

The Low-Autocorrelation Problem

Renuka Autar
(0318760)
Liacs, Leiden University
Leiden, the Netherlands
rautar@liacs.nl

Timo de Vries
(0308161)
Liacs, Leiden University
Leiden, the Netherlands
vriestimo@gmail.com

ABSTRACT

In this paper we designed an Evolutionary algorithm for the *low-autocorrelation* problem of binary sequences. We tried to find the optima of the *low-autocorrelation* problem based on experiments with string lengths of the binary strings for which the best known optimals are known.

We describe our designed EA and the methodology of how to find the optima of this problem. The results are also added with an analysis of the optima for the the string legths.

Keywords

low-autocorrelation binary sequence, evolutionary algorithms

1. INTRODUCTION

The description of the *low-autocorrelation* problem of binary sequences is as follows:

Feasible Solutions: Binary Sequences

$$\vec{Y} \in \{-1, 1\} \quad (1)$$

Objective Function:

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

Autocorrelation is a mathematical tool used frequently in signal processing for analyzing functions or series of values, such as time domain signals [1]. The goal of optimization for the *low-autocorrelation* problem is to find a sequence that maximizes F , or equivalently, minimizes E . The *merit* function is our objective function, that we used to optimize our Evolutionary algorithm. The list of best known values so far can be found at [2].

2. EVOLUTIONARY ALGORITHM

Evolutionary Algorithms are based on the *evolution loop* [3]. The general outline of an Evolutionary algorithm is as follows:

```
t := 0;
initialize P(t);
evaluate P(t);
while not terminate do
    P'(t) := select-mates(P(t));
    P''(t) := variation(P'(t));
    evaluate(P''(t));
    P(t + 1) := select ((P''(t)) ∪ (P'(t)));
    t := t + 1;
od
```

Our evolutionary algorithm starts with initializing the first generation and then evaluating them with the *merit* function, *see section 2.1*. After the evaluation, the proportional fitness values are calculated and a selection of the individuals is made for mating, *see section 2.2*. The next step in our algorithm is recombination and mutation for generating the next generation, *see section 2.3*.

2.1 Coding and Fitness evaluation

Our population is encoded with a matrix consisting of 1's and -1's. In the beginning this matrix is randomly initialized with dimensions stringlength × nr. of offspring individuals.

We evaluate all individuals using the *merit* function. The best fitness value is saved together with the corresponding bitstring. Finally, we subtract the minimal fitness value from all the individual fitness values and square them. This subtraction is to eliminate the worst individual(s) and squaring them is to increase the difference between the fitness values.

2.2 Selection

First we calculate the proportional fitness values, after which we create a list with the cumulative values. The cumulative list starts with a low value, almost zero and ends with 1. This could be seen as the numbers on a roulette-wheel. The difference between two values is the proportional fitness of an individual. So in theory, every individual has a chance to get selected for mating by the roulette-wheel selection procedure. We tried different selection methods such as tournament selection as well, but roulette-wheel selection seemed to work best in our case.

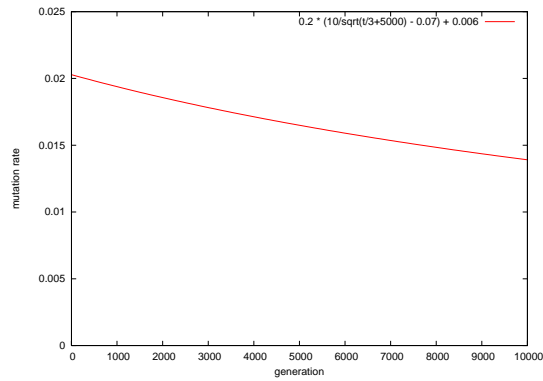


Figure 1: The decreasing mutation rate

2.3 Variation

The first step in the variation process is recombination. For each child individual to be created, there is a chance for it to become an exact copy, i.e. a clone, of a parent individual. The crossover rate determines the chance for crossover to take place. When this happens, two parents are randomly selected and one-point-crossover is applied to form the offspring individual. We have decided to use a time-dependent crossover rate, constantly decreasing over time, starting at 0.9 and ending at 0.1. The next and final step in the variation process is mutation. All parent individuals have the same chance of getting mutated. The mutation rate decreases over time, see Figure 1.

The ratio of the 1's and the -1's in the random matrix is dependent on the mutation rate. Finally, by multiplying the individuals with the random matrix, they are mutated.

3. EXPERIMENTAL RESULTS

We obtained the following results for the different string lengths.

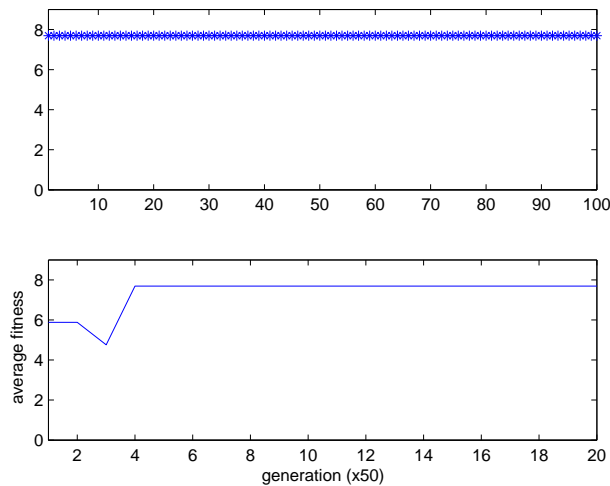


Figure 2: Best fitness for population $n = 20$

For $n = 20$, the results are shown in Figure 2.
For $n = 50$, the results are shown in Figure 3.

