Examining the effects of changing microbial genetic algorithm parameters impact on fitness using the knapsack problem

215758
G6042: Acquired Intelligence & Adaptive Behaviour

May 28, 2021

Abstract

Microbial genetic algorithms explore a fitness landscape, these algorithms have several parameters for different aspects of the evolutionary process, such as crossover probability, mutation probability, and window size. The problem we will be solving is the knapsack problem, this is where there is a set of weights, corresponding benefits, and a max volume. We have to achieve the best benefit value without exceeding the max volume. In this study, we experiment with changing the parameters of the GA to see how it affects fitness. We found that parameters are the main driving force for the evolutionary aspects of this algorithm. Crossover affects the fitness drastically, to the point where it cannot escape past the max volume is too high. Window size is important, too large and it will reduce the learning rate, halving the fitness compared to a good value. Mutation probability has to not be too high or too low otherwise the growth is stunted too, but in-between the fitness is roughly the same. The results may be skewed as the size of the max volume may be hiding other patterns and this data could only be applicable for this volume size.

1 Introduction

1.1 The Knapsack problem

The knapsack problem can be described as having a set of items that hold different physical weights and benefits for taking the certain item, however, your knapsack can only hold so much weight, you need to figure out the best combinations of items to bring with you [5]. This problem can be solved in a multitude of ways, one of which is using a dynamic programming approach, which is computationally complex, and another which we will be looking at is using a genetic algorithm.

1.2 Microbial Genetic Algorithm

A genetic algorithm (GA) is an optimization technique to explore a fitness landscape, modeled after genotypes and evolution[4][6].

We will be using a microbial genetic algorithm, which is based on bacterial conjugation [3].

It starts with randomly generating a population of genotypes, which are lists of genes where a gene is a value of some sort. An example could be a real number or using binary encoding where only one and zero are used. They are then tested to find their fitness'.

Fitness is a measurement of how well a particular genotype is doing for the problem it is facing, it would be represented numerically, and the higher the value the better.

Selection generally is where two genotypes are selected, then their fitness' are evaluated and compared, finally returning the winner and loser. A particular selection method is called Demes, it acts like an island model[7], where there is a selection window where another genotype can be selected from, these are then compared and the remaining steps follow the general flow of a selection function.

Crossover is where a genotype transfers genes, depending on a probability for each individual, to another genotype. Mutation is similar to the aforementioned, however, there is no exchange, just a genotype's genes depending on some probability changing to a different value such as switching from one to zero or visa versa. These two probabilities can change the fitness drastically depending on the probability as it essentially relocates them into another position on the fitness landscape. Overall changing the parameters can give better performance [1].

1.3 Our Examination

Our examination will be analyzing the amount of tournament versus fitness, the change of fitness after x tournaments while incriminating the probability of crossover and mutation separately as well as doing the same but for the window size.

We have the default values for each parameter as $p\_crossover = 0.01$, $p\_mutate = 0.15$, $window = 3$.

We predict that crossover probability will stagnate at just over around the beginning of the first quartile due to not retaining the good genes and completely relocating them into an area with much lower fitness and due to them losing over and over, the crossover will be used still once picked making them stay in a low level. The mutation probability should increase the fitness but in the higher ranges it will drop from replacing the good gene choices due to it exploring the current area in the fitness landscape but too much will cause jumps into a whole new location. Lastly, the window would roughly stay high in fitness until a point within the window size, at half or just over, where it will drop and stay at a lower level than before.

2 Methods

We approached the Knapsack Problem by implementing it through the following; We used two arrays, with the length of forty to try and make it more difficult for the GA to evolve into having good fitness. The first array, ***volumes***, was to store the weight of the item at each position of the index. There were correlating benefits for these different volumes in another array called ***benefits***. The volumes and benefits were generated using random numbers picked between and including one and ten. The knapsack problem also needs a maximum volume which can be derived from the total benefit and making it smaller to some degree. We decided to divide the total by half and then remove more away from the half until it's at a size that is not impossible but not easy either.

