Evolutionary algorithms Jordy van Miltenburg (s2336553)
j.p.van.miltenburg@umail.leidenuniv.nl

# 1 Introduction

Human evolution a process that proved itself to work in an efficient way, solving the problems given by the universe. After many generations, humans learned to survive on this earth by adapting themselves to withstand the harsh conditions thrown at them. Developing a way to communicate with each other and learned how to make fire. But not everyone survived the less fit among the population are the most vulnerable, therefore more likely to die first. The remaining population will produce the new offspring, the humans of the future. Genetic algorithms are based on human evolution in order to solve a given problem through selection, mutation and recombination or in other words crossover. All the fundamentals of the human evolution form a basic genetic algorithm

# 2 Assignment

## 2.1 Problem

Genetic algorithms (GA) is a subfield of evolutionary algorithms focused on solving problems in the binary spectrum. Opposed to evolutionary strategies which are focused on the continuous numbers. For this assignment, it's not necessary to go into the continuous number because the problem set we have to tackle is composed of 23 pseudo boolean problems, therefore only using binary representation. In order to solve the 23 problems, we have to develop a GA that solves the problems as good as possible. The choices that we make for the implementation determine the performance of the algorithm. For analysing the created GA's a tool is used the IOH Profiler with this tool different algorithms can be compared to your own algorithms or a preset of example algorithms.

## 2.2 Selection

In each generation, a set of parents is necessary to breed a new generation. The selections made during this process are based on the fitness value of the parents. Where the fitter parents have a higher chance to be selected for the breeding of the new generation. A different selection operator is chosen for a particular problem, two of those operators are proportional selection and tournament selection. Both of the operators have a random number generator to prevent the fittest parents to be chosen and the convergence to a solution will not be as fast. By doing selection in this way a broader search space is investigated.

### 2.2.1 Proportional selection

The proportional selection operator works like a pie diagram where the fitter individuals have more space on the pie and the less fit a smaller space. The fittest has, for example, five

spaces on the pie and the least fit only one space. In this way, the randomly generated number has a higher chance to land on a fitter individual but there still is a chance that a less fit parent is chosen.

### 2.2.2 Tournament selection

The tournament selection operator is also working with a random number generator but every individual has an even chance in the first stage of the operator. Tournament selection works by selection k individuals from the whole population. To give an example the population size is 10 and k equals 4, this means that 4 individuals are randomly selected from the population. Of those 4 individuals, only the best is selected for breeding. Opposed to proportional selection tournament selection has a higher chance to select worst parents for breeding, which does explore the search space more.

## 2.3 Crossover

The breeding mentioned earlier is also called recombination or crossover. Within this operator, the selected parents—done by using one of the earlier mentioned selection operators—are used to create new offspring for the next generation. The genes of the parents are combined to form offspring that has a bit of both. Different crossover operators can be used within a genetic algorithm. 1 point, n point or uniform crossover are the most common ones.

### 2.3.1 1 point crossover

The name already suggests a lot about this crossover operator. The offspring is created by selecting the first part of the genes from parent 1 till a random point and the following genes from parent 2. To give an example the individuals consist of 10 genes and the random split point is 4. From parent 1 the first 4 genes are used in de offspring, the remaining spots in the offspring are filled with the last 6 genes of parent 2. By using this operator there is a high chance to copy a large piece of correct genes into the offspring. On the contrary, this could result in getting stuck in a local optimum because of the fast convergence.

### 2.3.2 n point crossover

N point crossover is basically the same as 1 point crossover but instead of having only 1 point where the crossover happens. There are multiple points selected at random to crossover between parent 1 and parent 2.

### 2.3.3 Uniform crossover

Another way of recombining two parents is uniform crossover. This operator cycles through all the genes in the parents and flipping a coin to select the right gene. So there is a 50/50 chance that the gene from parent 1 or parent 2 is selected.

## 2.4 Mutation

Just like in human evolution mutation plays a role in GA moreover when the crossover happened there is a slight chance the genes mutate. With a predetermined mutation rate, the genes of the offspring stay in their original state or they are flipped. The mutation operator does this gene by gene. Without the mutation operator, there is no random change therefor there is no exploitation of the search space.

