CS F407 : Artificial Intelligence

Genetic Algorithm

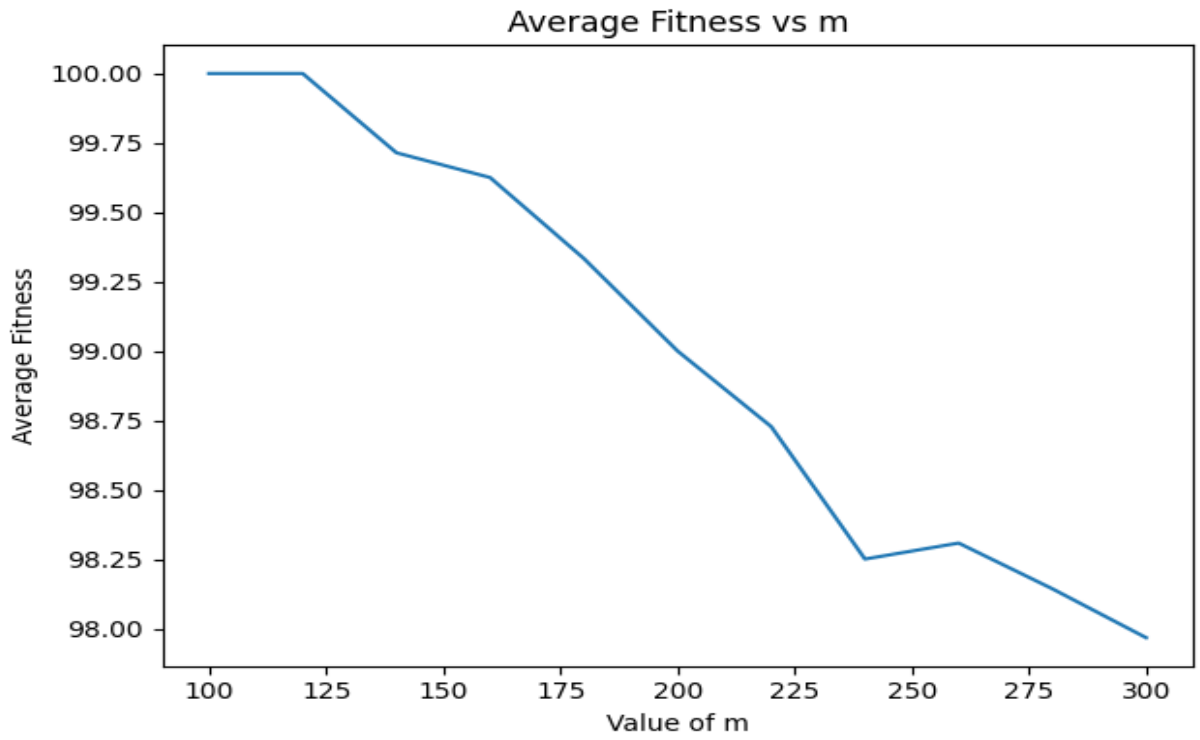
Programming Assignment 1

PROJECT REPORT SUBMITTED BY
Taher Lilywala
2018B1A70609G

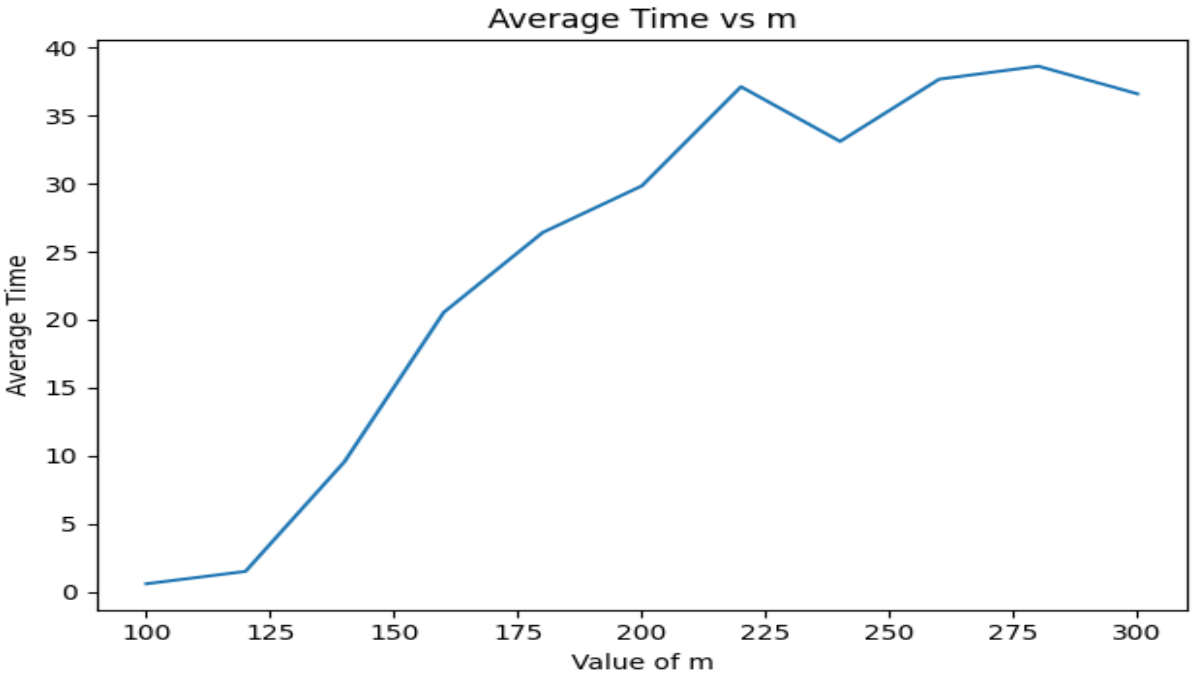
INSTRUCTOR IN-CHARGE
Professor Sujith Thomas

