# The effect of mutation rate on the performance of a microbial genetic algorithm solving the Knapsack Problem


## Abstract

In this study, a microbial genetic algorithm (GA) is presented to solve the 0-1 Knapsack problem, a combinatorial optimization problem. Genetic algorithms are initialized with a population of genotypes/chromosomes, and through a series of iterations (generations), new solutions and populations are formed according to their fitness until the optimal solution is found or an exit condition is met. The performance of a GA depends on the choice of population size, number of generations, selection, reproduction and mutation rate. This study explores the effect of the following mutation rates: 0.001, 0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.8, 1.0 in order to gain a deeper understanding of the effect of mutation rate on the performance of a microbial genetic algorithm in solving the Knapsack Problem. The microbial GA employed to solve this problem is roughly modelled on Harvey's (1996) [3]. Although previous studies have suggested that a mutation rate of 1/Length, which produces 1 mutation per genotype, is optimal, here we find contradictory evidence for this.


## Introduction

The Knapsack problem describes a resource allocation problem where given a set of items, each with a benefit and volume associated, and a threshold total capacity, we want to find the optimal choice of items such that the total volume does not exceed the threshold, and the benefit is maximised. Genetic algorithms provide a solution to the Knapsack problem, where a genotype encodes a solution to the problem, the phenotype is the fitness of said solution, and by exploring the fitness landscape we can find the best solution. The term fitness landscape describes a representation of the space of all possible genotypes and their corresponding fitness (Sewell, 1931)[1].

**Equation 1. Knapsack problem**

$$maximise \sum_{i=1}^{n} x_i w_i$$

$$subject\ to\ the\ constraint: \sum_{i=1}^{n} x_i w_i \leq W$$

$$W = maximum\ weight\ capacity$$
$$w_i = weight, v_i = value$$
$$x_i \in \{0,1\}, i = 1,2,\ldots . n$$

An important aspect of using a genetic algorithm is the trade-off between exploration and exploitation to ensure that the algorithm doesn't spend too much time in the same areas, but also does spend time in high fitness maxima where the optimal solution may be. A significant choice in balancing the exploration and exploitation of a fitness landscape is in choosing the mutation rate. If the mutation rate is too low, evolution will be slow and can get stuck in local optima, and conversely if the mutation rate is too high, the population can move out of good areas, where the global maxima may be. Although not demonstrated here, a solution to this problem could be through using a variable mutation rate which has been explored by [5, 8].

This study will test the hypothesis that an effective general mutation rate is 1/L, as suggested by Muhelnbein, 1992 [7]. Genetic algorithms are inspired by the mechanisms underlying evolution, such as mutation, reproduction and the survival of the fittest, allowing them to be useful in solving a wide range of practical problems. Central to this are the notions of heredity and variability, meaning that offspring are roughly identical to their parents, and there is variation between generations, respectively.

