

# The Low-Autocorrelation Problem

Antanas Kaziliūnas  
Universiteit Leiden  
Niels Bohrweg 1  
2333 CA Leiden  
The Netherlands

akaziliu@liacs.nl

## ABSTRACT

This paper describes the design and results of genetic algorithm that computes the low-autocorrelation problem.

## General Terms

Algorithms.

## Keywords

Genetic algorithms, search, low-auto correlation, MATLAB.

## 1. INTRODUCTION

The low-autocorrelation problem is a subject to great research in a few industries, such as communications and electrical engineering. In this paper a short description of the problem is provided together with a genetic algorithm that attempts to calculate the possible solutions. The best found results are reported.

## 2. THE LOW-AUTOCORRELATION PROBLEM

**Feasible Solutions:** Binary Sequences  $\vec{y} \in \{-1, +1\}^n$

**Objective Function:**

$$f(\vec{y}) = \frac{n^2}{2} \cdot E(\vec{y}) \rightarrow \text{maximization} \quad (1)$$

s.t.

$$E(\vec{y}) = \sum_{k=1}^{n-1} \left( \sum_{i=1}^{n-1} y_i \cdot y_{i+k} \right)^2 \quad (2)$$

i.e., the low autocorrelated binary string problem is defined as finding the string with the maximal objective function.

## 3. ALGORITHM

To solve the problem an algorithm following the classical evolutionary algorithm [1] containing evolutionary loop was designed. The basic algorithm is described in Algorithm 1.

### Algorithm 1. Evolutionary Algorithm

```
t := 0;  
initialize P(t);  
evaluate P(t);  
  
while not terminate do  
    P'(t) := select-mates(P(t));  
    P''(t) := crossover(P'(t), p_c);  
    P'''(t) := mutate(P''(t), p_m);  
    evaluate(P'''(t));  
    P(t + 1) := P'''(t);  
    t := t + 1;
```

od

The full algorithm was implemented in MATLAB. Program code can be found in Appendix A.

Generally, a new generation of individuals is formed using selection. With the proposed operators the new population is transformed. The decrease in fitness values is allowed.

### 3.1 Selection

For the next generation offspring selection two methods were implemented.

#### 3.1.1 Roulette Selection

Using this selection method, a certain number of best individuals compete to be selected in to the proceeding generation. One individual is selected with a probability which is proportionate to its fitness value. The procedure is repeated until the predefined population size is reached.

#### 3.1.2 Tournament Selection

For this selection method two individuals are randomly picked from the population. The individual with better fitness value is to be included in the next generation population. This procedure is repeated until the predefined population size is reached.

### 3.2 Crossover

A one point crossover is the crossover operator in the algorithm discussed. Two proceeding strings are picked and crossed at a randomly drawn point. No further randomization in picking the individuals is used due to the random nature of the position that each individual holds.

### 3.3 Mutation

Each bit of a string is flipped with a defined probability. Depending on the selection method and other parameters, the mutation rate varies accordingly.

## 4. RESULTS

Various code runs were performed, using different parameters and testing two different selection methods. In Table 1 the best found and so far known values can be found.

Though no thorough research on which selection method outperforms the other was conducted and looking at Table 1 no conclusions can be drawn, the convergence graphs showed tournament selection to be more promising, as most of the cases the roulette selection (as implemented) would get stuck in a local maxima too soon.

**Table 1. Best found and known values**

n	Best Found f	Best Known f	Selection Method
20	7.6923	7.6923	Both methods
50	5.9809	8.1699	Roulette
100	5.1125	8.6505	Tournament
199	4.4366	7.5835	Tournament

## 7. Appendix A (algorithm code in MATLAB)

```
% N - string length
% l - permutation rate
% T - times of run
% P - number of parents
% O - number of offsprings
% sel - which selection method (roulette or tournament)

function best = autoGA(N, l, T, P, O, sel)

bestC = [0]; % best known values
bestSeq = []; % best sequence

% initialize
Y = 2*(rand(N,O) > 0.5)-1;

r = 1;

% evaluation
F = merit(Y);

while (r <= T)

    if sel == 1
        % roulette selection
        selectedY = rouletteSelection(Y, F, P, O);
    else
        % tournament selection
        selectedY = tournamentSelection(Y, F, O);
    end
end
```

200	5.0607	7.4738	Tournament
201	4.8935	7.5263	Tournament
202	4.6836	7.3787	Roulette
203	4.3534	7.5613	Roulette
219	4.6791	7.2122	Roulette
220	4.3888	7.0145	Tournament
221	4.5646	7.2207	Roulette
222	4.7216	7.0426	Tournament

## 5. CONCLUSIONS

The designed algorithm proved to have potential to tackle the proposed problem. However, the results are far from the best. For further research, the exploration of best selection operator and its possible variations could be useful. Also, some meta evolutionary algorithm for finding best parameters might give good results.

## 6. REFERENCES

- [1] T. Bäck, D. B. Fogel, Z. Michalewicz "Evolutionary Computation 1. Basic Algorithms and Operators.", 2000, Institute of Physics Publishing, Bristol, UK, pp. 59-63.

```

%one point crossover
selectedY = crossover(N, selectedY, 0);

%mutation
Y = selectedY .* (2*(rand(N,0) > 1-1/l)-1);

%evaluation
F = merit(Y);

[C, I] = max(F);

if max(bestC) < C bestSeq = Y(:,I); end

bestC(r) = C;

r = r + 1;

end

%bestC
bestSeq
best = max(bestC)

function selected = rouletteSelection(population, fitnesses, parents, offsprings)

    [sortedF, IX] = sort(fitnesses, 'descend');
    sortedF = sortedF(1:parents);
    IX = IX(1:parents);

    roulette = cumsum(sortedF) / sum(sortedF);

    rouletteValues = rand(1, offsprings);
    %rouletteValues = betarnd(1, 1, 1, 0);

    for (i=1:1:offsprings)
        selected(:,i) = population(:,IX(find(roulette >= rouletteValues(i), 1, 'first')));
    end;

    %elite selection
    %selectedY(:,0) = bestSeq;

function selected = tournamentSelection(population, fitnesses, offsprings)
tournament = unidrnd(offsprings, 2, offsprings);

for (i=1:1:offsprings)
    if fitnesses(tournament(1, i)) > fitnesses(tournament(2, i))
        selected(:,i) = population(:, tournament(1,i));
    else
        selected(:,i) = population(:, tournament(2,i));
    end
end

function crossed = crossover(N, population, offsprings)

    for i=1:1:offsprings/2
        cPoint = unidrnd(N-1, 1, 1) + 1;

        A = population(:,i*2-1);
        B = population(:,i*2);

```



