The Low-Autocorrelation Problem

A genetic algorithm approach

Eyal Halm
Leiden University, LIACS
Niels Bohrweg 1
2333 CA, Leiden
The Netherlands
ehalm@liacs.nl

Michiel Helvensteijn
Leiden University, LIACS
Niels Bohrweg 1
2333 CA, Leiden
The Netherlands
mhelvens@liacs.nl

ABSTRACT
The Low-Autocorrelation Problem of binary sequences is subject to actual research and is of big interest for industrial applications, e.g. communications and electrical engineering. Autocorrelation is useful for finding repeating patterns in a signal, such as determining the presence of a periodic signal which has been buried under noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies. We will attempt to create a genetic algorithm to find binary sequences that represent the best possible solution to this problem. It is a non-trivial problem, because there is no smooth evolutionary landscape to work with. In this paper, we will investigate several techniques that may help.

General Terms
Algorithms

Keywords
Genetic algorithm, Evolutionary algorithm, Low-Autocorrelation Problem

1. INTRODUCTION
The Low-Autocorrelation Problem is indeed challenging. The search space is extremely large and the solutions lie on a very chaotic landscape.[1] In this paper we will investigate different approaches to try to solve this problem.

Section 2 describes the Low-Autocorrelation Problem mathematically. Section 3 describes the algorithm, as well as the different variations we tried. It sometimes uses metacode to clarify. In Section 4, we present the results of our experiments with these variations. In Section 5 we discuss these results and offer posible explanations for their relative performances. Section 6 concludes the paper with some closing remarks.

[1] The landscape might be by definition chaotic, since the goal of the problem is to find a sequence in which the elements have the smallest possible correlation with each other.

2. LOW-AUTOCORRELATION PROBLEM
Let $\overrightarrow{y}$ be a binary sequence $(y_1, y_2, \ldots, y_n)$ where
$\forall\ 1 \leq i \leq n : y_i \in \{-1, +1\}$.

The energy function of a binary sequence $\overrightarrow{y}$ (of length n) is defined as follows:

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

The Low-Autocorrelation Problem is, given a string length n, to find a binary string $\overrightarrow{y}$ with the lowest possible $E(\overrightarrow{y})$. So:

$$E(\overrightarrow{y}) \quad \longrightarrow \quad min$$

To make it easier to compare results of different string-length and to allow techniques like roulette-wheel selection, the following maximization problem can also be used:

$$f(\overrightarrow{y}) = \frac{n^2}{2 \times E(\overrightarrow{y})} \quad \longrightarrow \quad max$$

The search-space is an n-dimensional space with 2^n discrete solutions. Enumeration of the solutions is infeasable for strings exceeding a certain length.

The complexity of $E(\overrightarrow{y})$ and $f(\overrightarrow{y})$ is $O(n^2)$.

We have tried to analyze the Energy function in an attempt to better understand the evolutionary landscape. We have visualized it in Figure 2. Every line-segment in the figure corresponds to a $\times$ in $E(\overrightarrow{y})$. Every group of line-segments represents an iteration of the outer summation. It is clear that a mutation on any bit has influence on exactly $n-1$ iterations of the inner summation. We were hoping to find some asymmetry to exploit in the mutation function, but

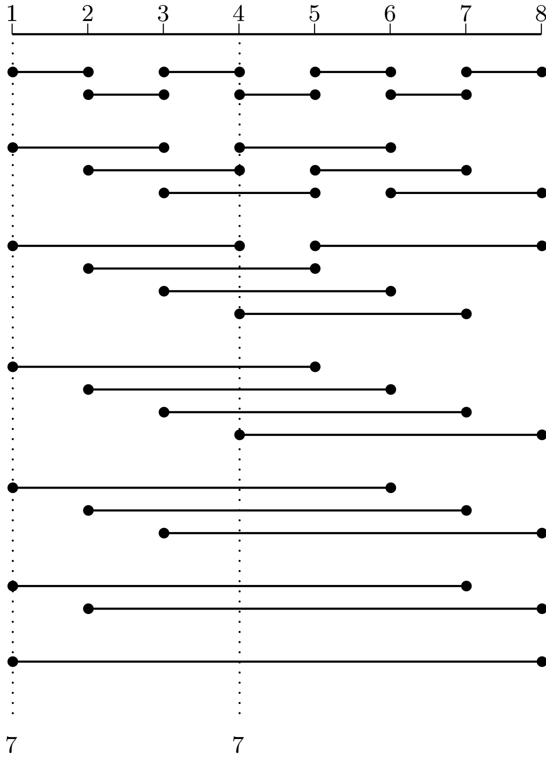

Figure 1: Iterations of the objective function with string-length 8. Every bit is referenced $(n-1)$ times.