# 3 Implementation

## 3.1 Genetic algorithm

For this assignment, I chose to go for the (μ,λ) implementation. The comma notation means that the parents and offspring are not both considered to be selected for the new generation. With the comma notation, the offspring is always used for the next generation, thus forming the new parents. For the selection operator, I implemented a tournament selection function that selects two parents for the crossover operator. The function that I implemented for the crossover operator is uniform crossover. To come to this set of functions and parameters I did multiple tests with some deviating results, explained in the experiment section. Overall the algorithm I wrote is performing well compared to the example algorithms provided in the IOH profiler. In some specific problems even beating all the algorithms see fig.1

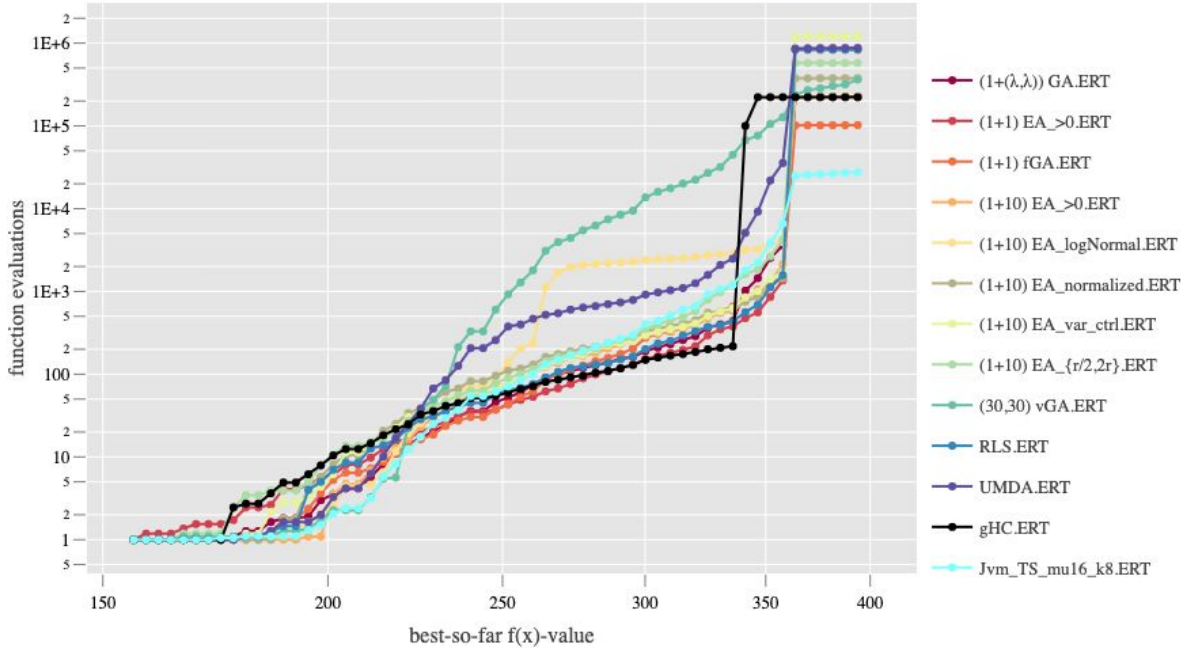


Fig.1 Plot of the expected runtime for problem 20. Here we can see that my algorithm outperformed all the example algorithms within the IOH profiler.

## 3.2 Defining the final parameters

In the process of defining the final parameters for my GA, multiple things had to be accounted for. The population size μ which I set to 16 and offspring λ which is also 16. For the selection operator, I used a tournament selection function with a k of 8 as explained earlier 8 random parents are chosen of the 16 the best of those 8 is pushed forward as the selected parent. For the crossover operator in used a crossover rate of 0.95 according to [1]. The mutation rate is set to 1.0/n where n is the dimension of the problem, so the number of genes within 1 individual. The set parameters in the IOH Experimenter are: problem 1-23, a dimension of 100, the budget is set to 50000 and the independent runs to 20.

# 4 Experiment

## 4.1 Selection operators

For selecting the best operator for this particular problem I've experimented with three different selection operators. Best fitness, proportional and tournament selection. In all the experiments I've done, just picking the parent with the highest fitness is not working. The search space is reduced too fast. The algorithm takes to long to come anywhere. Proportional selection is better at certain problems within the given set. Looking at the Aggregated Empirical Cumulative Distribution (AECD) we can see that on average the proportional selection is not the way to go in comparison to for example tournament selection.