# Methods

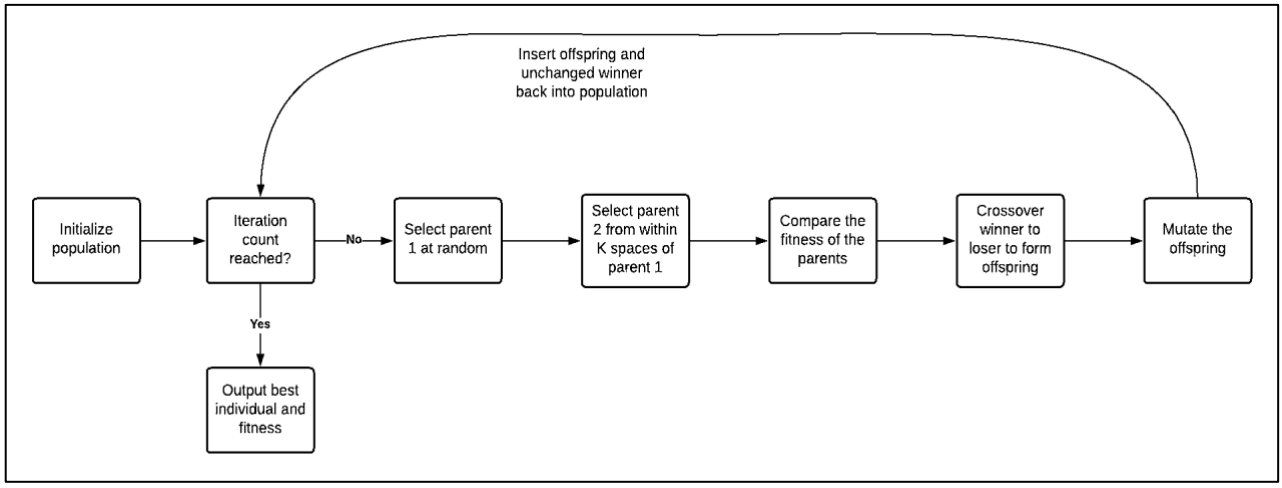



*Figure 1. Flow diagram showing the general steps of the microbial GA*

**Algorithm 1:** Microbial Genetic Algorithm

```
Result: Genotype with the highest fitness
initializatize random population P;
for each individual in P do
    assign each individual a position x in P;
end
for each tournament do
    for each genotype in P do
        g1 = random individual in P;
        g2 = random individual within 5 spaces of g1;
        winner, loser = compare(g1, g2);
        loser = crossover(winner, loser);
        loser = mutate(loser)
    end
    update population;
end
return the fittest individual;
```

*Figure 2. Pseudocode for the microbial GA used to solve this Knapsack Problem*

## Population

A population size of 100 is used for all iterations, and each genotype is generated randomly with a length of 30 genes in binary.

## Genotype to Phenotype Mapping

In our knapsack problem, the genotypes are binary arrays where 1 encodes for the presence of the item in the bag, and 0 codes for the exclusion of the item. The phenotype here is the fitness of the genotype in solving the knapsack problem.

Fitness function

**Equation 2. Fitness function for the GA**

$$Fitness = \sum_{i=1}^{n} w_i x_i \quad if \sum_{i=1}^{n} w_i x_i \leq W ,$$

$$Fitness = 0.01 \sum_{i=1}^{n} w_i x_i \quad otherwise$$

In order to evaluate the suitability of a genotype for solving the knapsack problem, we have designed the following fitness function.

The fitness of each genotype is equal to the sum of the selected items benefits. In order to make a smoother function, instead of assigning genotypes which exceed the threshold 0 fitness, we have penalised them by multiplying by 0.01. A smooth function is more likely to reduce the number of local optima so that the global optima can be more easily identified.

Selection

We have used a variant of tournament selection to select individuals from the population, with a tournament size of 2. Tournament selection is used as a way of controlling selection pressures [4]. We have used 500 tournaments for the iterations of the GA, to ensure that the fitness converges even when tested at low mutation rates. The first parent is selected at random, and the second is selected randomly from within a number of spaces in the population array of the first, called a 'deme' [3]. For the purpose of the knapsack problem, we chose the deme size to be 5. Our algorithm also exhibits elitism, as the winner from each comparison is returned the population unchanged.

Sexual Recombination

Our algorithm also incorporates sexual recombination, as the loser
Using crossover is an effective way of exploring the search scape as individuals are able to "share" information about the fitness landscape (schema theorem)

Mutation

Mutation rate has a general rule that it should be 1/chromosome length. The chromosome length used here is 30 and so according to this heuristic, the mutation rate should be 0.03 giving an average of 1 mutation per genotype.

Crossover

Crossover is an essential operator in a genetic algorithm, and involves choosing a random locus to exchange the sub-sequences before and after that locus between two genotypes to create offspring [2]. The microbial GA outlined uses a crossover probability of 0.5, so any gene in the in the loser has a 50% probability of being over copied by the winner's gene. We also use horizontal gene transmission, allowing faster evolution of the population and exploration of the fitness landscape.
The algorithm is also a 'steady-state' GA, which means that only one new offspring is produced and replaced for each iteration of the generation.

Replacement

In contrast to a generational GA, for this problem we have used a 'Steady State GA', where for P iterations, one individual is replaced by a single other individual. [3] Specifically, the individual with lower fitness – the 'loser' – is replaced by the offspring, which has undergone crossover and mutation. This operation can be described as "Evolution without Death" [3].

To evaluate the genetic algorithm, we have used 10 runs of the microbial GA for each of the following mutation rates: 0.001, 0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.8, 1.0 in order to gain an in depth understanding of the effect of mutation rate on the performance of a microbial genetic algorithm in solving the Knapsack Problem. For each of the runs, a different random seed is used to generate the population of genotypes. The results from these experiments are outlined in the following section.