The genomes are structured as a binary encoded list the same size as the other two arrays mentioned in the previous subsection, each individual in this list would act as a gene. Representing taking an item, would be a one in the index of the item which you want to talk, else you would use zero.

To calculate the fitness, seen in algorithm 1, we take the input of a genotype and calculates the volumes and benefits depending on the positions where the number one appears. If the max volume exceeds the volume limit, it returns the benefits but at one hundred times less; this is to encourage to achieve higher fitness' over evolution. Otherwise, it just returns the sum of the benefits.

We implemented the microbial GA as discussed in the introduction using Demes selection which helps with maintaining diversity [2]. This can be seen in algorithm 2. Using this algorithm we will adjust the crossover probability, mutation probability, and window size to examine its consequences. We will run each adjusted value ten times to find the mean, then adjust it to be slightly larger and run until it has reached the maximum it can be.

Crossover probability will affect the chance an individual in a genotype is copied from one genotype into another. Mutation probability is the same but for the chance for an individual to be switched from 0 to 1 or visa versa. Lastly changing window size changes the number of genotypes that could be selected for the Deme tournament.

Crossover and mutation probabilities will jump in the value of 5% from 0% to 100%, while the window size will iterate using a value of 1, from 0 until the genotype length, which is forty.

3 Results

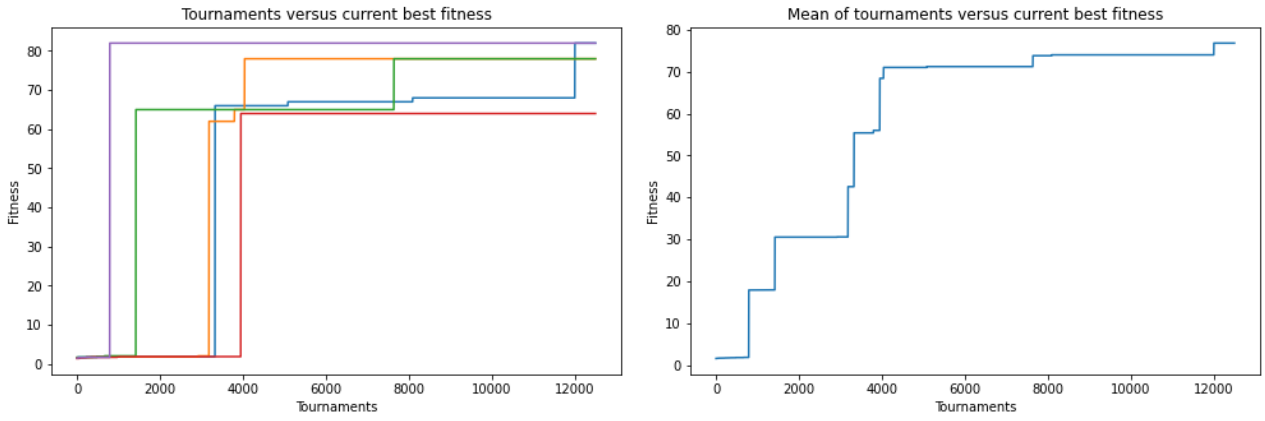

Figure 1: Amount of tournaments verses fitness.

First, Figure 1 shows five different runs of 1200+ tournaments versus current best fitness's showing how it evolves over time and the mean of the five different runs. It seems that the peak fitness is just over 80, as two of the runs converge to the same point in the later stages of the tourneys. Most of the development happens within the first 4000 tournaments where they jump by a large fitness as they have finally found a

solution that doesn't exceed the max volume which then gives them a solid foothold to evolve from. From here the genotypes either stagnate as they found the best fitness or slowly evolve at a reduced rate.

