Optimization Algorithms

Benjamin Ell-Jones - 19016317

# Introduction

In this paper I intend to research and compare competitive performance of several numerical functions. For example, I will be testing the performance of Fitness, selection, mutation, and crossover functions. I will be presenting my findings in the form of graphs. I will approach this task by selecting a few of each of these functions and comparing the results of each test.

# Background Research

## Introduction to the research

Another algorithm for solving optimization problems is the artificial bee colony. The algorithm works similar to a bee colony as you would expect, it has 3 types of bees, employed bees, onlookers and scouts (Karaboga, Basturk,2007). Within this algorithm possible solutions are not represented as individuals with genes as is the case with genetic algorithms and a fitness. Food source represents possible solutions in an optimization problem and the nectar represents the fitness (Karaboga, Basturk,2007). Using this information, it is clear that the representation of the problem is similar in concept both genetic algorithms and bee algorithms, have a problem solution they represent in their respective ways then they have a way of representing the fitness of said problem solution.

## ABC Structure

Breaking down the steps, the first step of the process is the initialization phase. This is where a randomly distributed initial population is produced (Karaboga, Basturk,2007).

The next phase is the employed bee phase, where in employed bee search for new solutions within the neighborhood of the food sources their memory, when they find one,

they evaluate the fitness of the food source (Karaboga, Gorkemli, Ozturk, 2014). The next step once a new food source is produced and greedy selection has been applied between it and its parent, employed bee share the information about the food source with onlooker bees in the hive that are currently dancing in the dancing area (Karaboga, Gorkemli, Ozturk, 2014).

The next phase is the onlooker phase. In this phase food sources are probabilistically chosen by onlooker bees, based on the information that they received from the employed bees (Karaboga, Gorkemli, Ozturk, 2014). In the case of this article, (Karaboga, Gorkemli, Ozturk, 2014) it is stated that a fitness-based selection algorithm like roulette wheel selection can be employed. Fitness based selection is also used in genetic algorithms they are just used in different ways, to select individuals for crossover as opposed to, using selected found food source to apply greedy selection between it and a neighborhood food source after its fitness has been determined (Karaboga, Gorkemli, Ozturk, 2014).

The next phase of the ABC algorithm is the scout phase. Any solutions that can’t be improved through a predetermined number of trials there employed bees become scouts and those solutions abandoned (Karaboga, Gorkemli, Ozturk, 2014). Scout will then start to find new solutions at random (Karaboga, Gorkemli, Ozturk, 2014).

## ABC parameters

Comparing parameter options for both Genetic algorithms, basic genetic algorithms can have multiple parameters, in the case of this research paper, including Generation, population, mutation rate and crossover rate compared with ABCs, in the case of this study “A comparative study of Artificial Bee Colony algorithm” the basic ABC used in this study employs only one control parameter, which is called limit (Karaboga, Akay, 2009). The limit described in the study is, when the limit is exceeded the food source will no longer be exploited and will be considered abandoned (Karaboga, Akay, 2009). However other basic parameters in the study are used such as, population number and maximum evaluation number (Karaboga, Akay, 2009).

## Comparison of Genetic algorithms and ABCs

Both genetic algorithms and ABC have their uses in solving problems, however artificial bee colonies are

# Experimentation

My program’s structure is as follows, first I generate an initial population of individuals. Then test that populations initial fitness. Deep copying is used in all places where the population is being copied from one array to another. After initial fitness has beings calculated the selection process begins. After this step has been completed crossover then mutation is applied to the population. The final step is to calculate the minimum fitness of the generation and the mean fitness of the generation then the elitism function is called, and the best individual is selected and replaced with the worse individual for the next generation.