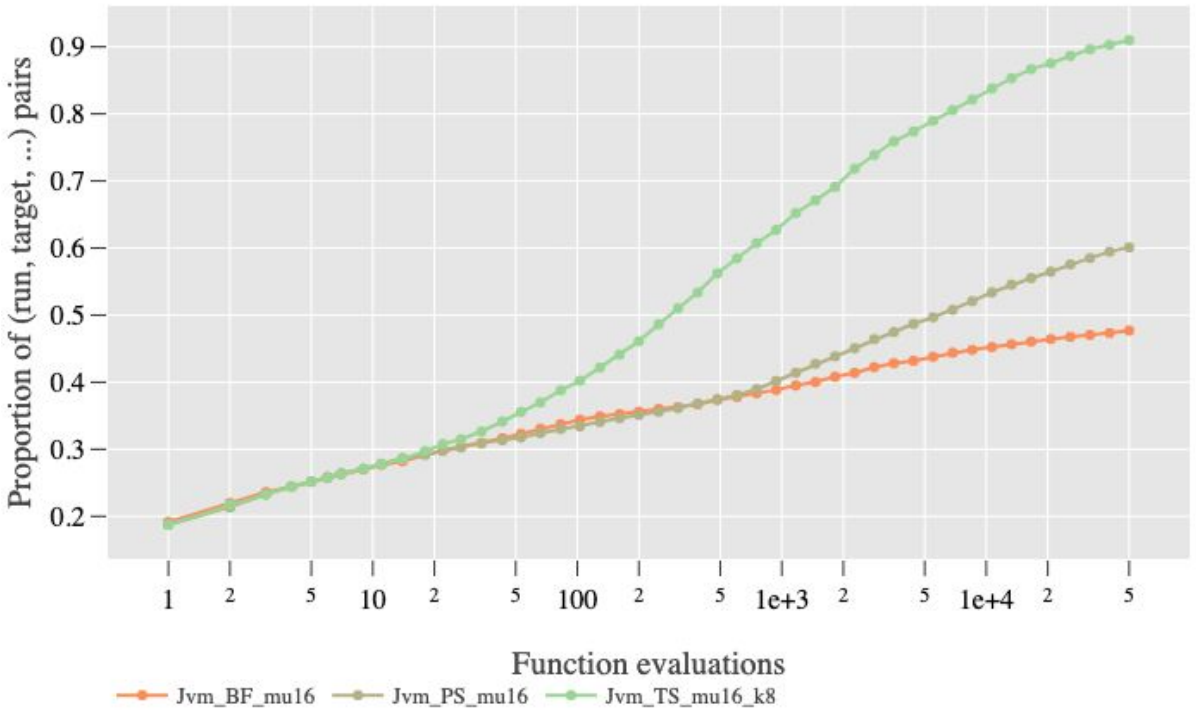


Fig.2 Shows the difference between best fitness (BF), proportional selection (PS) and tournament selection (TS) with the use of aggregated empirical cumulative distribution.

Tournament selection had a better overall performance compared to the other two. At the start of the one max problem both the PS and TS have an equals start. But at around an f(x)-value of 55 the PS spikes and isn't able to yield a 100% score. The TS continues in the right direction. Looking at the deviation between PS and TS we can see that the change of parameter has more impact on PS. In figure 2 we can see that by changing the population size of the PS algorithm, the f(x)-value deviates more than the TS algorithm.