## Results

In this section, we outline the results from the experiments on different mutation rates. We used 20 runs of the genetic algorithm for each of the 9 mutation rates. In the appendices you can see the results for the experiments with 10 runs.

As seen in Figure 3, mutation rates are essential for finding the solution to the knapsack problem, because otherwise there is no diversity in the population and the fitness landscape is not explored. It is therefore a problem of finding the right mutation rate to solve a problem.

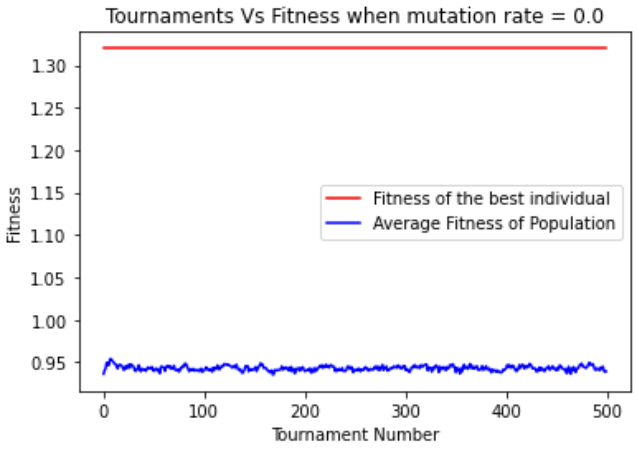



*Figure 3. Fitness over Tournaments without mutations*

Figure 4 shows how the fitness of the best individual changes over the tournaments and shows that a mutation rate of 0.2 and 0.8 both result in the highest fitness. Conversely 0.01 and 0.03 produce the lowest fitness. This data directly contradicts the heuristic that 1/L, therefore 0.03, would be the optimum mutation rate for a genetic algorithm. However, these data were collected from only a single run of the GA for 500 tournaments so do not provide complete insight into the effects of mutation rate on performance.

