CS F407: ARTIFICIAL INTELLIGENCE REPORT

Programming Assignment 1

Omkar Garad

2019A7PS1010G

ABSTRACT

In the first programming assignment, we were expected to create a modified Genetic Algorithm that found the best model to satisfy the most clauses in a 3-CNF sentence made with a variable number of clauses.

INTRODUCTION

I implemented the Genetic Algorithm given in the textbook- Artificial Intelligence: A Modern Approach, Third Edition, Stuart J. Russell and Peter Norvig. This involved passing in a population of 20 randomly generated models into the Genetic Algorithm in order to obtain the optimal model.

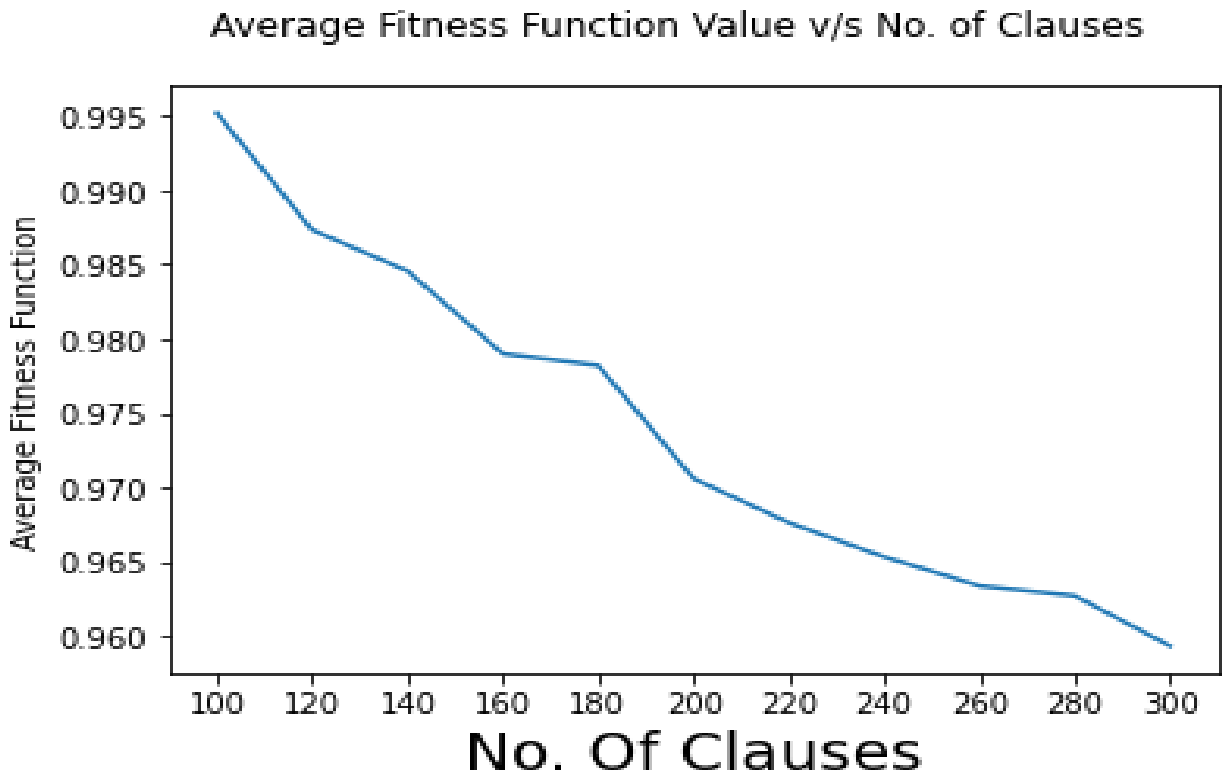
The optimal model is the one which has the greatest fitness function value. This fitness function is the percentage of clauses that are satisfied in the 3-CNF sentence. Our whole aim in this exercise is to maximize the fitness function and minimize the time required to do so.

The population size initially was 20 and the number of variables was fixed at 50. After coding the algorithm given in the textbook, I modified it using techniques said in class as

well as my own ideation (explained in Q3).

Then I plotted the graphs for average fitness function values and average time taken to reach those values in the graphs given in Q1 and Q2.