we could find none. We suspect that a mutation on any of the bits in the sequence has the same influence on $E(\overrightarrow{y})$ on average.

3. THE GENENETIC ALGORITHM

Here we will describe the genetic algorithm and the techniques we tried. In Section 3.1 we will outline the structure of the main loop and the baseline variation- and selection operators. In Section 3.2 we introduce new selection and reduction operators, together with a slight change to the objective function representation. Section 3.3 introduces a technique we tried to avoid local optima. Section 3.4 explores the possibility of smoothing out the evolutionary landscape using a different bitstring encoding to perform standard mutation on. Lastly, Section 3.5 explains another attempt to do this, but this time by making the mutation operator try various possibilities before committing to a mutation.

From now on, the following notations shall be used.

- $\overrightarrow{y}$ will represent a binary sequence $\in \{-1, +1\}^n$.
- n will represent its length.
- $f(\overrightarrow{y})$ is the objective function value of $\overrightarrow{y}$.
- $(\preceq, \{-1, +1\}^n)$ is the total preorder such that: $\overrightarrow{x} \preceq \overrightarrow{y} \iff f(\overrightarrow{x}) \leq f(\overrightarrow{y})$
- K is the parent population size in the algorithm (assumed globally available).

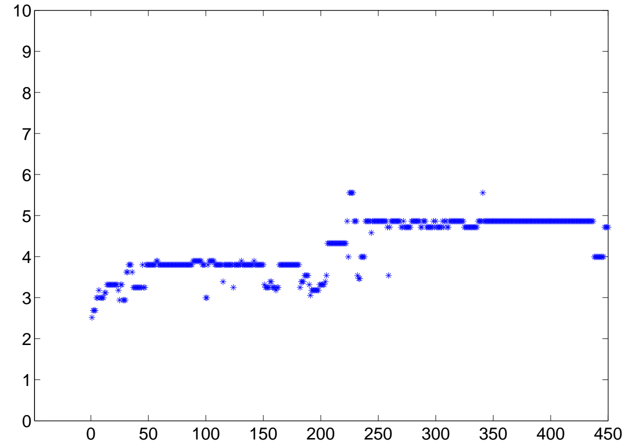

Figure 2: A run of the baseline algorithm. The x-axis shows the generation and the y-axis shows the best fitness value of the population of that generation. The best fitness of this run has been found twice, but promptly forgotten, which can sometimes happen in an evolutionary algorithm.

- L is the offspring population size in the algorithm (assumed globally available).

3.1 The baseline algorithm

The main loop of the algorithm is as follows:

randomly initialize population P
evaluate P
$Best_{\text{all}} \leftarrow \max_{\preceq}(P_4)$
do until *termination*
  $P_1 \leftarrow$ selectmatingpartners(P)
  $P_2 \leftarrow$ crossover(P_1)
  $P_3 \leftarrow$ mutate(P_2)
  evaluate P_3
  $P_4 \leftarrow$ reduce(P_3)
  $Best_{\text{all}} \leftarrow \max_{\preceq}(P_4 \bigcup \{Best_{\text{all}}\})$
  $P \leftarrow P_4$
enddo
return $Best_{\text{all}}$

In Figure 2 you can see a sample run of this algorithm.

We will now explain how each of the operators works:

3.1.1 *selectmatingpartners*

This operator uses the roulette wheel method to redistribute the population space in such a way that individuals with a higher objective function value have a better chance of being chosen as a potential parent. It is possible (and likely) that some individuals will appear more than once in the resulting population and that some other individuals will be implicitly eliminated by this process.