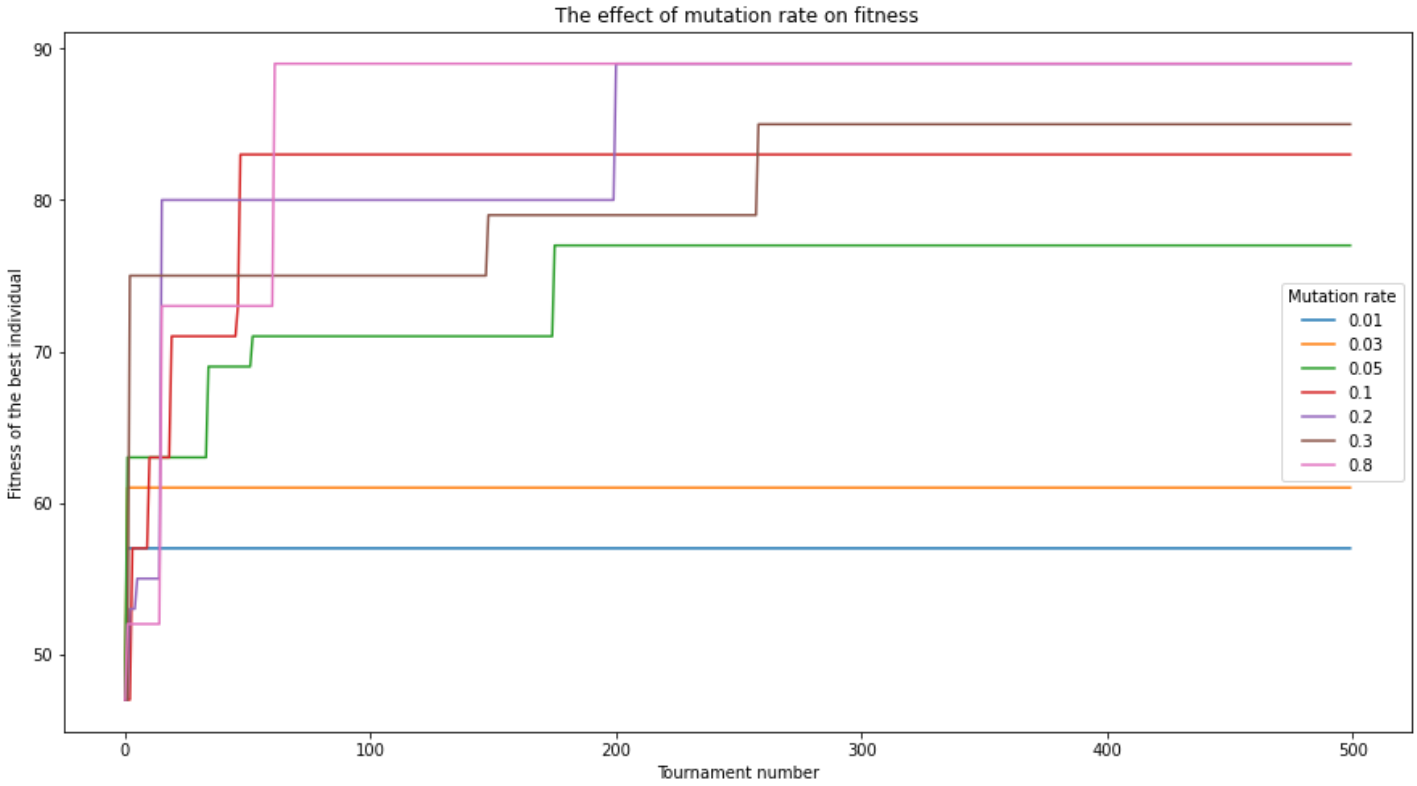



*Figure 4. Fitness vs. Tournaments with different mutation rates*

Figure 5 shows the distribution of fitness over the different mutation rates. For each boxplot I have also incorporated notches, which allows us to identify with confidence if the medians of two boxplots are different. The notch describes a 95% confidence interval that the difference in median is statistically significant. Therefore, as seen in Figure 5, we can say with 95% confidence that there is no statistically significant difference between choosing a mutation rate of 0.2, 0.3, 0.8 or 1.0 as the medians, notches and IQR overlap. The distribution of best fitness when the mutation rate is 0.001, 0.01, and 0.03 shows a markedly reduced best fitness. This provides more evidence against the 1/L rule for mutation rate.

To evaluate if this overlap is significant, a student's t-test has been used. The results of the t-test are as follows and given to 2sf. A t-test between data for 0.1 and 0.2 produced the values: T=3.5, p=0.00 suggesting there is no significant difference between the medians for this data.

Figure 6 visualises the mean and standard deviation for the 20 runs and demonstrates a clear distinction between the performance of the microbial genetic algorithm between 0.03 and 0.05. In order to investigate this difference further, we performed a series of experiments using smaller intervals between the mutation rates: 0.03, 0.033, 0.04, 0.045, 0.05.

