Artificial Intelligence (CS F407)

Assignment-1

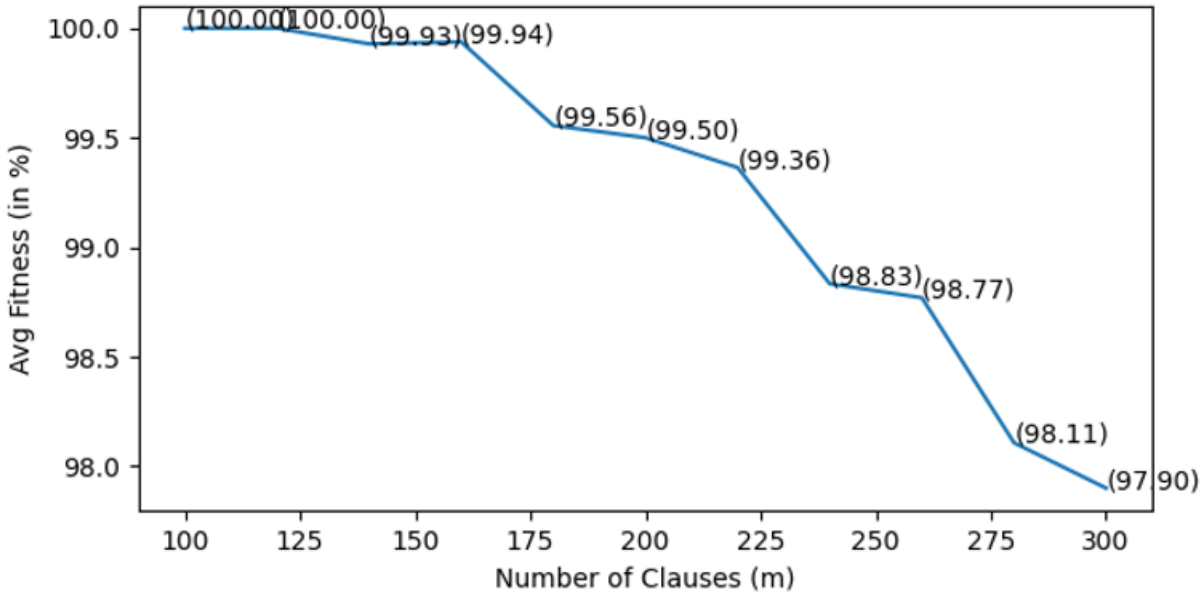
Name: Atishay Jain

ID: 2018B5A70908G

Problem: Genetic Algorithm to solve 3-CNF SAT Problem

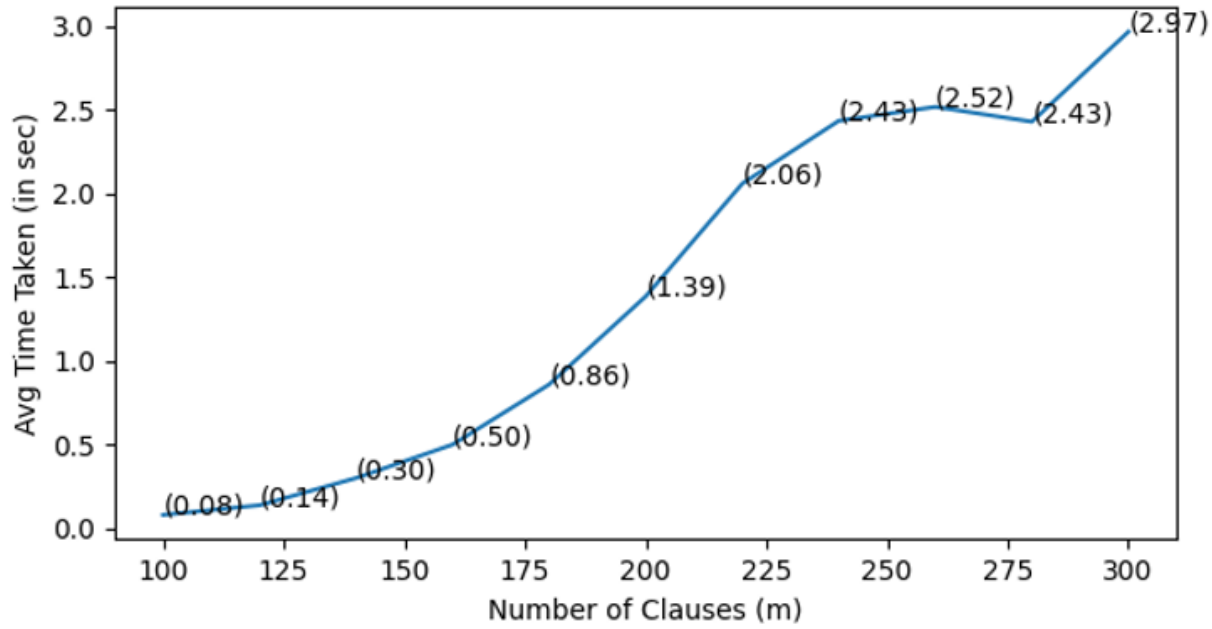
Given a propositional logic formula in 3-CNF form. You have to find a model that maximizes the percentage of satisfied clauses in the formula. The propositional logic formula will be built using 50 variables.

