# TreeGen: Finding Optimal Tree Placement for Survival

**August 2, 2019**

Kristian Darlington · Rhett Devlin · Austin Lee

# Table of Contents

## List of Figures

## Abstract

The TreeGen team is composed of three software engineering students at the University of Victoria - Kristian Darlington, Rhett Devlin, and Austin Lee - who were brought together in their fourth year artificial intelligence class to utilise genetic algorithms to solve a real-world problem. Common interests in software and environmental sustainability led them to the idea addressed by TreeGen: finding the optimal tree planting configuration for a given environment. The genetic algorithm used in the TreeGen project searches the realm of all possible tree planting configurations for a given environment and, using a digital encoding of this environment that involves two dimensional spatial representations of ground layouts and environmental factors like sunlight levels, soil compatibility, and water availability, attempts to find the best solution available that maximizes the number of trees planted and minimizes the competition for resources between them. To determine whether resulting tree configurations are optimal, variables for the genetic algorithm like termination conditions, crossover frequency, and mutation frequency are varied and results determined by each variation are compared.

# 1. Introduction

The team involved in the TreeGen project is composed of three Canadian undergrad students from the University of Victoria studying Software Engineering; Kristian Darlington, Rhett Devlin, and Austin Lee.

## 1.1 Background

Kristian, Rhett, and Austin were brought together during their fourth year Artificial Intelligence class, instructed by Professor Zahra Nikdel, upon the assignment of a genetic algorithm project to be tackled in groups of three.

The TreeGen team, currently all in their 20s, was raised in a world just beginning to process the advent of major climate change. Living in Canada, the importance of sustainable living and environmental conservation was enforced continuously - this came in the form of elementary and secondary education, mass media, and family values. As products of their developmental environments, Kristian, Rhett, and Austin hoped to combine their engineering skills, knowledge of artificial intelligence techniques, and interest in sustainable practices to produce a prototype genetic algorithm that may have practical value upon future development and refinement.

## 1.2 Motivation

With climate change an undeniable fact in the 21st century, it is imperative that humanity aids the recovery of forest ecosystems when possible. Events like forest fires and mass wasting, as well as destructive human activities like logging, reduce the Earth's long-term carbon storage and production of oxygen. While current methods of reforestation are a step in the right direction, it is possible to improve the process through the prevention of undue tree death, and therefore waste, due to competition for resources between trees through careful evaluation of the landscape in question.

The evaluation of a landscape for the purposes of reforestation is an appropriate problem for genetic algorithms; a machine has the ability to simulate numerous tree-planting configurations, evaluate tree survival given certain constraints, and make conclusions based on results in a significantly shorter span of time than any human could. Genetic algorithms aid this process in particular by repetitively performing simulations on different tree configurations and interweaving the best results numerous times.

The goal of the TreeGen project is to provide optimal tree planting configurations for the survival of trees after reforestation efforts given an input landscape and associated attributes like soil type, water retainability, and sun availability, combined with tree species and associated growth requirements.

# 2. Related Work

Research into other efforts similar to the TreeGen project yielded two projects that investigated the idea of optimal tree placements for survival. The first is a metaheuristic optimization algorithm, designed by S. Aykol and B. Alatas that compares nine different groups of metaheuristics to find an optimal solution to map an organization of the desired plant based on its "intelligence" [1]. It compares its algorithm to similar other metaheuristic methods that use genetic algorithms, but ultimately does not use genetic algorithms to solve the problem.

A second project that relates to the optimality of tree planting is the Forest Optimization Algorithm (FOA) developed by M. Ghaemi and M. Feizi-Derakhshi [2]. This algorithm takes in meta-data relating to the types of trees being planted and evaluates the spread between each tree needed. FOA does not make use of genetic algorithms and, when compared to genetic algorithms, claims to take less evaluations to find a solution. It functions by guessing the starting positions of trees and evaluating the layout.

# 3. Problem Formulation

The following sections discuss the techniques used to convert a real-world problem, the optimal configuration for tree planting given some environment, into a digital format that can be interpreted and acted upon by a genetic algorithm.