This code roughly explains how this works:

function `selectmatingpartner`(Old)
  $Sum \leftarrow \sum_{x=1}^{K} f(y_x)$
  for $i = 1$ to K
    $TSum \leftarrow \text{rand}(0, Sum)$
    Choose $I : \sum_{x=1}^{I} f(y_x) \leq TSum \leq \sum_{x=1}^{I+1} f(y_x)$
    $New(i) \leftarrow Old(I)$
  endfor
  return New

The population size does not change in this phase. Only the distribution has changed.

3.1.2 crossover

This is the phase where the population size increases. The new population consists of copies and combinations of the initial (parent) population. More specifically, we use one-point crossover with a crossover rate of P_c.

The meta-code is as follows:

function `crossover`(Old)
  for $i = 1$ to L
    $I_1 = \text{rand}(1, K)$
    $I_2 = \text{rand}(1, K)$
    with a chance of P_c
      $loc = rand(1, n-1)$
      $New(i, 1 \ldots loc) \leftarrow Old(I_1, 1 \ldots loc)$
      $New(i, loc+1 \ldots n) \leftarrow Old(I_2, loc+1 \ldots n)$
    else
      $New(i) \leftarrow Old(I_1)$
    endchance
  endfor
  return New

So each individual in the resulting population has chance of P_c to consist of two parents combined with one-(random)-point crossover and a chance of $1 - P_c$ to be a copy of a random parent.

3.1.3 mutate

This relatively simple baseline operator flips ($1 \mapsto -1$, $-1 \mapsto 1$) every bit in the binary sequence with a chance of P_m. This is done for each individual. The size of the population does not change as a result.

3.1.4 reduce

The `reduce` method uses the roulette-wheel method, like the `selectmatingpartners` method, to select the individuals that survive this selection-process. The difference is that in this case, each individual can only be chosen once. So when an individual is chosen, it is eliminated from the set and the sum of evaluations so it can not be chosen again.

3.2 Tournament selection

This is one of the changes we tried on the baseline algorithm. Instead of using roulette-wheel selection, we use tournament selection. Both in `selectmatingpartners` and `reduce`. Basically, we randomly choose X candidates and the one with the highest objective function value is chosen. Here it is only the relative function value that matters, whereas with roulette-wheel selection, the exact value is taken into account.

So tournament selection is less precise. However, it is faster, and there is one other advantage. We can split the merit function into two parts. We only have to calculate $E(\overrightarrow{y})$ and minimize it.

We found that it has about the same performance as roulette-wheel selection. The results will be discussed in Section 4.3.

3.3 Local optimum avoidance

This is a technique we thought might help combat the local optimum problem. The algorithm (like any evolutionary algorithm) has a tendency to eventually stagnate. The objective function value doesn't improve anymore. The population is stuck in a local optimum. If you want to find the global optimum, the population will have to get out of there and never return.

To this end, we record the points where the population gets stuck and proximity to these points influences the objective function value negatively. Every time stagnation is detected (if the objective function value hasn't improved within the last g generations), the local optimum is recorded and the population is reset to random positions.

The problem here is that proximity is not well defined for binary strings. The hamming-distance is one choice, but it does not define a smooth landscape. It is possible that a population is hanging around (insofar as this term has any meaning in binary space) a certain local optimum, only hamming distance 3 or soaway from the global optimum. Scoring proximity to this local optimum negatively could result in the global optimum never being found. So with this technique, the algorithm has no *global convergence*.

Then there is the question of how strongly the avoid-points influence the fitness value and how much the actual distance factors into this.

3.4 String encoding

In a desperate attempt to change the evolutionary landscape, we attempted to encode the binary sequence to gray-code before mutating it (and decoding it after the mutation). This turned out to have a negative effect on the results, so we will not discuss this method any further (see Section 4.5).

3.5 Intelligent mutation

Another (possibly less desperate) attempt to change the landscape involved a smarter mutation operator. The operator would attempt several random mutations and choose the one it found most effective. The question would then be what the most effective choice would be.

One option we tried was to choose the mutation that resulted in the highest fitness. The other option (specifically meant to smooth out the landscape) was to choose the mutation that made the smallest change to the fitness value (be it positive or negative).