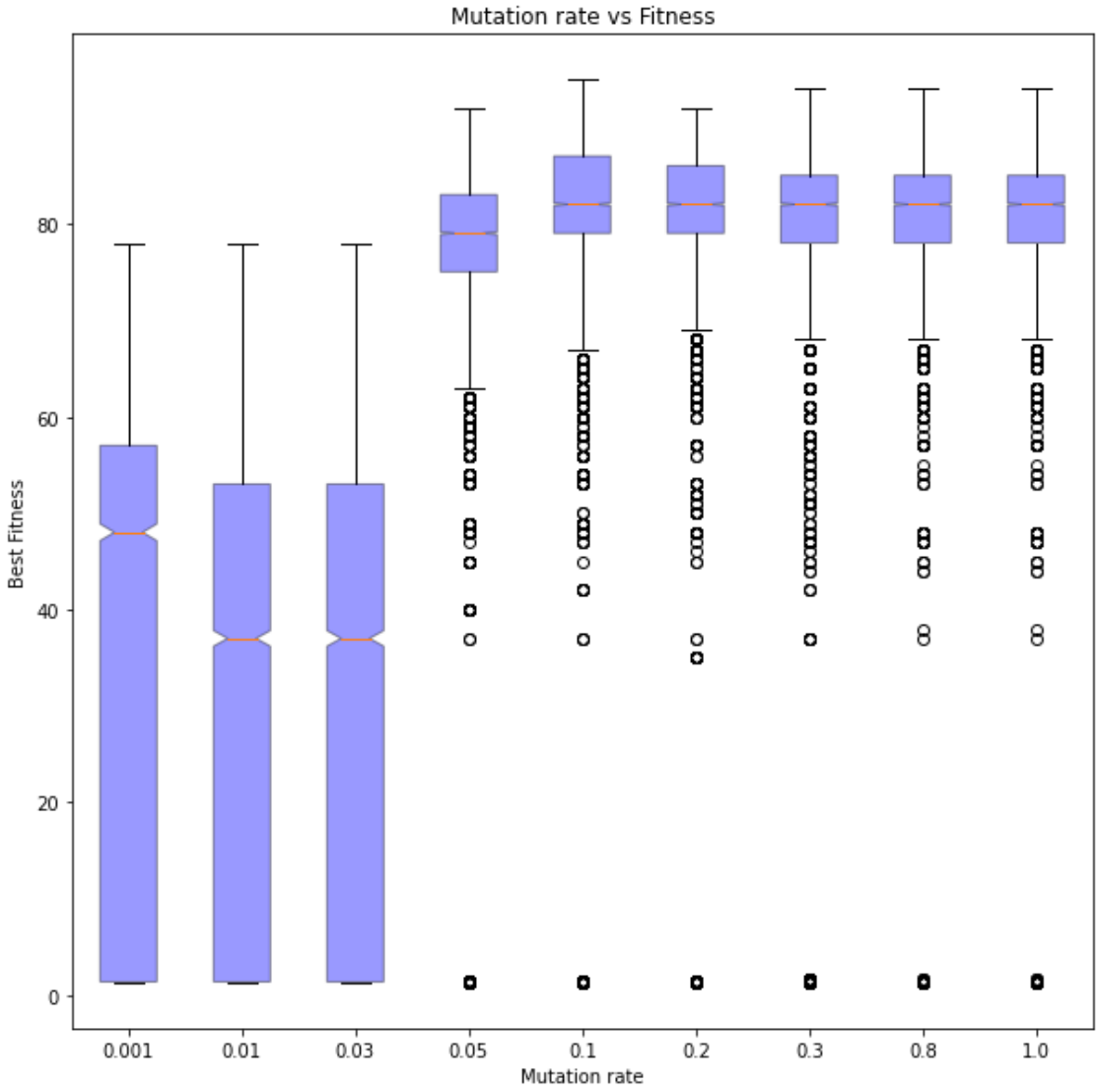



*Figure 5. Boxplot of the distribution of fitnesses over different mutation rates using 20 runs of 500 tournaments each.*