Several parameters are included at the top of the program they are as follows, N, Pop, MUTRATE, MUTSTEP, GEN, MIN and MAX. N is equal to the number of chromosomes in each individual. Pop is in reference to the max number of populations in each Generation. Gen or generation is how many times the above GA algorithm will run I.E how many times, selection, crossover, and mutation will happen. MIN and MAX are in reference to the minimum and maximum values and given chromosome can be in an individual. MUTRATE or mutation rate is how often an individual solution will be mutated. When dealing with real numbers as opposed to 1 and 0s you can’t simply flip a 1 to a 0 and vice versa to mutate them. Therefore, MUTSTEP or mutation step is required to determine what will be taken away or added to an individual chromosome to mutate it.

# Mutation rate and mutstep combinations and their average resulting fitness

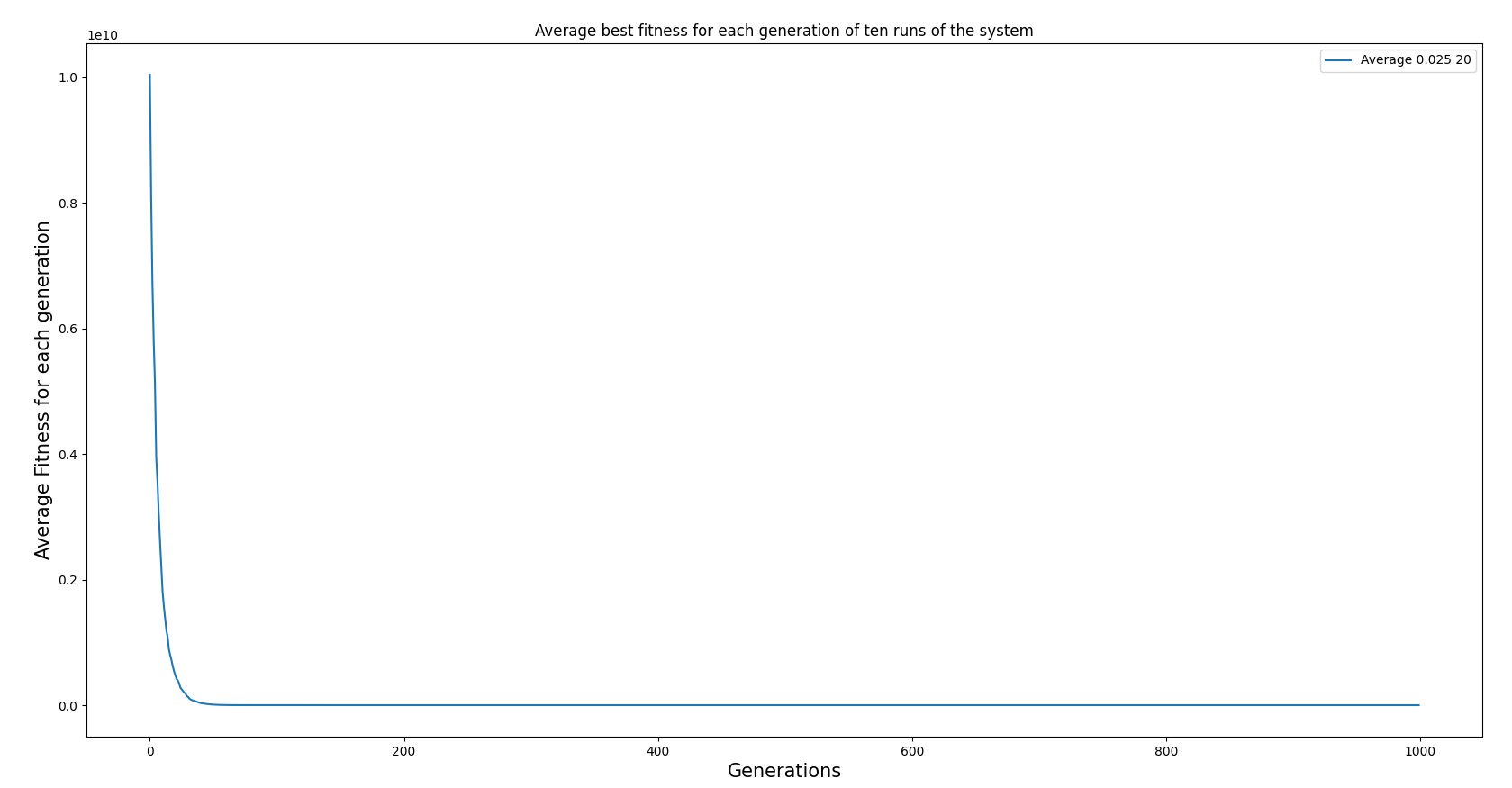