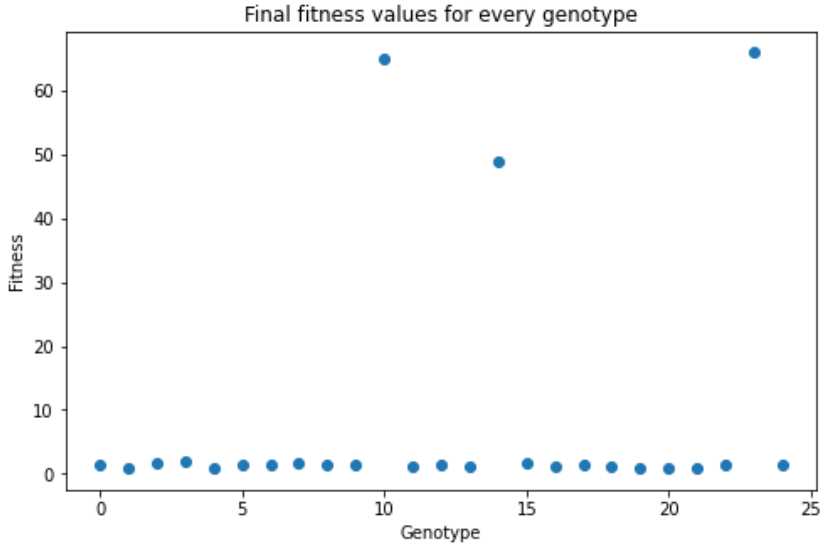

Figure 2: End fitness' of running the GA over 1200+ tourneys

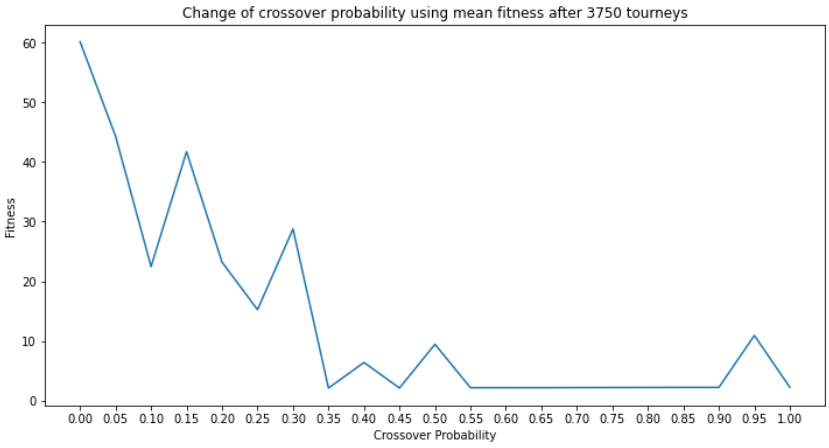

Figure 3: Change of crossover probability size effecting fitness

Next, Figure 3, initially it starts with high fitness at a low level of crossover, and quickly reduces until it flat-lines at 55%. As mutation is still being used, this may explain the higher fitness' early on however it seems the more crossover there is, it reduces the overall fitness due to the starting population most likely being at lower values which when larger crossover happens it may overwrite the progress which is happening due to the mutations happening. We can see in figure 2 that most of the population has rather low fitness and only a few made it out. Comparing this with the previous figure shows that the crossover affects the results drastically showing its importance.