I. Number of Clauses vs Fitness Value



It is observed, with the increase in the number of clauses the average fitness function value decreases as the probability of the sentence being completely satisfiable decreases and the complexity of the algorithm increases.

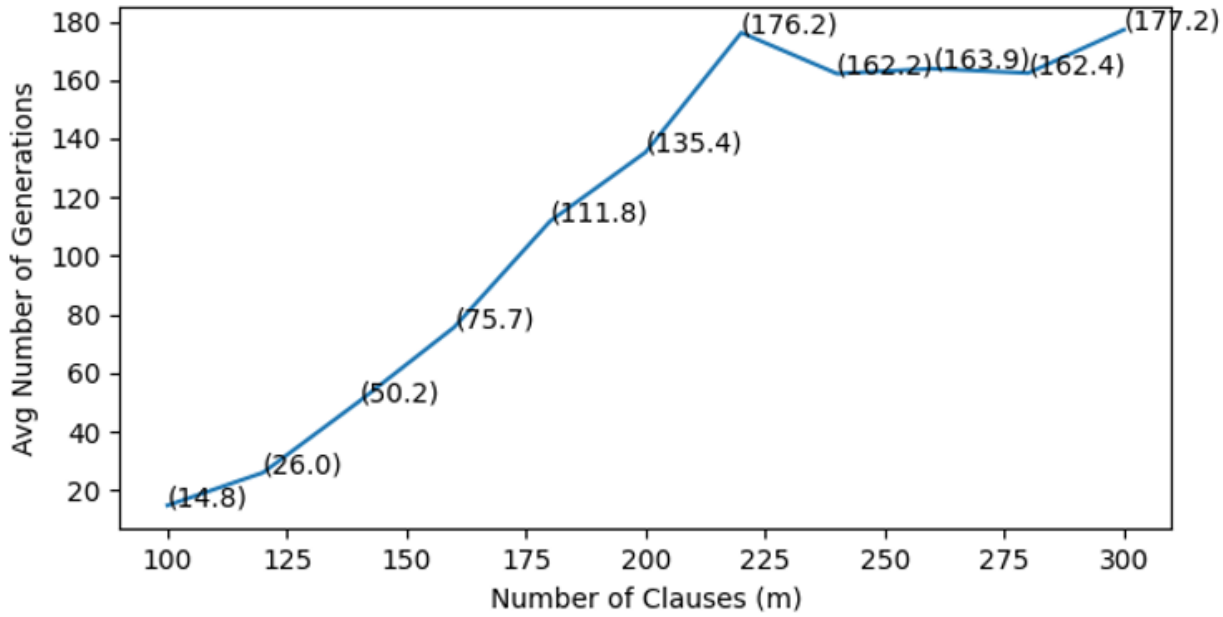
II. Number of Clauses vs Time Taken



As the complexity of the algorithm increases with the increase in the number of clauses, it is observed that the average running time also increases.

Please note, the data shown was generated locally on an Intel I5 8th Gen (1.8 GHz) processor with 8 GB RAM.

For deeper insight, a plot of **Average Number of generations vs Number of Clauses** is also made.



It is observed that the average number of generations follows a similar trend as the average running time of the algorithm.

III. Improvements in the Genetic Algorithm

The Genetic Algorithm given in the textbook leaves a great scope of improvement as it was meant for learning purposes only. With a large number of hit and trial experimentation and some calculated guesses, a good number of changes were made that improved the algorithm.