3.6 Random subset introduction

Another way to escape out of local optima might be to introduce a portion of completely random individuals each generation. Possibly with a higher crossover rate to take advantage of this and a lower mutation rate to make that operator fully exploitative (and less explorative).

It should introduce more variety into the population. This means the algorithm might be slower, but with more opportunities to escape local optima.

3.7 Tuning algorithm

During the first trials, we found that the starting parameters (population sizes, mutation rate, crossover rate) had great influence on the results. Instead of trying to finetune these parameters by hand, we wrote a second evolutionary algorithm to find the starting parameters of our genetic algorithm.

It starts off with a population of 10 individuals which hold three random starting parameters:

1. Offspring-factor: This is the ratio between the parent population size and the offspring population size. The parent population size is fixed at 15.
2. Crossover rate: The probability that a single-point crossover will take place in the recombination stage.
3. Mutation rate: The probability that a mutation will take place on a single bit of an individual in the mutation stage of the autocor algorithm.

They start randomly within the boundaries of the problem, of course. The mutation rate is initially allowed to range from 0 to 0.1. The mutation rate from 0 to 0.08 and the offspring factor from 2 to 8.

Every tuning-generation the population is increased to 20. The extra 10 are chosen with roulette-wheel selection on the initial population. Then mutation occurs on the extra individuals. Every individual gets assigned a fitness value, which is calculated as the average over the results of 10 runs of the autocor algorithm with a fixed amount of function-evaluations[2].

Hopefully, after enough generations of the tuning algorithn we will get a pretty good parameter-configuration for thi string length.

The best configuration found with this algorithm will b used for all variations of the algorithme except for the ran dom subset variation, which needs its own parameter tun ing. There is an extra parameter to the autocor algorithn in that case. The percentage of the population that is fille with random individuals. We used a slightly altered ver sion of the tuning algorithm for this case. One that als evolutionarily varies the randomness percentage.

[2] The reason for giving each run a fixed amount of function evaluations and not a fixed amount of generations is that w want the resulting set of parameters to use the allotted time efficiently. We do not want the tuning algorithm to blindly increase the size of the offspring population.

4. EXPERIMENTAL RESULTS

4.1 Tuning algorithm

We instructed the tuning algorithm to allot 63000 function evaluations to each run of autocor. Of course, if autocor was just in the middle of a generation, it is allowed to finish it.

Each individual calculates its fitness value by the average of 10 runs of autocor. It is a $(10 + 10)$ strategy. So 10 parents and 10 offspring each generation.

The progression of the parameters and the average and best fitness of autocor are displayed in two graphs (Figures 3 and 4. We used two graphs because the mutation and crossover rates are too small to compare to the fitness values and the offspring factor. Had we plotted them in the same graph, their changes would have been indistinguishable.

As you see, we eventually found a globally optimal solution for string length 50. $f(\vec{y}) = 8.16993$.

$\vec{y} =$

```
 1  1  1 -1 -1 -1  1  1  1 -1 -1 -1 -1
-1 -1 -1  1  1  1 -1  1  1  1 -1 -1  1
 1 -1  1  1  1 -1  1  1 -1  1 -1  1 -1
 1  1 -1  1  1 -1  1  1 -1  1 -1
```

The parameters the algorithm eventually found are these:

- Parent population size: 15
- Offspring factor: 2.33913 (Offspring size = 35)
- Crossover rate: 0.08112
- Mutation rate: 0.02737

These gave an average fitness value of 5.24329.

4.2 Baseline algorithm

We ran this algorithm 100 times, with the parameters found by the tuning algorithm. The results are displayed in the histogram of Figure 5.

The average fitness value was 5.0201. The best found fitness value was 6.4767.