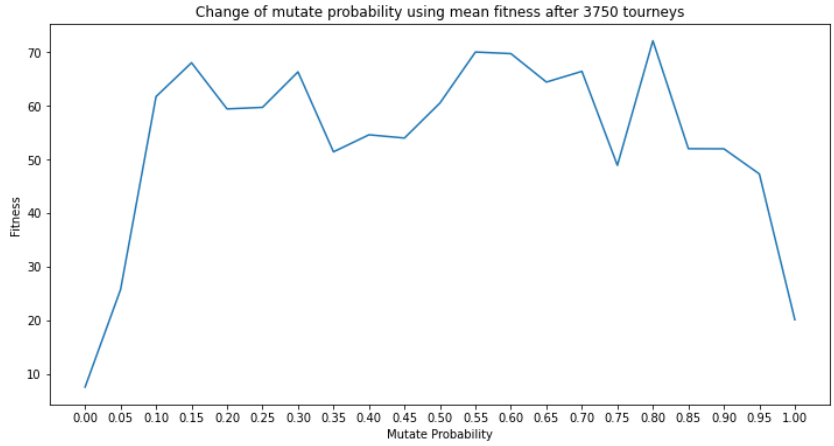

Figure 4: Change of mutation probability size effecting fitness

In Figure 4 it shows that using a mutation rate that is higher than 15% causes a stable higher fitness compared to figure 3, in the later stages, there are more dips until it completely drops from 80% onward. At 100% it drops to a sub-par level as no changes take hold but is higher than the figure using the same probabilities 3, but crossover will affect the fitness scores which may have contributed to the ≈ 20 fitness. Having both at the same time seems to help with increasing fitness and keeps hold of the progress exploring the fitness landscape, but must not be too high otherwise, the progress will be lost, gained very slowly, or even incapable of finding a foothold. It seems to get the best results you just need to have a number that isn't too high or too low, it doesn't drastically hurt the fitness outside of that range. It's important but doesn't affect the results as much crossover.

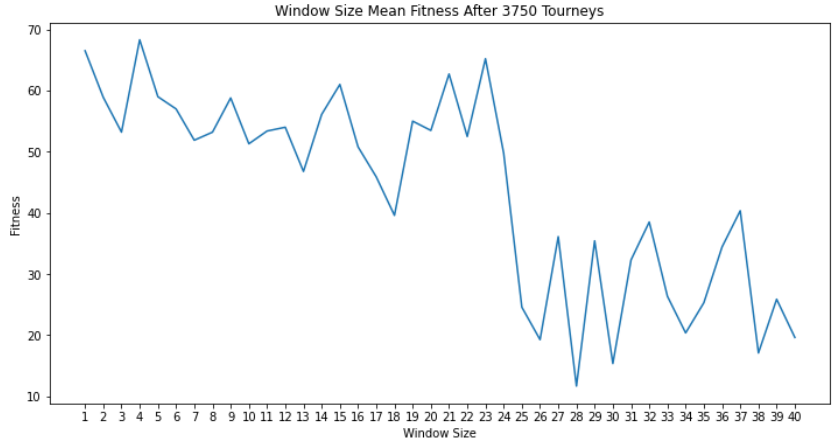

Figure 5: Change of window size effecting fitness

Finally, Figure 5 has a similar structure to figure 3, it fluctuates a lot throughout the graph but there is a notable decline nearer the end which never makes it past halfway the height of the taller section. There only really is a negative effect if you make the window size just over half the size of the genotype length. The fitness' seems to be just below what figure 4 was like even though the mutation used for the window size was in the range where it peaks, the window size reshaped the results drastically.

4 Discussion

Parameters are the main driving force for the evolutionary aspects of this algorithm. If they are incorrectly assigned the GA will fail to perform for this task, even not being able to break out of the max volume for 3750 tourneys.

Each of the three examined parameters has different amounts of effect on the resulting fitness. The parameter which most affected the results was the crossover probability, anywhere larger than 5% causes the GA to stagnate with the originally generated population, it can completely kill off the population getting outside of its current fitness levels. The window size is next, where it's pretty stable for the most part until larger sizes where it cannot recover back to the same levels as to where it was. Lastly, the mutation does

help a lot, but only if it's not within the first or last 10% range, it helps with the general evolution of the GA.

Next time I will find a baseline to compare the results to, to see how well it performs versus one with the 'default' values as it can show how well each is doing compared to it to gain some more meaning from the figures. I would also try changing the max volume which can be taken as it may allow more flexibility for the GA to grow and more patterns may appear.