Table

Description automatically generated  
The table above shows a collection of data produced by a mutation rate and mutstep sweep to find the best possible combination of mutation rate and mutstep values. Each fitness value is produced by a test set contains a set of 10 individual test where the same test with the same parameters is run 10 times, this is to account for the inconsistent results obtained when these tests are run due to the fact that mutation rate and population generation is random the results can vary quite a lot.

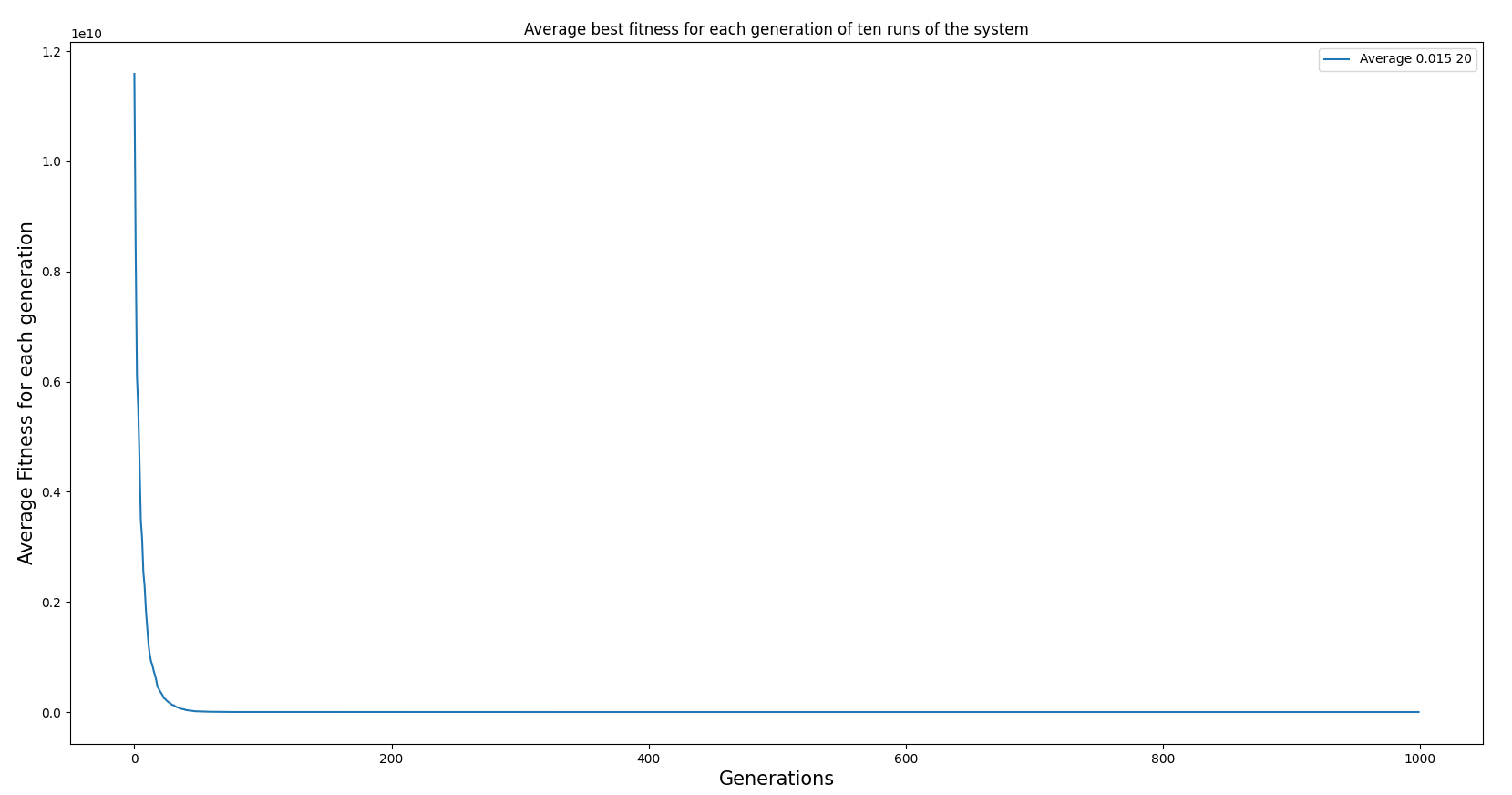
The best fitness values for each run of the system, from all 10 tests for a set is added to one array and then averaged. This then gives one average fitness value for one pair of parameters for example 0.009 mutrate and 10 mutstep will give the average, 890.7604301462177