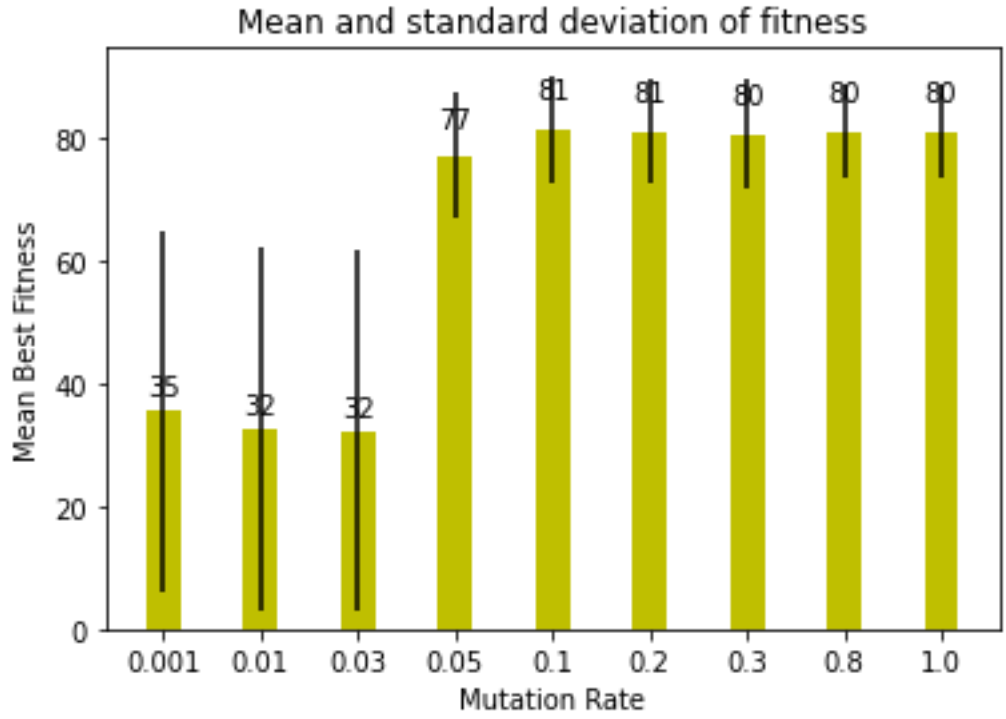



*Figure 6. Barplot of the mean and standard deviations of the best fitness using different mutation rates over 20 runs of 500 tournaments each.*

## Analysis and Discussion

We have therefore seen that mutation rate is an incredibly important factor in choosing the operators of a microbial genetic algorithm. This study has highlighted the requirement for a balanced mutation rate, which supports both exploitation and exploration of the fitness landscape in order to arrive at the global optima. Our results have also contradicted the heuristic that 1/L is an optimal mutation rate, as this produced the worst performance in our study.

It could be that the lower mutation rates will also converge to the higher fitnesses, but require a considerably higher number of tournaments to do so, or a variation of the other hyper-parameters chosen for this study.

Although we have studied in detail the effect of different mutation rates on the performance of a microbial genetic algorithm, a more important inquiry could be into the interaction of these different mutation rates with other hyperparameters, such as crossover rate, type of mutation, number of individuals in a population or any other choice that is made in designing a GA.

We would recommend the experimentation of a variable mutation rate in future studies, which is initially higher to encourage diversity in the population and then decreases through the tournaments, to encourage convergence on a global optimum.

## Summary and Conclusions

In summary, the quality of the solution reached by a genetic algorithm is strongly dependent on the selected mutation rate. A high mutation rate will increase the diversity of the population, supporting the exploration of the lands. Although previous evidence suggests that high mutation rates can also prevent the population from converging on the global optimum, this has not been observed in our study. It is therefore important to remember that like other operators selected for genetic algorithms, mutation rate should be tailored to the specific problem. In the case of the Knapsack problem, a mutation rate between 0.1 and 0.8 produces mostly equal performance in terms of the best fitness.

## References and Bibliography


1. Wright, Sewall (1932). "The roles of mutation, inbreeding, crossbreeding, and selection in evolution" . *Proceedings of the Sixth International Congress on Genetics*. **1** (8): 355–66

2. Mitchell, M. (1998). An Introduction to Genetic Algorithms. Massachusettss: The MIT Press.

3. Harvey, Inman. (199). The Microbial Genetic Algorithm.

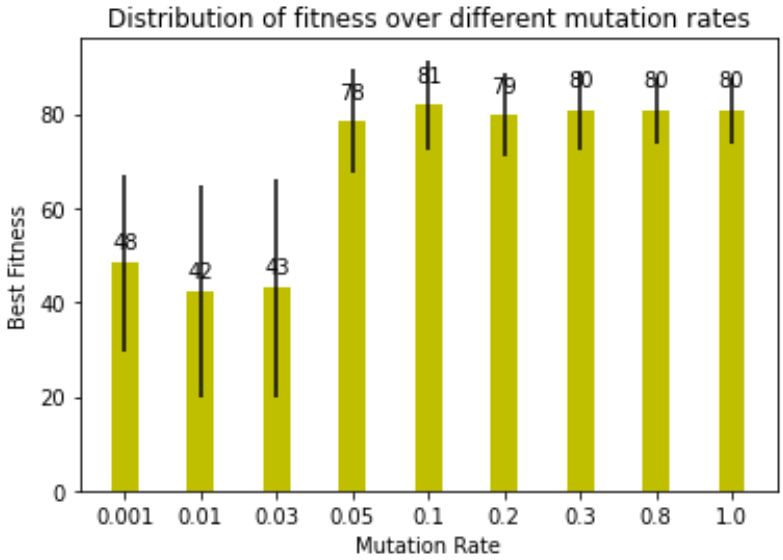
4. Ochoa, Gabriela. (2002). Setting The Mutation Rate: Scope And Limitations Of The 1/L Heuristic.. 495-502.

5. Pham, D. T. and Karaboga, D. (1997) 'Genetic algorithms with variable mutation rates: Application to fuzzy logic controller design', Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering, 211(2), pp. 157–167.