## 3.1 Search Space

Given that the problem being addressed by TreeGen is finding an optimal tree planting configuration for a given environment, the search space involves all possible configurations of trees in the environment. The number of trees involved in any given solution can range from a minimum of zero to a maximum determined by the amount of space available.

“Optimality,” in the context of this algorithm, looks at the ratio between two different values over the entire environment given a tree configuration; these values are the number of trees planted, or tree-count, and  “competition”. High competition in an environment is a primary indicator of future tree death due to the lack of resources available for the trees to prosper - keep in mind that different tree species require different resources and amounts of those resources, therefore the values determining competition for a configuration will differ with the species being “planted.” Due to the end-goal for a run of the algorithm being the planting of as many trees as possible while minimizing tree death, a higher tree-count to competition ratio signifies a better solution in the search space for the environment than a solution with a lower tree-count to competition ratio.

## 3.2 Encoding

The encoding of an environment for this algorithm takes the form of a two dimensional array, or matrix, which was determined to be the most efficient digital representation of a real-world ground layout when elevation (which is not considered in this project thus far) is not involved. To visualize the encoding, one can imagine looking directly at the ground from above and dividing up sections of ground by placing a grid of squares overtop.

The real-world area represented by each space in the matrix is crucial to providing a realistic representation of tree placements. Two clear options were considered for the basis of this value: (1) the approximate cross-sectional trunk base area of a tree when that tree is mature and (2) the approximate cross-sectional crown area of the tree when that tree is mature. Due to increased variance in crown development (in terms of shape and size) than in trunk development, it was determined that the girth of a tree's trunk would be easier to manage for the purposes of this encoding. The method used throughout history to place a value on the space taken up by a tree trunk is to use a measurement of the girth of the trunk at approximately "1.5 metres from ground level" [3]. To maintain simplicity for the project, trees with girths of approximately "18 to 30 inches" (0.5 to 0.8 metres) at maturity for a cross-sectional trunk base area of approximately 0.034 square metres on a plane perpendicular to the direction of growth, like the red maple tree, will be considered (see Figure 1).

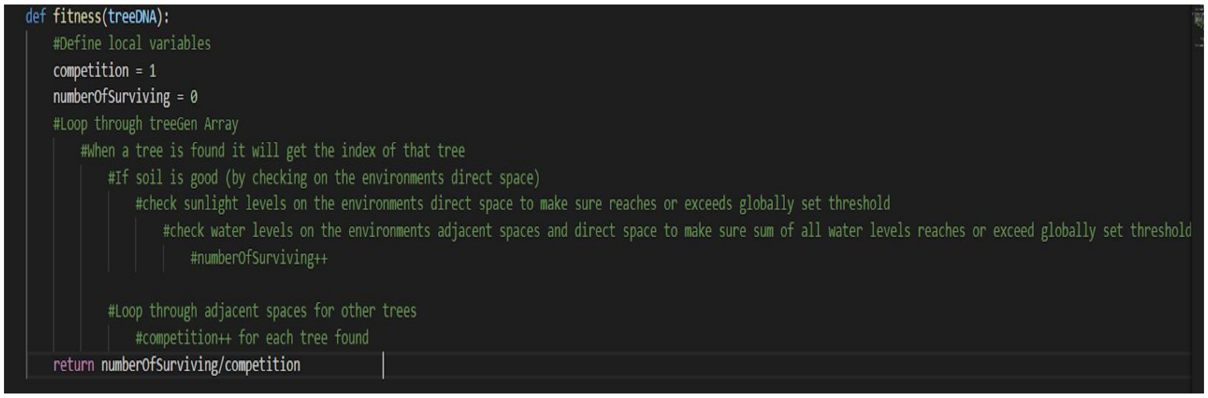
Figure 1: A photo of a red maple tree.

A satisfactory amount of ground space for this size of tree to grow can be estimated at 1 square metre, therefore each space in the matrix encoding of a tree configuration represents a real area of 1 square metre.

