Evolutionary Algorithms

Assignment 1 - Genetic Algorithm

Sevak Mardirosian
s2077086
s.mardirosian@umail.leidenuniv.nl

Olzhas Aldabergenov
s1928643
o.t.aldabergenov@umail.leidenuniv.nl

November 8, 2017

1 Introduction

In this paper we are going to briefly present the idea behind Genetic Algorithms in comparison with Mote Carlo Search algorithm for low Autocorrelation problem. We will start with a description of the fundamental problem statement and what the reason is of us doing implementing and comparing with MC. Further we will walk you through the implementation of Genetic Algorithm and explain pretty much our approach and what kind of issues we run into during the implementation. Afterwards we will present a table with our experiments to the required dimension(s) from both sides (GA and MC) along with plots and diagrams in which we prove which one of those algorithms has a lower autocorrelation. By the end we will draw the line with some discussion, in which one of those algorithms is better than the other and on behalf of this draw a solid conclusion.

2 Problem Description

Genetic Algorithm has specific way of solving problems rather than Monte Carlo search and we found difficulties on realization. Both algorithms have own constraints which are different from each other. First problem which we face was on generating population. There were two possible way of doing that: a) taking n-length bit string and equally divide it to subsets and take them as population (as shown in slide 3). b) generate multiple population of n-length bit string and in future select best of them. Secondly, problem with fitness function because at the beginning we use decoding function to evaluate subset. Last problem related to choosing right crossover and mutation probability.

3 Implementation

Important part of creating Genetic Algorithm is population. Starting from this point, our algorithm initialize $log(n) * 100$ sized population (pop_size) which are filled by n-length bit string. Next, we evaluate each n-length bit string by using autocorrelation function (labs.m) which was provided. Afterwards, we launch while loop which works till counts reach evaluation budget. In the loop we have parent selection part where probability calculated by using the following formula.

$$P_i^{'} = \frac{fi - c}{Sf - cu} withSf = \sum_{j=1}^{u} f_i$$

Only fourth population size will be selected as parents(ps_size = pop_size/4) to get fittest parents and some parents might be repeated. Moving to forward, we crossover parent by using probability of De Jong 1975 pcrossover=0.6 and create 3 times more new generation from selected parents to get better offspring. Then, we mutate it by using standard probability pm1= 1/n to get fittest parents and some parents might be repeated.

Generating population was one of the tricky moment on creating GA algorithm because slide 3 describes method which divide n-length bit string into subset. Each element become a population. Afterwards, each subset need to be decoded and evaluated. Our algorithm was wrong at the beginning, we made few mistakes. For example, code which is below:

```
xopt = rand(1,n) > 0.5;
fopt = labs(xopt);
for i = 1:pop_size
  pop_{i} = xopt((i-1)*lx+1:i*lx);
  pop_fit_{i} = u + (((v-u)/(2^lx-1))*bin2dec(num2str(pop_{i})));
end
```

Firstly, we generate n-length bit string and evaluate it by using labs method (autocorrelation) and divide n-length bit string to equally subset. Eventually, it gives us pop_size = n/lx (lx-length of bit string in subset). Next, we put each set to cell array and evaluate fitness by using decoding function.

$$h_i(a_{i_1}, ..., a_{i_{lx}}) = u_i + \frac{v_i - u_i}{2_x^l - 1} . \left(\sum_{j=0}^{l_x - 1} a_{i_{j+1}} . 2^j \right)$$

After crossover, mutation we merge subsets and finally evaluate it by autocorrelation method. Many testing and comparing work to Monte Carlo search was done (not included to this report). We've thought that it works well because it almost gives us equal result, same as Monte Carlo search. However, we realize that we do not need decoding function to evaluate subsets and start use autocorrelation method. In addition, we refused to use cell array (See it below)

```
pop_fit_(i) = labs(pop_(:,i));
```

Then again, nothing is changed. We've got same result. After many running algorithm we found that autocorrelation method gives us different result when you merge subset again. For example, let say that X,Y subsets of 10 length bit string . X = [0 1 0 0 0] , Y = [0 1 0 0 0]. Fitness of each subsets X,Y equals to 6.2500 respectively. Despite that this significant good result, we get 1.3514 after merging this subsets to one XY = [0 1 0 0 0 0 1 0 0 0].

4 Experiments

Below you can find our results that we have executed based on the number of required dimensions. During the experiments we were able to get a better idea about low auto correlation comparison between Monte-carlo search and Genetic Algorithm. During the testing we run against few issues such as the lack of memory [1]

D	Monte-Carlo Search		Genetic Algorithm	
	Avg	Std dev	Avg	Std dev
10	3.8462	0.000000000000000004556259157	3.8462	0.000000000000000004556259157
20	4.6925	0.970701111392119	4.9641	0.911501059848678
40	3.0987	0.203502390649348	3.1892	0.301297267384444
80	2.2680	0.185549441748913	2.4136	0.179444293119991
160	1.8038	0.0931510619012652	1.8466	0.0824696362594252

Table 1: Final solution quality after 5,000 * dimension function evaluations, averaged over 20 runs