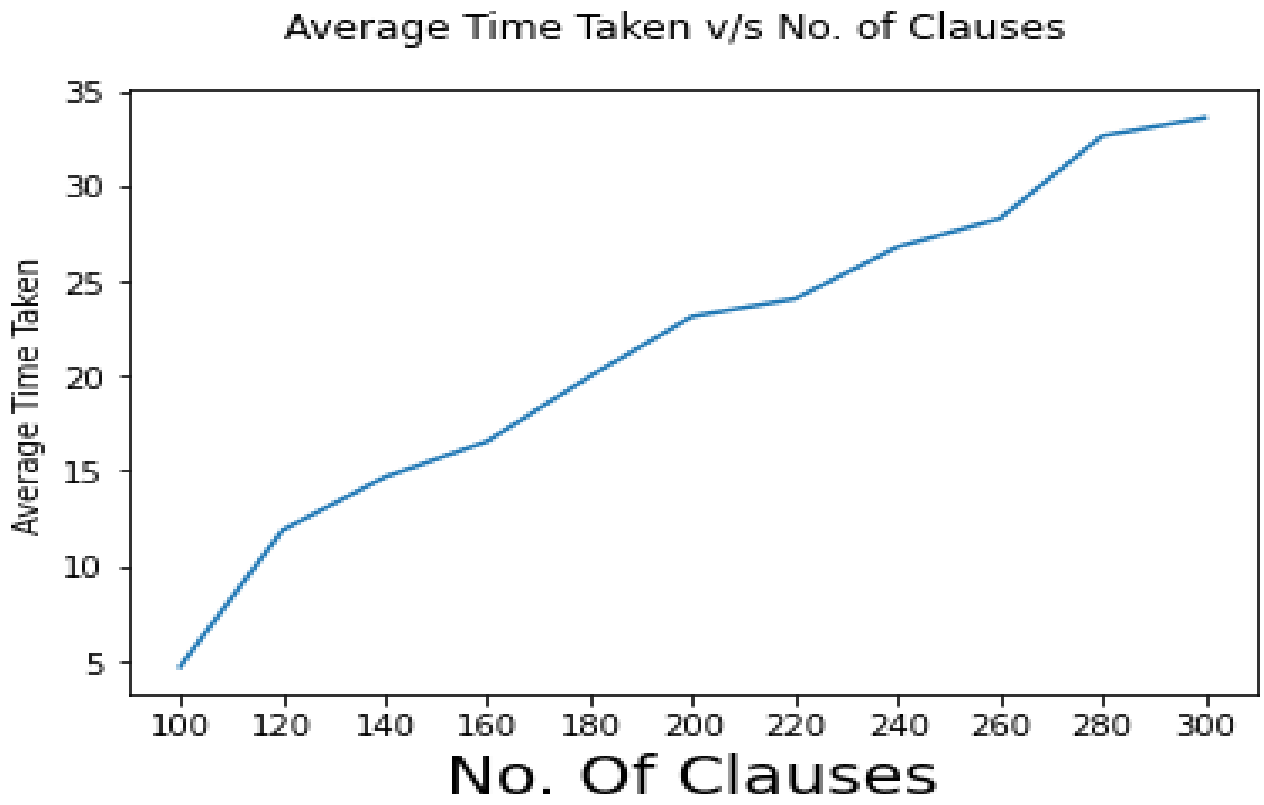
Q1. AVERAGE FITNESS FUNCTION GRAPH



NOTE: Here Average Fitness Function value lies between 0 and 1 (with 1 being the maximum fitness function value).

This graph gives the average fitness function value of the optimal model returned from the modified algorithm v/s the no. of clauses (ranging from 100 to 300 in increments of 20). The number of sentences to be taken were at least 10 so here I have taken 25 sentences to create a smoother curve than what would have appeared at 10 sentences.

Q2. AVERAGE TIME TAKEN GRAPH



NOTE: Here Average Time Taken is in seconds.

This graph gives the average time taken for the modified algorithm to achieve the optimal model v/s the no. of clauses (ranging from 100 to 300 in increments of 20). The number of sentences to be taken were at least 10 so here I have taken 25 sentences to create a smoother curve than what would have appeared at 10 sentences.

I had also decided to terminate the algorithm if the best fitness function value wasn't changing over several generations (50) and also if the fitness value reached 100%.

Q3. How did you improve the GA algorithm? Which approaches failed to give the expected results?

I initially started off with some of the variations that were discussed in class and then used my own ideas as well.

1. **SIZE OF THE POPULATION-** As the size of the population decreases, the time required for the algorithm to execute should also decrease. Given an initial population of size 20, while trying to reduce the size to lower values, **although the iteration time of the algorithm decreased, the fitness function value also decreased and gave an overall poorer result.**

   On the other hand, increasing the population size in order to increase the space the algorithm searches also led to poorer results. Hence, I decided to keep the population size as 20.