Each individual in a generation takes the form of a single matrix with the specifications discussed above. A space in this matrix is labelled as “True” if it contains a tree and “False” if it does not. The environment that tree configurations will be placed on and compared against takes the form of a single matrix where each space is assigned three numeric values. These values represent sunlight level, soil compatibility, and water availability. For a run of the genetic algorithm, thresholds are specified for each of these values for each tree species being planted and these thresholds are compared against the values assigned to the environment spaces.

## 3.3 Fitness Function

The fitness function performs evaluations using a ratio between the number of surviving trees and competition in the environment, taking the matrix encoding of the tree configuration as input. The function begins by iterating over each space in the matrix to find trees. For each tree, it is ensured that the necessary resources - soil compatibility, sunlight level and water availability - are present in its location for the tree’s immediate survival. The values for these resources are compared against threshold values determined by the species of tree in question; if any value is below the corresponding threshold, the tree is not considered in the surviving tree count. A further check for surrounding water in adjacent spaces is performed and counted towards the total available water for the tree before checking the water threshold value. If the resource levels are sufficient, the value for total surviving trees is increased by one. To determine the value for competition, adjacent spaces to each tree is checked for other trees. Each other tree found increases the total competition value. The final value for surviving trees is divided by the final value for competition for the ratio, which is then returned by the fitness function (see Figure 2).



```
def fitness(treeDNA):
    #Define local variables
    competition = 1
    numberOfSurviving = 0
    #Loop through treeGen Array
        #When a tree is found it will get the index of that tree
            #If soil is good (by checking on the environments direct space)
                #check sunlight levels on the environments direct space to make sure reaches or exceeds globally set threshold
                    #check water levels on the environments adjacent spaces and direct space to make sure sum of all water levels reaches or exceed globally set threshold
                        #numberOfSurviving++

            #Loop through adjacent spaces for other trees
                #competition++ for each tree found
    return numberOfSurviving/competition
```

Figure 2: Python pseudocode of the fitness function.

## 4. Methodology

The following section describes methods and goals for determining when the genetic algorithm finishes, comparison between populations, the environment changes between runs of the algorithm, comparison of results of different runs of the algorithm, and the conclusivity of the final solution provided by a run.

### 4.1 Termination Condition

To determine when the genetic algorithm should be terminated, a set of trial runs is completed. The trial runs of the algorithm are each terminated at a different number of generations. The number of generations used to terminate further runs of the algorithm is then set as the value where variance in final fitness values was minimized. Determining this value adds confidence in the optimality of final solutions provided by each run of the algorithm on an environment. After the trials, the number of generations needed to get a global maxima as opposed to a local maxima will be used as a set variable in the *treegen.py* file. If this method of narrowing down on a maximum generation limit does not suffice, another method can be considered in which the algorithm continuously checks for increases in the best fitness scores of populations at runtime - when fitness scores no longer increase for a number of generations, the algorithm will halt with confidence that the chosen solution is optimal.

## 4.2 Variable Changes

During trial runs of the genetic algorithm, variables such as crossover frequency, mutation frequency, and population size are set to different values for the purpose of evaluating resulting solutions for different inputs. Variables such as a limit on the number of generations, tree water level requirements, and tree sunlight requirements are assigned at the top of the *treegen.py* file and can also be adjusted. Additionally, the weighting given to the number of surviving trees and weighting given to competition between trees can be manually adjusted to manage fitness function evaluations. To test various crossover frequencies, mutation frequencies, and population sizes automatically, a file exists in the repository called *run_for_statistics.py*. This is used to automate trial runs of the algorithm and compare resulting solutions for a particular input map.

## 4.3 Comparison and Total Runs of the Algorithm

As specified above in section 4.1, the highest fitness score of each generation is maintained during a run of the algorithm. As such, each run of the algorithm is represented by its highest fitness score of the entire run. These fitness scores are then used to compare different runs of the algorithm on a given environment. In the additional *run_for_statistics.py* program, the amount of trials can be specified in the standard input. The output of the program is a text file that records the results of each trial with the highest fitness score listed for each for easy comparison. Evaluating the fitness scores and the respective genetic algorithm variables used to produce them provides insight into the balance between exploration and exploitation within the TreeGen search space.