The other parameters are as follows, population: 100, generations: 1000, and the min and max values for each individual chromosomes in an individual are min: -100 and max: 100.

As can be seen from the mutation sweep for the Rosenbrock fitness function the best fitness is 86. 37863837605724. Meaning a mutation rate of 0.02 and a mutstep of 20 is a good starting point to further fine tune the mutation rate and step.

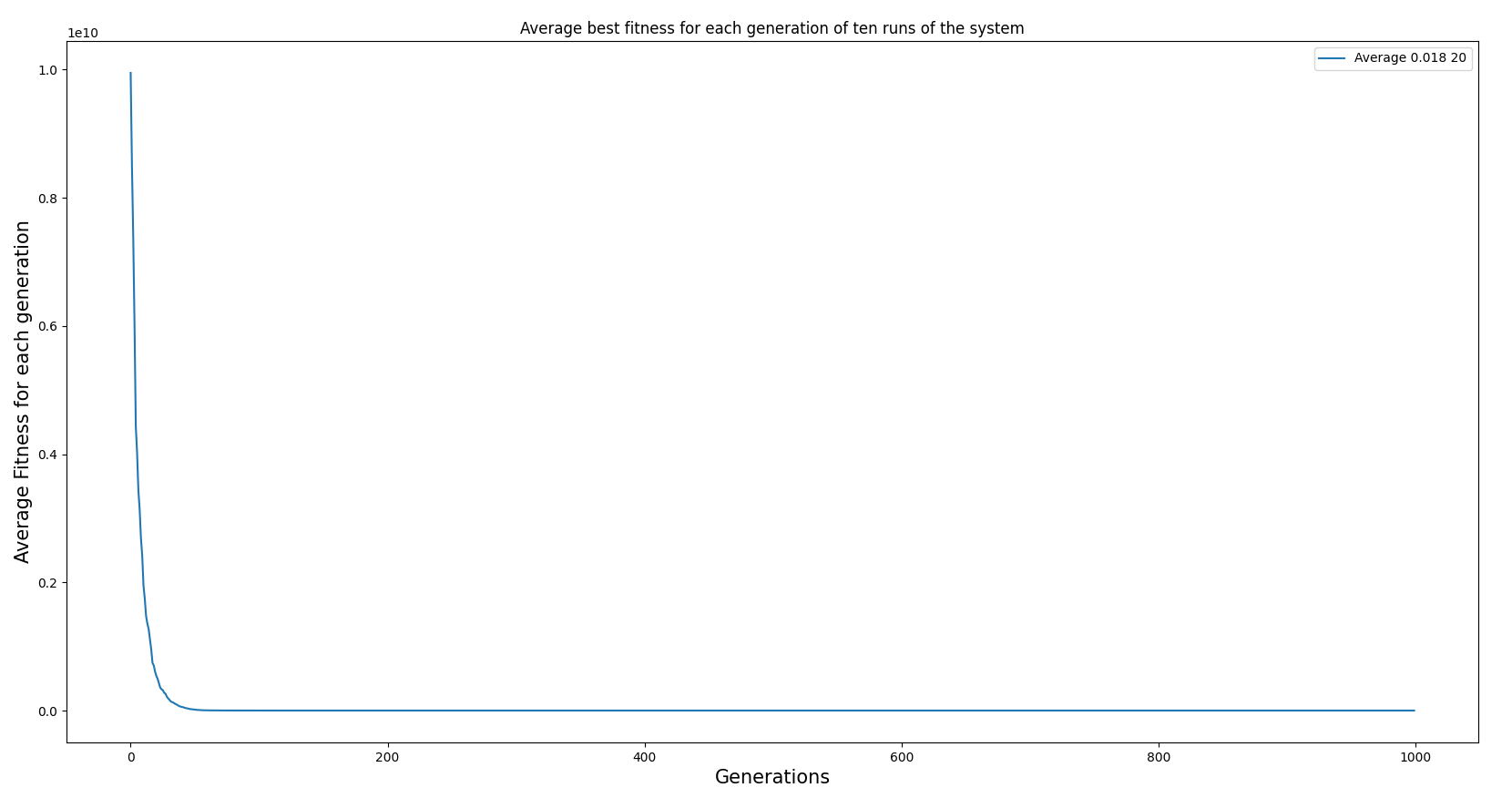


Further testing gave a result of 100.33042501253033average fitness over 10 runs of the system with parameters set to 0.025 and 20. The average best fitness in this case has increased, therefore the best course of action is to reduce the mutrate down to 0.015.



92.04232755107304

85.75877647346996



Graphical user interface, text, application, Word

Description automatically generated  
85.90280031936548

Table

Description automatically generated  
Using the same testing method as the RosenBrock function we can see that -10.747671191312671 is the smallest fitness in this mutation sweep. Therefore, the best starting point for experimentation is 0.02, 10.

Graphical user interface

Description automatically generated with medium confidence  
Average fitness -10.747366262774914

Fitness values = [-10.746675251491215, -10.750616228570166, -10.736150739847638, -10.744122108428925, -10.75081760850445, -10.752249076745283, -10.755871645185875, -10.747797526060046, -10.741251697012238, -10.748110745903318]