Contents:

1. Average Fitness Values v/s Number of Clauses(m)



2. Average Time For Execution v/s Number of Clauses(m)



3. Final Algorithm and the Improvements Made

The Final Algorithm:

To start off a population is created as a numpy array of size 50(number of symbols specified) each being assigned a value of 1 or 0 signifying True or False respectively.

A while loop is started and for each generation, the fitness is calculated and the population is sorted in the order of decreasing fitness. The population size starts off at 5 initially and is increased by 1 after every 100 iterations till it caps off at 15.

20% of the individuals with the highest fitness are sent directly to the next generation. Of the remaining, 90% of the fittest individuals mate with each other to produce a number of children equal to the population size which are then added to the new generation. Selection for this mating happens through a weighted probability distribution based on fitness. A mutation is introduced in each child with a probability that decreases with generations but increases when fitness starts to plateau.

The model is terminated if one of the following occurs:

- Maximum fitness of 100% is achieved
- Run time passes 45 seconds
- Fitness has plateaued for 30 seconds (Early Stopping)

Improvements Made:

The following changes were made to the standard Genetic Algorithm, with their reasoning explained:

- Dynamic Population Size- It starts out with a small population size to obtain more generations more quickly before increasing steadily till a cap of 15 to have a larger pool to breed from. This approach was adopted partly through trial and error as it gave better results than a stationary or decreasing population.
- Elitism- First, by directly sending 20% of the fittest individuals to the next generation and second by mating only the top 90% of individuals. This helps bias the algorithm towards fit individuals.
- Probabilistic selection- Picking mating pairs using random.choices, using the fitness values as weights.
- Early stopping- A modified form of the usual approach was used to exit based on time elapsed(terminating model if fitness plateaued for 30 seconds). Keeping a

standard number of generations elapsed without improvement before termination was not very helpful as different values of m drastically affect the number of generations executed.

- Mutation Probability Function- A probability value given by p = e^(-g/(100*c) was used, where g is the number of generations and c is the generations elapsed without improvement. This function was used as it starts out with a high probability of mutation which reduces with time to make the model more dynamic starting out. It also increases mutations if a plateau is reached to help explore more variations even at a higher number of generations elapsed.

Failed Approaches:

- Having a constant mutation probability- It gave subpar results because of either having too few or too many mutations at any given point in the model execution. Hence a dynamic approach was adopted.
- Increasing population indefinitely- Beyond a point the returns were sparse, so a cap was introduced to ensure there was a balance between time to execute one generation and total generations elapsed.
- Breeding to obtain children through all possible parent pairings- The idea was to have more options to pick the fittest from for the next generation. However the tradeoff taken to execute was too long and instead a probabilistic function was introduced to have only as many children as needed.

4. Inference From Graph to Determine Problems Where the GA Algorithm Struggled

As the value of m increases, the problem space to be explored increases exponentially. Smaller models also have a higher likelihood of a possible solution existing and a larger number of total solutions because of the fact that there are fewer constraints to satisfy. With an increasing number of clauses(m), the possibility of the function getting stuck in a local maxima is also higher as thanks to the larger problem space the distance between them would be higher. Thus, the model generally struggles with larger values of m.

Also since it is not possible for the model to know beforehand whether a satisfiable solution or one better than the current one exists, it has an almost exponential increase in runtime likely wasted exploring dead-ends.

5. Inference From Graph About the Difficulty of Solving a 3-CNF in n Variables

The difficulty of solving a 3-CNF in n variables increases with an increase in the value of the number of clauses. This is because with an increase in the number of clauses, the number of possible assignments to each of the n variables becomes more tightly constrained, with unsatisfiable sentences becoming more common as m increases. As the number of clauses increases, the number of times a given variable appears also increases. Each of these occurrences require particular assignments to satisfy and there is an increase in the possibility of having contradictory assignments, ie, having one clause where the variable needs to be T and another where it needs to be F to satisfy the whole 3-CNF. This is evident from the graph given how the average maximum fitness reached decreases with the value of m. So, a 3-CNF in n variables becomes harder to satisfy at larger m values.