## 4.5 Determining Optimality of Solution

When the algorithm stops running, there is a singular individual that is represented by the highest fitness score found. Because the algorithm tracks the maximum fitness score encountered amongst all generations in order to prevent acceptance of local maxima, there is confidence that the fitness score and corresponding tree configuration returned is an optimal solution for the provided input map.

## 5. Results and Discussions

With the genetic algorithm developed, it was initially hypothesized that numerous runs of the algorithm on the same map using different crossover and mutation frequencies would yield an optimal combination of these frequencies; the optimal combination would be shown by the highest fitness score produced during the runs. In the context of crossover and mutation frequencies, optimal means the best-performing balance between exploration and exploitation of candidate solutions within the TreeGen search space such that the algorithm eventually converges on the global maximum amongst fitness scores. It was thought that, despite the crossover and mutation frequencies being found using a single map, they would work efficiently for any map (different in resource configuration and/or size) and population size due to the problems being similar in nature.

To test this hypothesis, the same randomly-generated map was fed into the genetic algorithm 300 times, each with a different crossover and mutation frequency and with a fixed maximum generation limit of 1000 and population size of 30. The best performing combination of crossover frequency and mutation frequency was 54% and 12%, respectively.

To test the hypothesis described above, a different map yielding an approximate fitness score of 6.5 in a previous trial was provided to the genetic algorithm (see Figure 3).

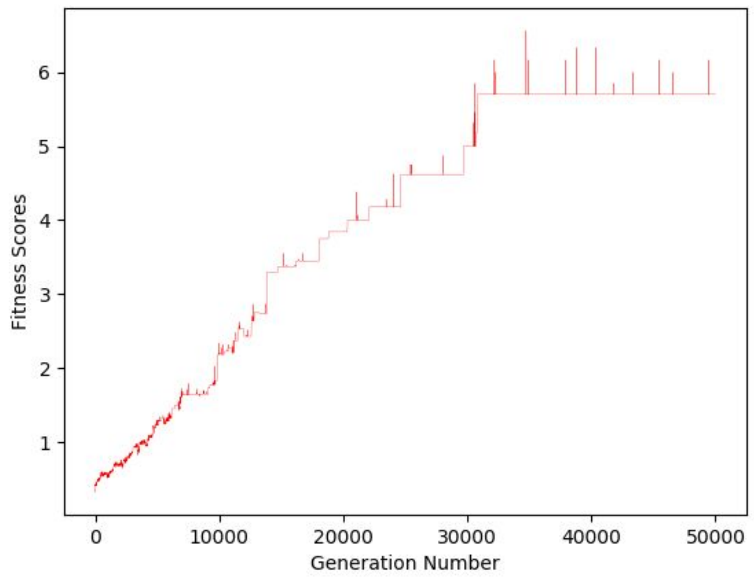


Figure 3: Trend in fitness scores using non-optimal crossover and mutation frequencies.

With the suspected "optimal" crossover frequency of 54% and mutation frequency of 12%, as well as after 50000 generations, the algorithm had hardly discovered a solution with a fitness score above 3.0 (see Figure 4).