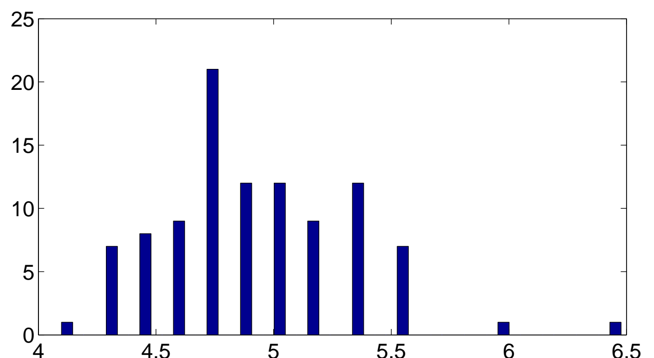

Figure 5: The histogram over 100 results of the baseline algorithm.

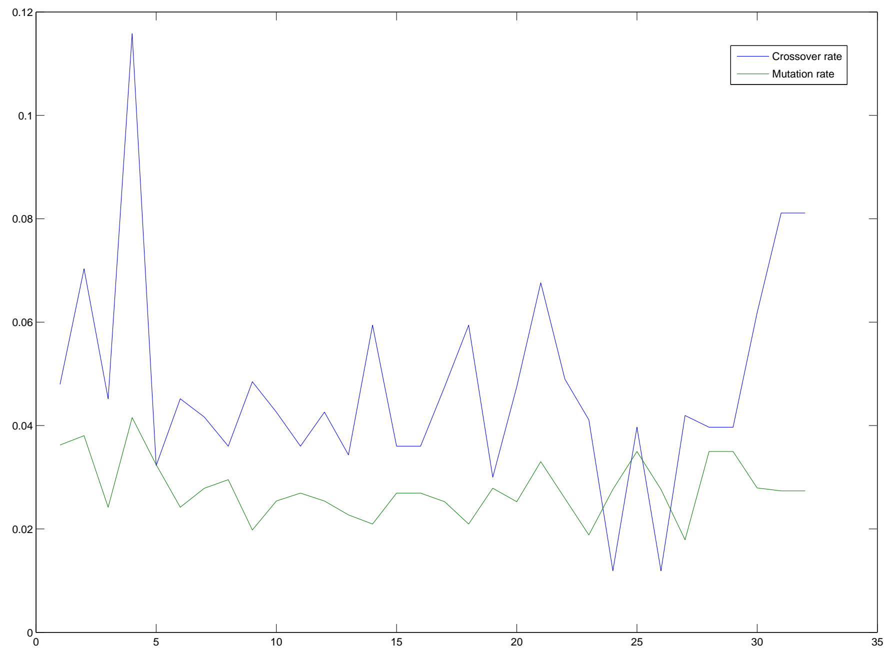

Figure 3: Progress of the crossover rate and mutation rate in 32 generations of the tuning algorithm.

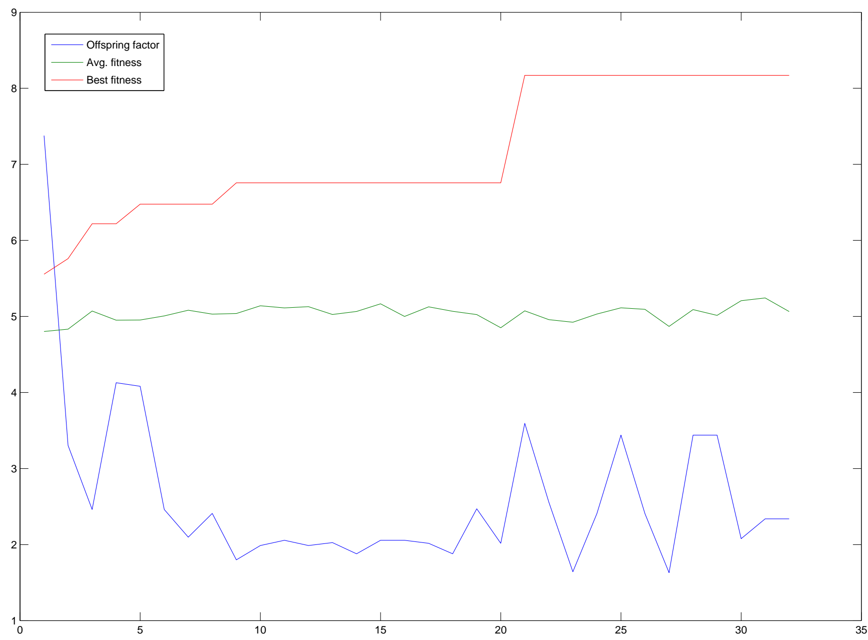

Figure 4: Progress of the offspring factor, the average fitness value and the best fitness value in 32 generations of the tuning algorithm.

