

The Low-Autocorrelation Problem of Binary Strings

[A Genetic Algorithm Approach]

Alexander Aleman
Leiden University
alexander.aleman@gmail.com

ABSTRACT

In this paper the author tries to solve the low-autocorrelation problem using genetic algorithms. Two algorithms will be presented to the reader, as well as a comparison of the two algorithms with each other and a comparison with the best known results. The algorithms have been implemented in MATLAB.

1. INTRODUCTION

Given a binary string of dimension n , $\vec{y} \in \{-1, +1\}^n$, the merit factor is defined as:

$$f(\vec{y}) = \frac{n^2}{2 \cdot E(\vec{y})}$$

where

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

The low-autocorrelated binary string problem is defined as finding the string with the maximal merit factor.

2. THE ALGORITHMS

In the following two sections two different algorithms will be discussed. The first algorithm is a more or less standard genetic algorithm with (μ, λ) selection. The second algorithm is the algorithm as indicated in the Evolutionary Algorithms course, by Prof. dr. Thomas Back, material <http://www.liacs.nl/baeck/EA/slides/ea-5.pdf>, page 5.92. The implementation of both algorithms can be found in the zip file that accompanies this report and are called `lowauto1.m` for algorithm 1 and `lowauto2.m` for algorithm 2. Both algorithms use the `merit.m` evaluation function which was written by Ofer M. Shir and `myrandint.m` as a random integer generator written by someone helpful from the MATLAB forum, who forgot to state his name. A `readme.txt` file is there to explain how to use the algorithms.

2.1 Algorithm 1

The first algorithm uses a (μ, λ) selection, where μ parent binary strings create λ offspring binary strings. The algorithm works as follows:

λ parent couples are selected from the population of μ parents. The chance for a parent to be selected is proportional to their fitness in the population. Parents with a higher fitness value are more likely to be selected for reproduction than parents with lower fitness values. Once the parents are selected, each couple produces one child. The child has a high probability to be a recombination of both parents and a small probability to only be a copy of the first parent. The recombination in this algorithm is the so called 1-point crossover. This means that a random point in the binary sequence is selected. Before this point the child gets the binary sequence up to that point from parent 1 and after that he gets the sequence from parent 2. After the reproduction cycle the parents die and the offspring are mutated with a small probability $p = \frac{1}{n}$. From this mutated offspring the best μ children, according to their fitness value, are selected to be the new parent population. Now the parent selection can start again and this will continue until the predefined number of iterations is reached. At all times the best binary sequence is stored.

2.2 Algorithm 2

The second algorithm works slightly different. In this algorithm we have N parents in the population. Each parent creates k offspring in the following way:

1. Per individual \vec{x} : Create k offspring by mutating \vec{x} :
 - (a) Repeat for $i = 1, \dots, k$:
 - Create q individuals by random 1-bit-changes of \vec{x}_i .
 - Memorize the best i -bit-mutant \vec{x}_i .
 - (b) Return $\vec{x}_1, \dots, \vec{x}_k$ (best 1-bit-mutant, ..., best k -bit-mutant).
2. Select the best N out of $(N + k \cdot N)$ parents and offspring.
3. Repeat until stop criterion is satisfied.

3. PERFORMANCE COMPARISON

In this section the two algorithms will be compared for several different parameters. The results can be seen in the

Table 1: Performance

n	A1	A2	Best known
20	5.8824	7.6923	7.6923
50	6.2189	5.9809	8.1699
100	4.5704	4.9116	8.6505
199	4.3739	4.4129	7.5835
200	4.3290	4.5788	7.4738
201	4.3648	4.3200	7.5263
202	4.1425	4.4849	7.3787
203	4.1333	4.9854	7.5613
219	4.1367	4.5721	7.2122
220	4.0837	4.2561	7.0145
221	4.2090	4.4940	7.2207
222	4.0036	4.6017	7.0426

table above. Each line in the table represents the maximum value the algorithms obtained with the given amount of evaluation time. For Algorithm 1 the standard population size μ is equal to 15 in this comparison. The amount of offspring λ is equal to 100. For Algorithm 2 the standard parameters are $N = 15$, $q = 25$ and $k = 10$. The amount of iterations for the algorithms is set in such a way that, both algorithms get nearly the same amount of evaluation time. Both algorithms ran 25 times per value of n and the maximum value is placed in the table. The last column in the table represents the best known values of the merit function for the given n .

4. CONCLUSIONS

As became clear from the performance comparison, the second algorithm performs slightly better than the first one. Also clear is that much progress is still to be made to increase the chance of hitting the optimal value. At this moment in time both algorithms are still too likely getting stuck in local maxima. And actually never reached the best known value so far except for $n = 20$.