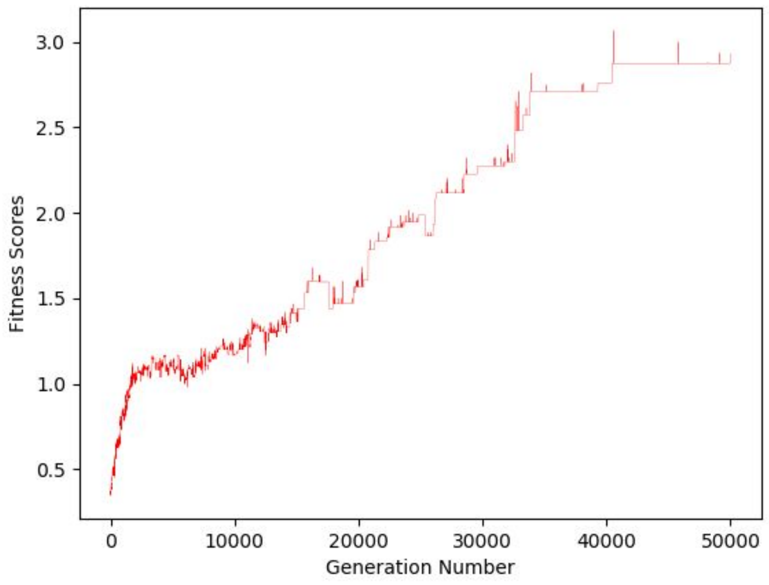
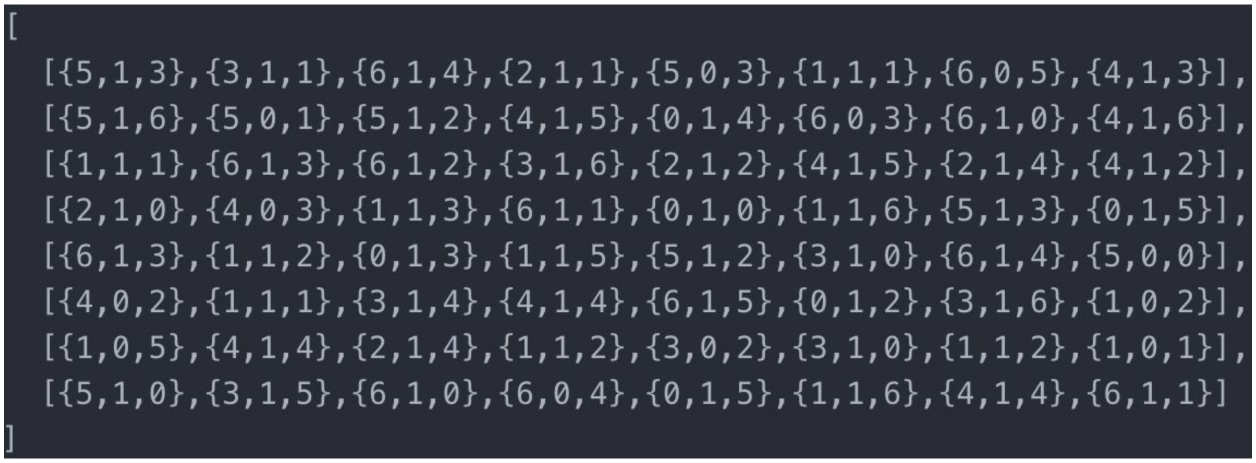


Figure 4: Trend in fitness scores using the "optimal" crossover and mutation frequencies.

The above result revealed that the optimal crossover and mutation frequency can differ greatly depending on the map, generation limit, and population size provided - not all maps can be solved efficiently using the optimal genetic algorithm variables for a single map, thus disproving the hypothesis.

To improve confidence in solution optimality and with the knowledge that no perfect crossover and mutation frequency pair exists for solving every map - at least, not one that can be discovered in this domain without extensive experience with genetic algorithms - the idea of a fixed generation limit was questioned. Instead of setting a limit on the generations manually at the beginning of a run, the algorithm could instead check the variance in the best fitness scores over the last number of generations and halt itself when the fitness score no longer improves over time - this implementation could be done at runtime. For the purposes of further experiments that use this method, the fitness score variance was checked manually by the person executing the program, but future development will include coding this functionality into the project.

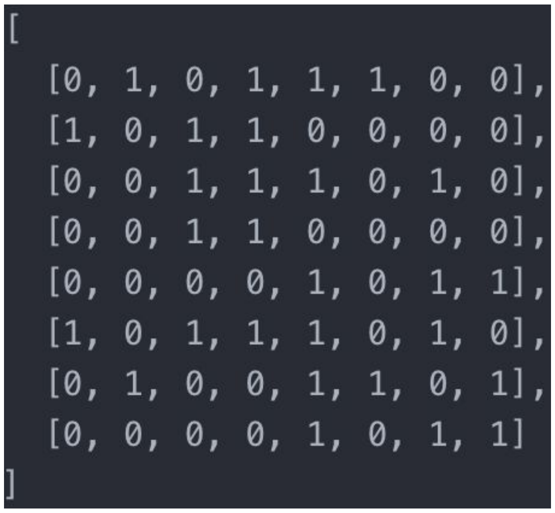
An example encoding of a real 8 by 8 map provided to the genetic algorithm is provided below, in which each space (marked by curly braces) contains a number representing water availability, soil compatibility, and sunlight level, respectively. The map was chosen for this report due to its relatively small size, as opposed to the 30 by 30 to 100 by 100 maps used during trial runs (see Figure 5).