2. **MIXING NUMBER-** In the original GA, we take two parents with some weighted probability based on their fitness function values and reproduce a child model. I tried increasing the number of parents involved in reproduction to 3, however, **it made the algorithm more computationally intensive and achieved poor results with respect to fitness function value.**

3. **SELECTING PARENTS-** I tried selecting the best two parents by selecting n parents from the population (in my case n=4) and after sorting based on fitness function value chose the best two parents for reproduction. **However, this again led to poorer fitness function values as compared to just selecting parents based on their weighted probabilities.**

4. **INCREASING WEIGHTS OF BEING CHOSEN-** I did, however, improve the chances of better parents getting picked by multiplying their fitness function value by 100 and then raising to power 6 to make sure higher fitness value models were chosen with a higher probability. **This managed to help improve the fitness function value with no visible change in computation time.**

5. **SELECTING A CROSSOVER POINT-** Instead of randomly selecting a crossover point, I iterated through the model in order to obtain the crossover point that produced the best fitness function value and returned that. **Although it did increase computation time by a bit, it helped tremendously improve the algorithm.**

6. **MUTATION RATE-** In my algorithm, the fitness function value was improving in the first couple of iterations, but then remained stagnant for the remaining time (hence, why I decided to terminate the algorithm after 50 generations). This, I believe, is due to the algorithm reaching the local maximum instead of the global maximum.

   Hence, I kept the mutation rate low (around 0.01) until the frequency of repeated fitness function values reached 10. After that I increased the probability of mutating the child to 0.5 so that there was a better chance of the model exiting the local maxima and reaching global maxima.

   This did tremendously help the algorithm.

   Along with this, I decided to find out the best position for mutation of the child, which did increase computational time but also produced improved results for the algorithm. I also decided to return 2 children instead of 1. Hence, in the mutation algorithm, I returned two children that mutated at both the best and second best mutation position from the function (given their fitness functions were greater than the input child).

   This also greatly improved the algorithm.

7. **CONSTRUCTION OF NEXT GENERATION:**
   a) **ELITISM:** I retained the two best models from the previous generation in the new one. The rest were obtained from the present generation. **This improved the fitness function value of the optimal models returned.**
   b) **CULLING-** As I created two children every time I passed the reproduced child into the mutate() function, I used culling in order to select the models with the best fitness function value after sorting the new population created.

8. **REPRODUCING TWO CHILDREN-** I also tried to return the two best children from the reproduce() function. This was done by selecting the best and second best cutoff points for the input child and returning the best two children out of the four created in the process. **This however, made the fitness function poorer on average.**

9. COMBINATION OF BOTH- I tried combining returning the best and second best children from reproduce() function and then returning the best and second best *mutated* children after passing them into the mutate() function. After this I selected the best models from the new generation to append to the population. **However, this also reduced the average fitness function value.**

Q4. From the above graphs, what can you tell about the problems where the GA algorithm might find it difficult to find a good solution.

1. For the Genetic Algorithm, a large problem I faced was being stuck at a local maxima for the fitness function value rather than a global maxima. The GA must be able to search through the state space efficiently such that it can locate the global maxima and then converge to it, rather than searching in a small area with a local maxima. So if the state space becomes too large and the GA isn't efficient enough, the best solution may not be achieved.
2. Improving the functions to search for the **optimal position** to modify the child may lead to better initial fitness function values, but in the long run it is computationally expensive and may not lead to a large number of iterations. This will in turn reduce the time taken to search the state space for the global maxima. Hence, tweaked functions that are useful for some applications may not be the best for other applications
3. The larger the model size, the longer it takes for the GA to converge. It may also be the case that the GA cannot converge due to time constraints which are mostly always a parameter in real life applications.
4. Also, the larger the model size, the more time it takes to obtain the fitness function of each model in the population. This again increases the computational time of the algorithm, giving less time to search the state space.

Q5. What does the above graphs tell you about the difficulty of satisfying a 3-CNF sentence in n variables. When does a 3-CNF sentence become difficult to satisfy?

1. **As the number of clauses in the 3-CNF sentence increases, it becomes increasingly harder to satisfy.** This is especially true in the case where no. of clauses > no. of variables (n). This is because we may have the same variables appearing in multiple clauses which could lead to a large number of them being unsatisfied.
2. We can also see that even after reproducing new children and modifying them to give the models completely different Boolean values, the percentage of 3-CNF clauses being satisfied still decreases as the number of clauses increases. Hence, satisfying the entire 3-CNF sentence becomes increasingly difficult.
3. Hence, given the number of clauses are large enough and number of variables are low enough, the 3-CNF sentence won't be satisfiable by any given state in the state space.