4.3 Tournament selection

We ran this algorithm 100 times, with the parameters foun by the tuning algorithm. The results are displayed in th histogram of Figure 6.

The average fitness value was 4.7710. The best found fitnes value was 5.9809.

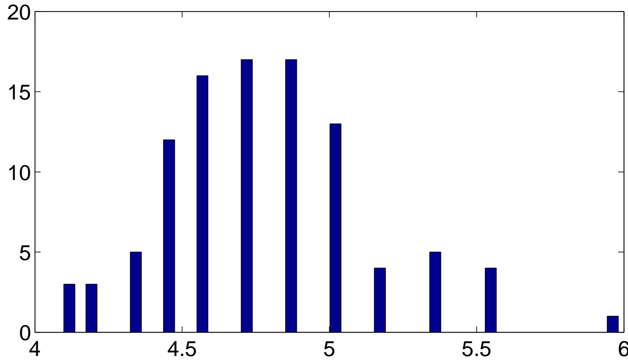

Figure 6: The histogram over 100 results of the tour nament selection variation.

4.4 Local optimum avoidance

Apart from the periodic reset and avoidance-list, this algo rithm was pretty much the baseline algorithm. However, fo the performance question to be answered, we would need t give this algorithm more time, because it is based on lon term exploration of the landscape. However, it is too time consuming to run 100 tests with enough generations for this algorithm to work its best.

Even so, we did run a couple of tests manually, and didn't get a very good improvement. This may be because the hamming-distance may not be the best distance measure for this problem.

4.5 Gray code encoding

We ran this algorithm 100 times, with the parameters found by the tuning algorithm. The results are displayed in the histogram of Figure 7.

The average fitness value was 4.0391. The best found fitness value was 5.5556.

4.6 Smallest-change mutation

We ran this algorithm 100 times, with the parameters found by the tuning algorithm. The results are displayed in the histogram of Figure 8.

The average fitness value was 4.2291. The best found fitness value was 5.0201.

4.7 Highest-value mutation

We ran this algorithm 100 times, with the parameters found by the tuning algorithm. The results are displayed in the histogram of Figure 9.

The average fitness value was 4.3941. The best found fitness value was 5.7604.

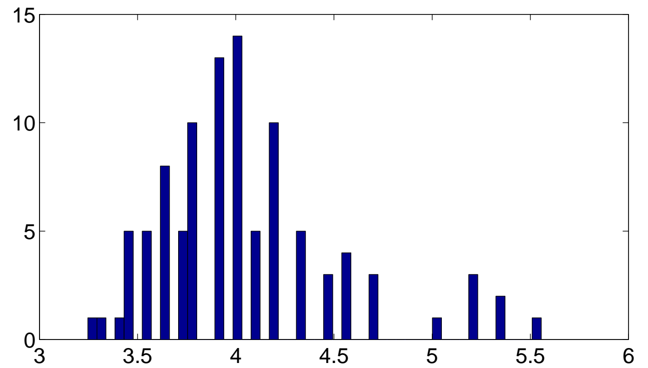

Figure 7: The histogram over 100 results of the gray code variation.

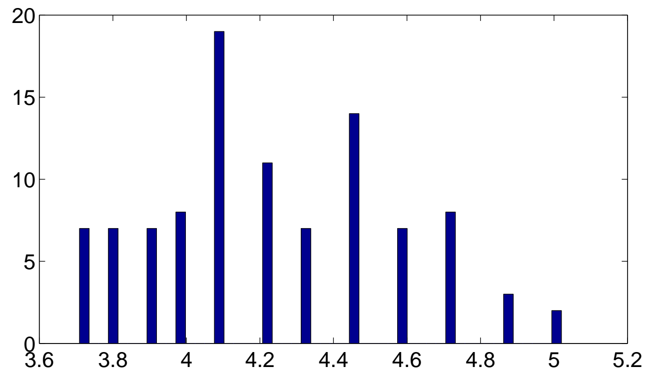

Figure 8: The histogram over 100 results of the smallest-change mutation variation.