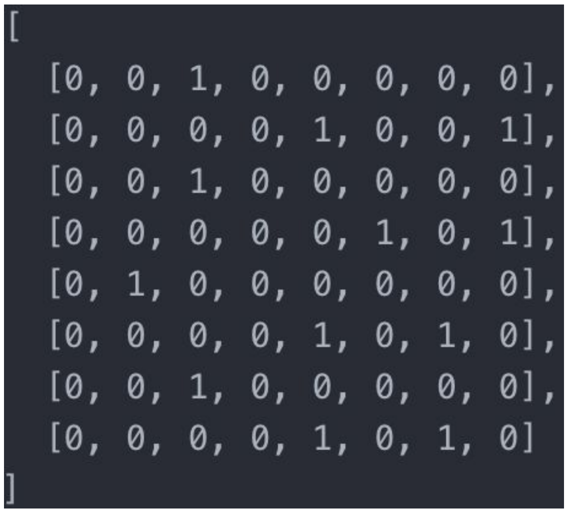
```
[
  [{5,1,3},{3,1,1},{6,1,4},{2,1,1},{5,0,3},{1,1,1},{6,0,5},{4,1,3}],
  [{5,1,6},{5,0,1},{5,1,2},{4,1,5},{0,1,4},{6,0,3},{6,1,0},{4,1,6}],
  [{1,1,1},{6,1,3},{6,1,2},{3,1,6},{2,1,2},{4,1,5},{2,1,4},{4,1,2}],
  [{2,1,0},{4,0,3},{1,1,3},{6,1,1},{0,1,0},{1,1,6},{5,1,3},{0,1,5}],
  [{6,1,3},{1,1,2},{0,1,3},{1,1,5},{5,1,2},{3,1,0},{6,1,4},{5,0,0}],
  [{4,0,2},{1,1,1},{3,1,4},{4,1,4},{6,1,5},{0,1,2},{3,1,6},{1,0,2}],
  [{1,0,5},{4,1,4},{2,1,4},{1,1,2},{3,0,2},{3,1,0},{1,1,2},{1,0,1}],
  [{5,1,0},{3,1,5},{6,1,0},{6,0,4},{0,1,5},{1,1,6},{4,1,4},{6,1,1}]
]
```

Figure 5: Encoding of an 8 by 8 TreeGen map with resource values.

The best solution within the first generation produced while running the genetic algorithm on the map above is located below on the left, in which a 1 indicates the presence of a tree in that spot and a 0 indicates the lack of a tree in that spot. It received a fitness score of 0.41. To the right, the optimal solution found by the algorithm with a fitness score of 13.5, is provided (see Figures 6 and 7).



```
[
  [0, 1, 0, 1, 1, 1, 0, 0],
  [1, 0, 1, 1, 0, 0, 0, 0],
  [0, 0, 1, 1, 1, 0, 1, 0],
  [0, 0, 1, 1, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 1],
  [1, 0, 1, 1, 1, 0, 1, 0],
  [0, 1, 0, 0, 1, 1, 0, 1],
  [0, 0, 0, 0, 1, 0, 1, 1]
]
```



```
[
  [0, 0, 1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 0, 1],
  [0, 0, 1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 1, 0, 1],
  [0, 1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 0],
  [0, 0, 1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 1, 0]
]
```

Figures 6 and 7: An early solution to the map in Figure 6 next to the optimal solution.

When comparing the early solution to the optimal solution, the differences in space between trees is easily seen, revealing the importance of managing competition for resources between trees within the algorithm. Less obvious is the fact that no tree in the optimal solution sits on a location with unsuitable soil or insufficient water and sunlight measurements (compare tree locations in Figure 6 to resource values in Figure 7 to see). As discussed above, confidence in the above solution's optimality was developed by watching the increase in fitness scores plateau and cease to improve over a number of generations - in this case, approximately 1000 generations elapsed with no improvement before halting the algorithm.