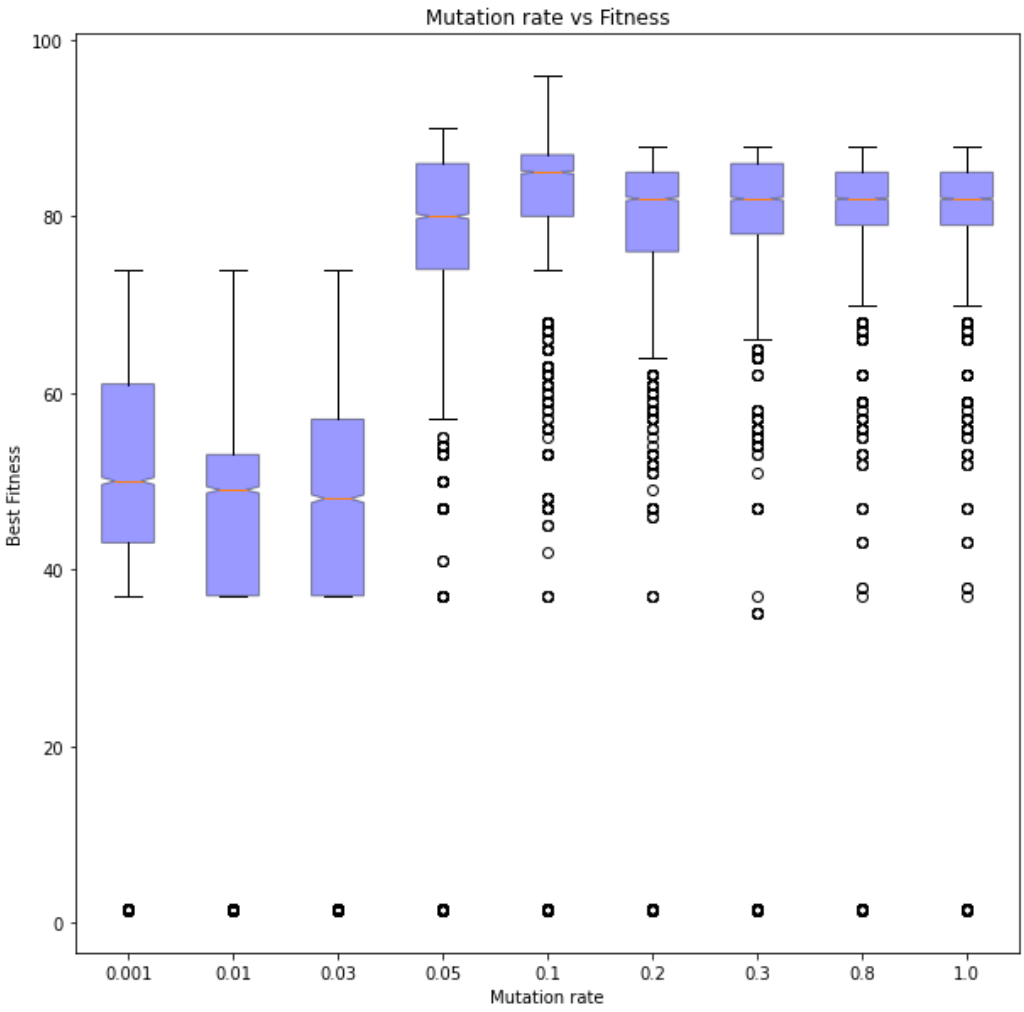
6. Ochoa, G., Harvey, I., and Bux-ton, H. (2000). Optimal mutation rates and selec-tion pressure in genetic algorithms. In Proceedingsof the Genetic and Evolutionary Computation Con-ference, pages 315–322.
7. Muhlenbein, H. (1992). How genetic algorithms really work: I. mutation and hill-climbing. In Manner, B. and Manderick, R., editors, Parallel Problem Solving from Nature 2. North-Holland.
8. Tuson, A. and Ross, P.(1998). Adapting operator settings in genetic al-gorithms. Evolutionary Computation, 6(2):161–184


## Appendices

1.



2.


Mutation rate vs Fitness
Best Fitness
Mutation rate
0
20
40
60
80
100
0.001
0.01
0.03
0.05
0.1
0.2
0.3
0.8
1.0