4.8 Random subset

We ran this algorithm 100 times, with the parameters found by the tuning algorithm and a randomness of 30%. The results are displayed in the histogram of Figure 10.

The average fitness value was 4.7793. The best found fitness value was 5.7604.

5. DISCUSION

It seems that the baseline algorithm performed best by far. However, this may be because the parameters were specifically finetuned for this algorithm. The variations may have merit, but their parameters probably have to be finetuned specifically for them as well, before they can perform better.

Also, in the case of the tournament selection algorithm, we believe that it performs worse because the selection of the mating partners is slightly more random than they would be with roulette-wheel selection. Only their relative fitness values are used, whereas with roulette-wheel selection the probability of a partner being chosen is proportional to its fitness.

The gray code method was introduced by us, only to see what kind of influence (if any) it has on the mutation operator. We didn't think it would perform very well, because the gray code encoding was arbitrarily chosen without look-

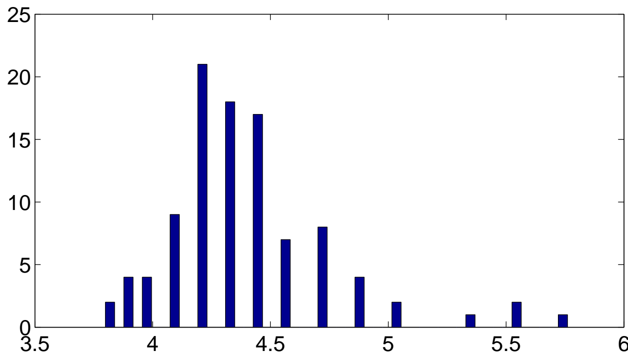

Figure 9: The histogram over 100 results of the highest-value mutation variation.

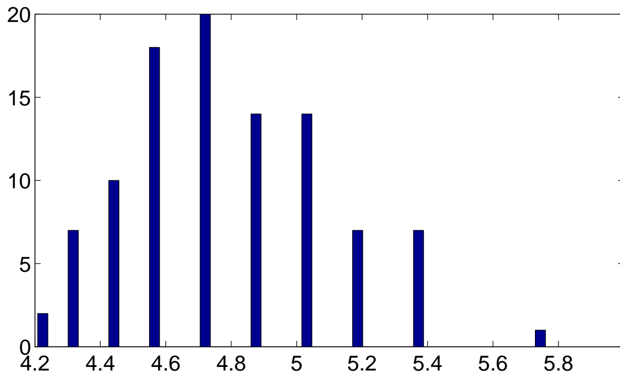

Figure 10: The histogram over 100 results of the random subset introduction variation.

ing at the energy function at all. The results show that we were right. But it did have influence. the question is, is the standard encoding of the binary sequence optimal? Or is there another encoding that would work better?

The intelligent mutation methods didn't perform well because we only allotted a fixed amount of function evaluations, and their mutation operator used them up very quickly. It doesn't seem to be a smart way to use them. They didn't get many generations because of this. This is even more the case for 'smallest change' method, because it is designed specifically to rise slowly (it takes the mutation that gives the smallest change to the fitness).

Last but not least, the random subset introduction method didn't perform very badly, but worse than the baseline algorithm. The search space could be too large for any randomly introduced individuals to make a change. However, we did 'waste' function evaluations on these random individuals. So it got less generations with the good ones than the baseline algorithm did. Another variation that might be tried sometime is to introduce individuals from T generations ago. They are not random, but might choose a different path from that moment on, and not get stuck in the same local optimum as the main population did.

6. CONCLUSIONS

We introduced several approaches to the genetic algorithm to find the solution to the Low Autocorrelation Problem. Some were successful, some weren't. The most successful move would have to be the tuning algorithm that finds the right parameters for autocor.

It is very possible that the variations we tried would work if we had taken more time to fine-tune and improve them. Specifically the encoding approach. If a good encoding could be found that smoothens out the landscape, it would make a world of difference.