## 6. Conclusion

The genetic algorithm, TreeGen, is a collaboration project that determines the ideal placement of trees in an environment to maximize the number of surviving trees in the given environment. The environment is a two dimensional matrix of potential tree locations with randomized water availability, sunlight levels, and soil composition compatibility and the populations is a two dimensional matrix of matching locations where trees are randomly placed. The population and environment are then run against each other to determine which trees in that generation of placements survive based on each tree's location's available resources. Once a fitness score is calculated for that generation, determined by comparing the number of surviving trees to competing trees, the generation then has a chance of undergoing crossover and mutation to introduce potentially new solutions. The algorithm continues this process for a manually set number of generations based on the size of the environment and at the end, the highest fitness score achieved is returned along with the placement of trees that survived in the environment.

TreeGen's original estimation of using a 33% crossover chance and 5% mutation chance proved to be false. Through a number of trials on the same environments, it was determined that a crossover chance of 54% and a mutation chance of 12% allowed for a better chance that the genetic algorithm would find the max fitness of tree placement within the set amount of runs assuming the total number of generations run was adequately large enough; however as shown in Figures 3 and 4 comparing runs of the "optimal" and "non-optimal" values, the graphs demonstrate that those values are still not perfect in determining the maximum fitness score of the environment. After further tests, it was found that the input resource configuration has a large effect on how fast the maximum fitness score can be found and the optimality of the solution, meaning it can be difficult to adequately gauge the combination of number of generations, crossover frequency, and mutation frequency needed to reach the optimal fitness.

Overall, the TreeGen project can be deemed a success in that it does converge to optimal solutions when run, as seen in the example provided for section 5. It accepts any input resource configuration of valid encoding and outputs a solution. The TreeGen team learned important aspects of programming genetic algorithms first-hand and, after encountering issues that can arise, reflected on design choices and future improvements.

## 7. Future Work

The algorithm runs with the number of generations being set as a static variable. In order for it to work more effectively on various map sizes, future work to create a solution that determines when the algorithm should end needs to be implemented. For small maps, such as a 5 by 5 grid, the algorithm finds the most optimal solution quickly so the number of generations needed to achieve the optimal solution is far less than those of bigger maps. To fix this for future runs of the algorithm, the algorithm will be made to terminate when the highest fitness score fails to increase after a number of generations that is determined by the size of the map and other factors, like the population size.

TreeGen in its current state highly simplifies the process of determining the resources needed for a tree to survive in its environment. In order to get more applicable results, further in-depth research of silviculture is needed. In the current state there is only 3 resources that are present in the map: water availability, soil compatibility, and sunlight levels. With a lack of silviculture expertise, we simplified the sunlight and water levels just to be randomized to be an integer and the soil be a binary boolean. The next step of the project would research the various soil types and have integer values to properly represent them. Furthermore, in the fitness function, calculations on tree survival would have to account for the various soil types. Also, in the current version of the project it is assumed that there is only one type of tree in the population and that all trees have identical resource needs. To make the algorithm more robust, there would be various tree types, each with their own threshold values for sunlight and water, as well as different soil compatibilities. This would not change the results drastically, but it would make for an algorithm that is more applicable to real world scenarios.

## References

[1] S. Akyol and B. Alatas, "Plant intelligence based metaheuristic optimization algorithms", Artificial Intelligence Review, vol. 47, no. 4, pp. 417-462, 2016. Available: 10.1007/s10462-016-9486-6.

[2] M. Ghaemi and M. Feizi-Derakhshi, "Forest Optimization Algorithm", Expert Systems with Applications, vol. 41, no. 15, pp. 6676-6687, 2014. Available: 10.1016/j.eswa.2014.05.009.

[3] "How to measure trees for inclusion in the Tree Register", *Treeregister.org*, 2019. [Online]. Available: https://www.treeregister.org/measuringtrees.shtml#girth. [Accessed: 12- Jul- 2019].