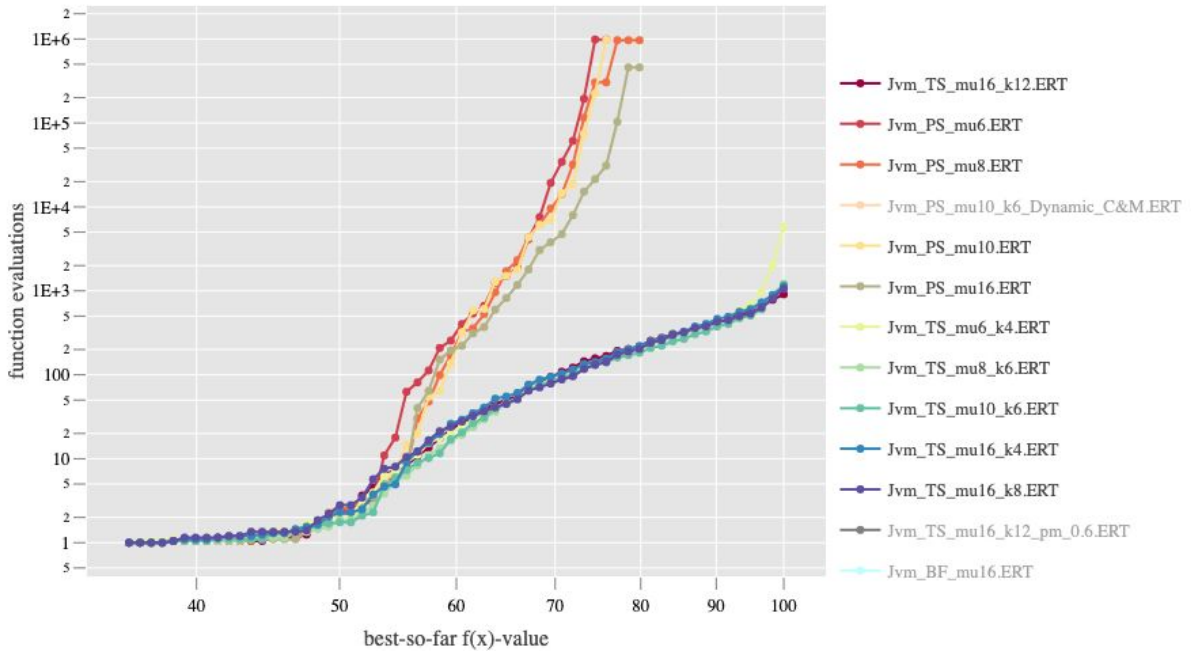


Fig.3 Expected runtime for problem 1 of the PBO set containing different variations of PS and TS operators.

## 4.2 Dynamic crossover and mutation

The crossover and mutation rate determine how often the operator is activated. In the previous experiment, the GA had a static crossover and mutation rate. The following experiment has a dynamic crossover and mutation (DCM) rate with the implementation of a scheme presented by [2]. The rates change according to the fitness of the parents and the offspring. Each generation the overall performance is calculated for both the crossover and the mutation. The calculated values are used to adjust the crossover and mutation rate throughout the different generations with a set factor 0.001 in my case. In fig.4 we can see that both the crossover and mutation rate change throughout the run.

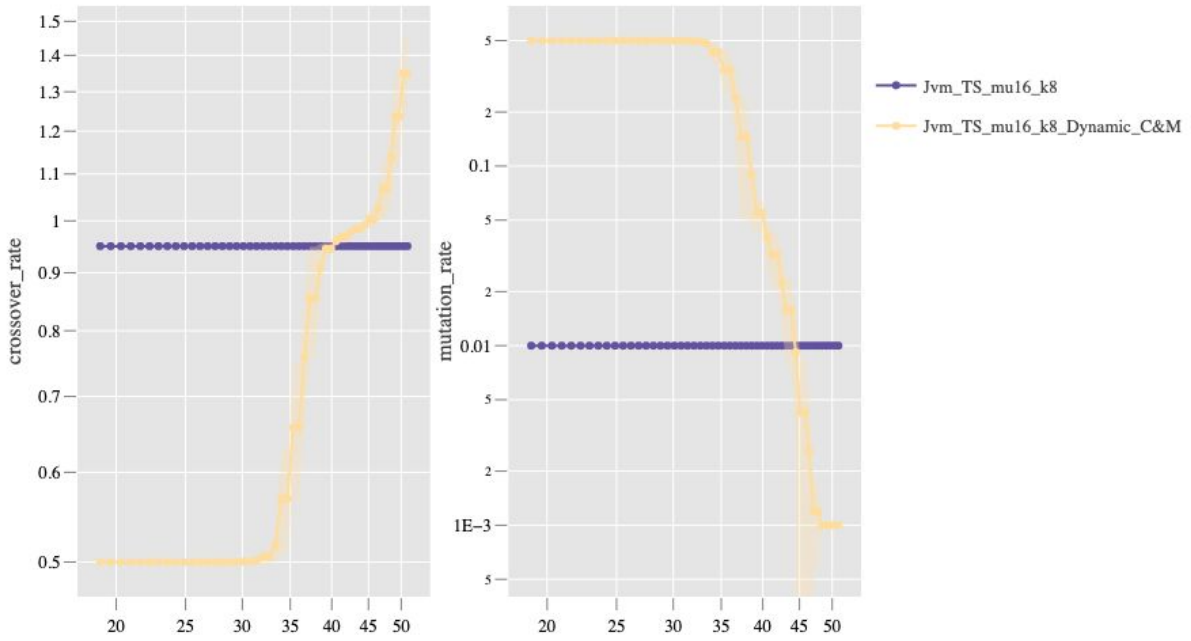


Fig.4 Difference in crossover and mutation rate between a static and dynamic algorithm.