A picture containing chart

Description automatically generated  
-10.751050230451302

[-10.749018209457141, -10.749323605541136, -10.755763605532014, -10.752698052683511, -10.74712017066351, -10.752781673067567, -10.752099887316557, -10.742563744291601, -10.755494829999918, -10.753638525960053]

Graphical user interface

Description automatically generated with medium confidence  
-10.750252846698531

[-10.746141420277478, -10.753125291629608, -10.751787159039562, -10.751476847978829, -10.756435924847743, -10.752764638118165, -10.740980017883855, -10.753313970109968, -10.75129667964665, -10.74520651745344]

Graphical user interface

Description automatically generated with low confidence  
  
-10.735657273542557

[-10.750903822462949, -10.748866041875683, -10.743439139668123, -10.7354127922906, -10.741437025171761, -10.7406925522227, -10.734579358912818, -10.724292167316225, -10.712159716793845, -10.724790118710859]

# Conclusions

# References

Karaboga, Basturk, D.K, B.B. (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization[online]. Volume 39, Issue 3. [Accessed 29 November 2021]

Karaboga, Gorkemli, Ozturk, D.K, G.B, C.O. (2014) A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review [online]. Volume 42, Issue 3. [Accessed 30 November 2021]

Karaboga, Akay, D.K, B.A. (2009) A comparative study of Artificial Bee Colony algorithm, Applied Mathematics and Computation [Online]. Volume 214, Pages 108-132 [Accessed 01 December 2021]