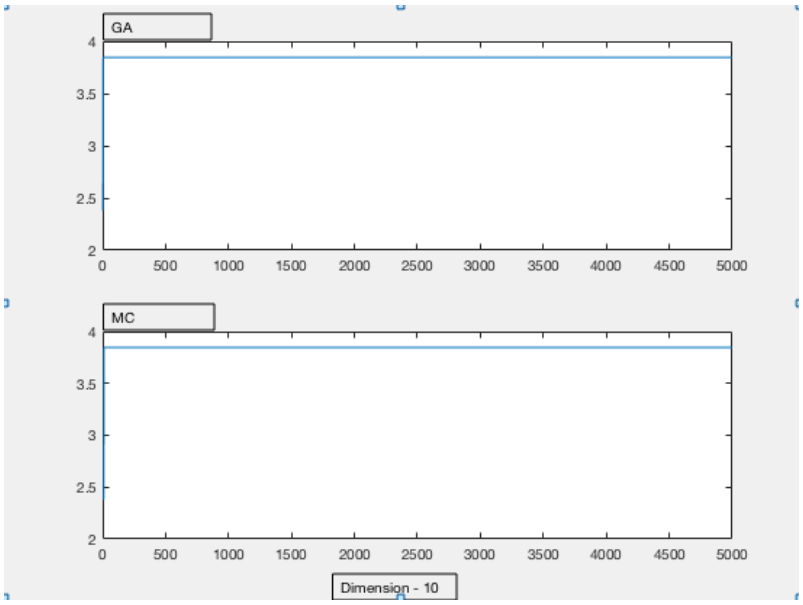

Figure 1: Comparison of GA and MC in 10 dimension

[1] Perhaps for the fact that we were using the Octave to run our experiments and so we were having number of issues regarding such as not willing to run on Linux and memory leak. We could have dogged this problem at first place if we have tried to use Matlab, but given the time limit and license issues we stuck with Octave

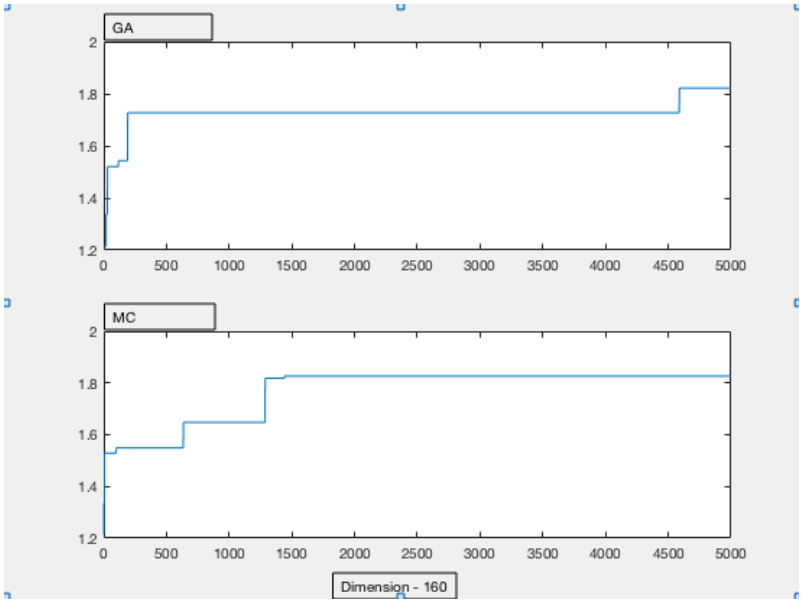

Figure 2: Comparison of GA and MC in 160 dimension

4.1 Discussion and Conclusion

During this assignment we learned few things, such as working with genetic algorithms and understanding the concept of it works (selection, crossover, mutation, etc..). We also had the chance to explore how optimization works and the working of Monte-Carlo search algorithm. By the end we believe we achieved the the idea of GA performance testing against to Monte Carlo (MC) search which was provided for this assignment.

For some of us GA was quite brand new topic. As a CS student I never came across such a topic and so learning the concept and the idea was quite essential to get started with the assignment. The topics were well explained by the professor and his assistants during the lectures and the workgroups and session labs helped a lot during all these times. And so by the end we believe we got a solid understanding of the concept and so we were able to get a good grip on the assignment as well.

In terms of execution we were having a bit of difficulties regarding matlab. The university does provide a version of matlab on the workstations, but as you might know most of us work on their laptops and places other than being at the university. We decided to give Octave a go for this one but unfortunately it didn't go pretty well. We believe the code we wrote was quite simplistic and somehow not expensive in terms of memory, yet Octave was having trouble executing and sometimes it produces different results on each run which was at some point confusing. On Linux we were unable to execute the code for some reason. We tried this on different Linux machines. The application kept crushing or the execution process was pretty slow.

In terms of solving low autocorrelation for binary strings. We believe we have achieved what we intended to achieve from the start point of this assignment which is proving GA has lower correlation than random search algorithm [2].

Genetic Algorithms use randomness to create initial solutions, new individuals (via mutation and crossover operators) and even select individuals. However, selection tends to favor the most fittest individuals. Also, genetic operators are not purely random but use randomness to (often slightly) change individuals. GA is a stochastic search method. To explain the differences between the GA and a random search method it would be useful first, to explain the differences between the stochastic and random phenomena. The term stochastic refers to a process which periodically and apparently-independently happens but a kind of dependency exists.

[2]Pure random search is often called Monte-Carlo Search

In a stochastic search algorithm like GA, ACO, PSO etc., the randomness is controlled by some general rules governing the evolution (optimization) process. However, in a random search, in fact, we only have a local blind trial-and-error at each iteration to hopefully escape local solutions. In other words the GA has pretty much nothing to do with with random search. However the random component of the algorithm (of the search) can be very important in some optimization problems and can be (must be) increased by increasing the mutation probability. Also, it should be noted that the mutation operator generates randomly a new solution in the vicinity, while a pure random search generates a new solution randomly in any point of the search space.

By the end the idea of comparison goes naturally to GA which has much lower autocorrelation than random search – or Mote-Carlo search algorithm for the reasons stated above.