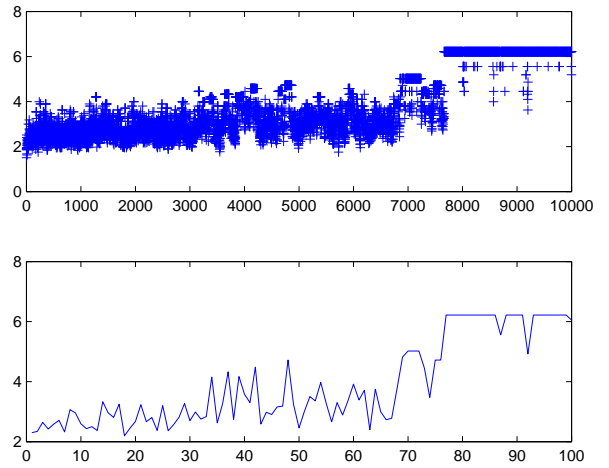


Figure 3: Best fitness for population $n = 50$

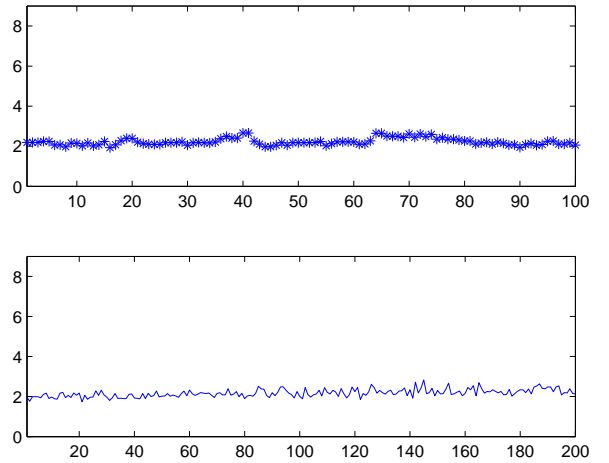


Figure 4: Best fitness for population $n = 100$

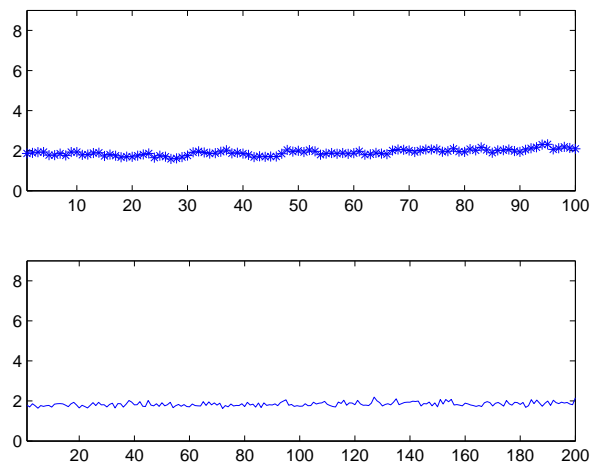


Figure 5: Best fitness for population $n = 199$

For $n = 100$, the results are shown in Figure 4.
 For $n = 199$, the results are shown in Figure 5.
 For $n = 200$, the results are shown in Figure 6.
 For $n = 220$, the results are shown in Figure 7.

The mutation rate is very important in the algorithm, the maximal fitness value is mostly effected by it.

4. REFERENCES

- [1] Autocorrelation, slide 5
<http://en.wikipedia.org/wiki/Autocorrelation>
- [2] Practical Assignment Evolutionary Algorithms
<http://www.liacs.nl/~baeck/EA/pa/pa.pdf>
- [3] Evolution loop, slide 4.7
<http://www.liacs.nl/~baeck/EA/slides/ea-4.pdf>

APPENDIX

Table 1: Experiment results

n	Best f in our algorithm
20	7.69
50	6.22
100	3.79
199	2.57
200	2.71
220	2.50

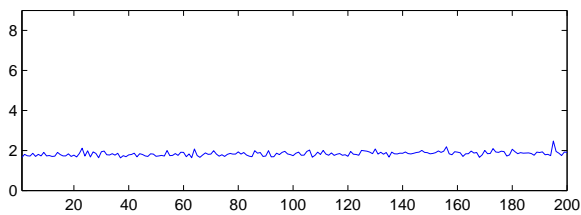
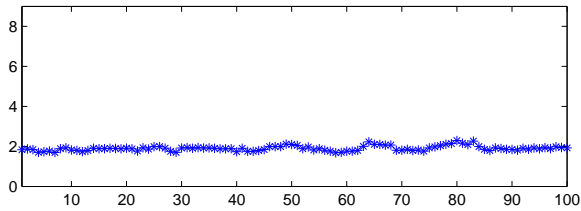


Figure 6: Best fitness for population $n = 200$

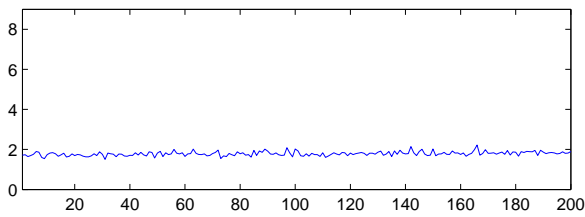
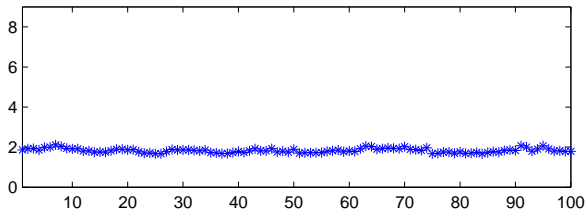


Figure 7: Best fitness for population $n = 220$