This change in the parameters may be a good thing in some situation but for my algorithm it was not beneficial. If we look at the amount of evaluation needed to reach the best possible fitness. One thing that stood out to me is how at a certain point the algorithm with the DCM performs a lot better. In fig.5 we can see that after 20000 evaluations or a fitness of 38 the dynamic algorithm starts performing for a little while and then cuts off. In terms of f(x)-value gain over function evaluations.

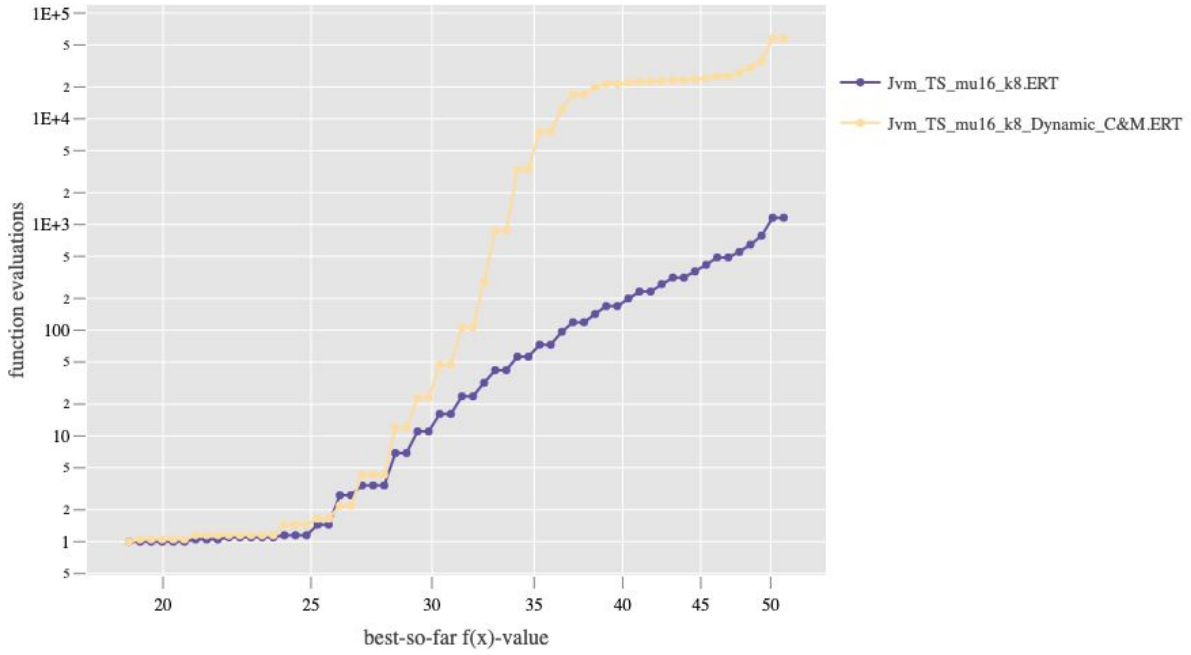


Fig.5 Expected runtime for problem 8 for two algorithms. One with static crossover and mutation rate the other with dynamic crossover and mutation.

# 5 Discussion and conclusion

For this practical assignment 1 I proposed a (16,16) algorithm to tackle the 23 pseudo boolean problems within the IOH profiler. The proposed algorithm performed decently compared to the example algorithms in some case better than all the algorithms. The (16,16) algorithm is built with tournament selection k = 8. The use of tournament selection greatly increased the performance of the algorithm compared to proportional selection. For this algorithm, the introduction of a dynamic crossover and mutation rate was not beneficial and didn't make the algorithm perform better than with static values.

# Reference

[1] Grefenstette, J.J. (1986) Optimization of Control Parameters for Genetic Algorithms. IEEE Transactions on Systems, Man and Cybernetics, 16, 122-128.

[2] Lin, Wen-Yang & Lee, Wen-Yung & Hong, Tzung-Pei. (2003) Adapting Crossover and Mutation Rates in Genetic Algorithms.. J. Inf. Sci. Eng.. 19. 889-903.