Some results may be skewed due to the initial population size may have generated an optimum solution instantly by sheer luck, but not having these randomly set fitness landscape locations means that we won't get accurate results either as the population needs to start out diverse to be able to find the best fitness as if they are all clumped together they will only be able to find the local maxima unless we were to adjust the mutation rate to be high to compensate to cause big jumps.

These results may only be applied to the knapsack problem, as there are also problems that are not just binary encoded which would give more possibilities to explore.

More experimenting needs to be done to truly see if this is the overall pattern by using different benefits, volumes, max_volume, and other variables which may cause overfitting of the results to be just for this max volume size.

5 Conclusion

Parameters are the main driving force for the evolutionary aspects of this algorithm. Crossover can stagnate a population if too large, the window size will decrease the potential of the GA but it will still break out of exceeding the max volume, and finally, mutation helps with the general progress, if it's too low or too high it can cause the progress to massively halt however like the window size it can still break out of exceeding the max volume. This can only be semi-reliable to be used for this problem, and the reason why it can only be 'semi' is due to factors such as the size of the max volume which may be hiding other patterns and this data could only be applicable for this size.

6 References

[1] T. Amudha and B. L. Shivakumar. Parameter optimization in genetic algorithm and its impact on scheduling solutions. In Lakhmi C. Jain, Himansu Sekhar Behera, Jyotsna Kumar Mandal, and Durga Prasad Mohapatra, editors, *Computational Intelligence in Data Mining - Volume 1*, pages 469–477, New Delhi, 2015. Springer India.

[2] Edmund Burke, Steven Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions on*, 8:47 – 62, 03 2004.

[3] Inman Harvey. The microbial genetic algorithm. In *Proceedings of the 10th European Conference on Advances in Artificial Life: Darwin Meets von Neumann - Volume Part II*, ECAL'09, page 126–133, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] John H. Holland. Genetic algorithms. *Scientific American*, 267(1):66–73, 1992.

[5] Tribikram Pradhan, Akash Israni, and Manish Sharma. Solving the 0–1 knapsack problem using genetic algorithm and rough set theory. In *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, pages 1120–1125, 2014.

[6] Colin R. Reeves. Fitness landscapes and evolutionary algorithms. In Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, Marc Schoenauer, and Edmund Ronald, editors, *Artificial Evolution*, pages 3–20, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[7] Darrell Whitley, Soraya Rana, and Robert Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7, 12 1998.

7 Appendix

Algorithm 1 Calculates fitness of a genotype

Require: $gene, benefits, volumes, max\_volume$
  $v = 0$
  $b = 0$
  $fitness = int(0)$
  for i in range(gene.shape[0] **do**
    if $gene[i]$ **then**
      $b \mathrel{+}= benefits[i]$
      $v \mathrel{+}= volumes[i]$
    end if
  end for
  if $v > max\_volume$ **then**
    $fitness = b * 0.01$
  else
    $fitness = b$
  end if
  $returnfitness$

Algorithm 2 Microbial GA

Require: $window, p\_crossover, p\_mutate, pop\_size, epochs, benefits, volumes, max\_volume$
  $genos = generate\_population(pop_size, individual\_size)$
  $geno\_fitnesses = np.zeros((pop\_size, epochs))$
  $current\_fitnesses = get\_population\_fitness(genos, pop\_size, benefits, volumes, max\_volume)$

  for epoch in range(epochs) **do**
    for gene in range(pop_size) **do**
      $winner\_id, loser\_id = demes\_tourney\_select(pop_size, current\_fitnesses, window)$

      $crossover\_loser = crossover\_function(genos[winner\_id], genos[loser\_id], p\_crossover)$
      $mutated\_loser = mutation\_function(crossover\_loser, p\_mutate)$

      $genos[loser\_id] = mutated\_loser$
      $current\_fitnesses[loser\_id] = fitness\_function(mutated\_loser, benefits, volumes, max\_volume)$
    end for
    $geno\_fitnesses[:, epoch] = current\_fitnesses$
  end for