of this research assignment was to measure how well does our implementation of an evolutionary algorithm perform based on five black-box benchmarks for real-valued optimization?

Based on the outcome of this experiment we can conclude the following:

The black-box functions are probably functions picked from different problem classes, there is not one algorithm that can be picked as clear winner on all functions. The 1+1 ES has the best overall results though, with 10000 function evaluation but especially considering the results at 1000 evaluations. Since we are limited to strategies we would pick the 3,12 ES as winner, with the 15,100 as a close second. The 3,12 ES seems to be especially fit for black-box function 3, as it is a clear winner on all measurements.

Another interesting observation is that the 15,100 ES converges very fast on the first 20-30 function evaluations. So when the evaluation budget is very low, we could definitely propose the 15,100 ES as best solution.