Approaches that improved the algorithm:

1. **Generate both the children in cross-over**

   This was the first improvement made in the algorithm. Earlier the algorithm was generating only one child from two parents (keeping parent-1 on the left and parent-2 on the right) which left the other child unused. In the improved algorithm, all possible children are generated at the cross-over stage (2 children in case of 2 parents) and the best one is kept (after a conditional check on their fitness function value). This improvement had a good impact on the average time of the algorithm.

2. **Selective Mutation (and removed the condition for mutation)**

   In the improved algorithm, a mutation on a child is kept only if it increases the fitness value function else the mutation is discarded. Since this condition is implemented in the mutation function, all children generated after crossing over are sent for 1-bit mutation i.e., the condition that only a child below a threshold value of fitness function is sent for mutation is removed. This greatly improved the accuracy when the algorithm is run for a short time (less number of generations).

3. **Preserving the Elite Parents**

   This was one of the most logical modifications made. The best 5 parents from the last generation were chosen and passed to the next generation. Some experimentation was done here by choosing the random weighted parents but selecting the top 5 worked the best. It significantly improved fitness value and running time.

4. **Crossing Over in the Middle**

   In the improved algorithm, instead of just left-right cross-over, cross-over in the middle is also possible (left-right cross-over being the special case). For this two random positions were chosen from one parent's string and the substring between these positions was

substituted with the other parent's corresponding substring (and vice-versa). This had a positive impact on both time and fitness value.

5. **Mutate and check before culling**

   As mentioned in the first point, only the best of the two children generated was kept and the other one was culled. In this modification, firstly, the selective mutation was done on both the children after crossing-over (right before culling) and then the best one was chosen (earlier it was done after culling one of the child). This had a significant impact on improving the fitness function value.

Approaches that failed to give results:

1. **Multi-parent Cross Over**

   Crossing over multiple parents doesn't seem to improve the algorithm's accuracy or runtime. Instead, crossing over two parents in the middle helped significantly as discussed in the last section.

2. **Weighted random choice for next best population**

   Earlier as an alternate to preserving the best parents, the next generation was chosen through weighted random choice on both parent and children array combined with fitness function value as the weights. But it had little to no effect on the algorithm.

3. **Selective mutation of Elites**

   Mutating the best-chosen parents with other children (and keeping the mutation only if it improves the fitness value) didn't help as it didn't improve the accuracy much but increased the runtime.

Other things to note:

1. **Saturation Check**

   One of the conditions for the termination of the program is to check if the fitness function value saturates at a point. It was noted that the saturation threshold should be kept at least 100 generations, i.e. terminate the algorithm only if the fitness function value doesn't change for at least 100 generations, else there is a scope of improvement.

2. **Increasing Population size didn't help with this particular problem.**

   Increasing population size increases the runtime and may have an impact on the accuracy. For this problem, the population size of 20 was understood as ideal.

Please Note: The remarks made here are based on Hit and trial method of testing an algorithm and are only for this particular problem (not tested on other problems apart from 3-CNF SAT)

V. Problems where Genetic Algorithm might find it difficult to find a good solution

From the above graphs, it is observed that as the number of clauses (m) increases, the overall fitness function value of the output model decreases, and the time taken increases significantly. It shows that the genetic algorithm might not find a good solution if the complexity of the fitness function is high. With the increase in the fitness function complexity, the number of new generations decreases in a given time, which in turn decreases the amount of overall mutation in the population space. So there is a greater probability of the algorithm to stuck at a local maxima, giving us an inaccurate result. Hence, it is better to use GA Algorithm for a problem for which the fitness function complexity is low.

Further, in general, it is observed that the Genetic Algorithm works better where there is a positional relation between characters of the solution string.

VI. Difficulty of Satisfying a 3-CNF sentence with n variables.

It is observed that for a given number of variables/symbols, 3-CNF sentences become more difficult to satisfy with the increase in the number of clauses (m). This is for two reasons:

1. With higher values of m, the probability of a 3-CNF sentence to be satisfiable is less as there are more number of clauses (and hence more propositional constraints) to be satisfied by a single model.
2. The complexity to check satisfiability increases with an increase in m.

However, if the number of variables (n) and the number of clauses (m) are less, then 3-CNF is relatively easier to solve as the complexity is greatly reduced.