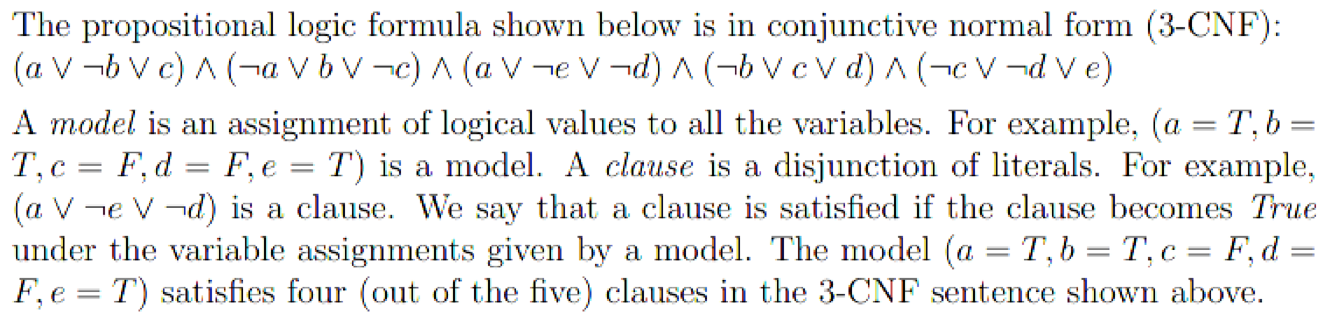
Programming Assignment 1 Report

Artificial Intelligence - CS F407
First Semester AY 2021-22
BITS Pilani K K Birla Goa Campus

Submission by:
Name: Susmit Wani **ID No.:** 2018A7PS0116G
Date: 29/09/2021

Problem Statement: Find a model that maximizes the percentage of satisfied clauses in the 3-CNF formula. We say that a clause is satisfied if the clause becomes True under the variable assignments given by a model.

Sample given below



The propositional logic formula shown below is in conjunctive normal form (3-CNF):
$(a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee \neg e \vee \neg d) \wedge (\neg b \vee c \vee d) \wedge (\neg c \vee \neg d \vee e)$

A *model* is an assignment of logical values to all the variables. For example, $(a = T, b = T, c = F, d = F, e = T)$ is a model. A *clause* is a disjunction of literals. For example, $(a \vee \neg e \vee \neg d)$ is a clause. We say that a clause is satisfied if the clause becomes *True* under the variable assignments given by a model. The model $(a = T, b = T, c = F, d = F, e = T)$ satisfies four (out of the five) clauses in the 3-CNF sentence shown above.

Methodology: First, a vanilla genetic algorithm was implemented to get a feel of how it worked and to understand how it actually worked. Then, certain modifications were made to the genetic algorithm to substantially improve the running times and to enable the algorithm to find solutions to 3-CNFs quickly and more frequently. In this assignment, the population size was fixed at 50 and the sentence length was varied.

Changes made to improve the Genetic Algorithm:
1. Changed fitness function to be the square of percentage of terms satisfied. This works better as there is a much clearer distinction between good and bad genes due to squaring of value.
Eg: If fitness percentage is 0.9 and 0.99, the difference between squares is substantial (0.1701) as compared to absolute difference (0.09)

2. Reject child (abandon further calculations and create new child) if fitness of current child is less than the fitness of both the parents. This way, we are essentially making sure we are keeping only stronger children in the population

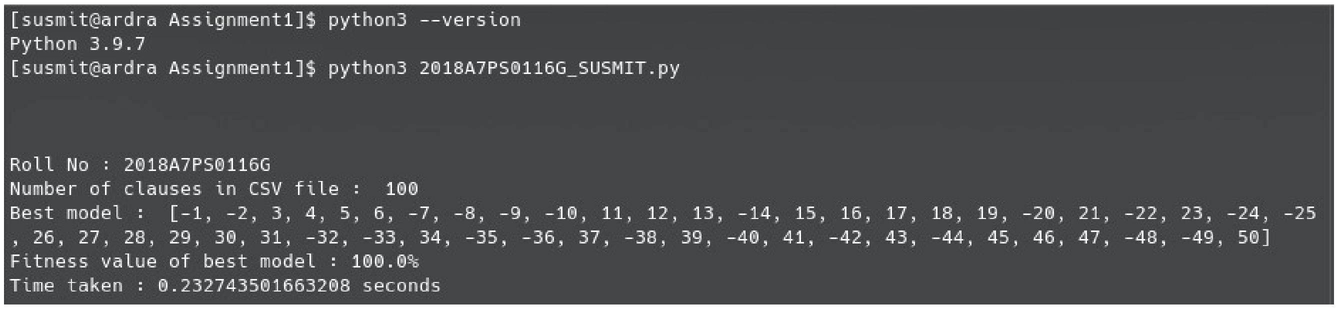
3. Modified mutate function to flip one bit based on the unsatisfied clauses to generate new child. If a clause is not satisfied then add all three variables to the unsatisfied variables list and use that to check which bit to flip.

4. Used the above modified mutate function and traditional mutate function to generate twice the number of children as the population size and chose best half amongst them. This is a greedy approach and works better most of the times.

Changes made to the genetic algorithm that didn't work:
1. Once the clause satisfiability percentage reached a certain threshold, say, if length of sentence is n then if satisfiability percentage reached (n-2)/n, then check each and every neighbour of the newly produced child. This approach failed very miserably because it was a lot of computation and because of the problem of being stuck in a local minima.

2. Changed reproduce function to take features from 4 parents instead of 2. But this gave very similar results to using 2 parents but with increased computation times so this idea was dropped.

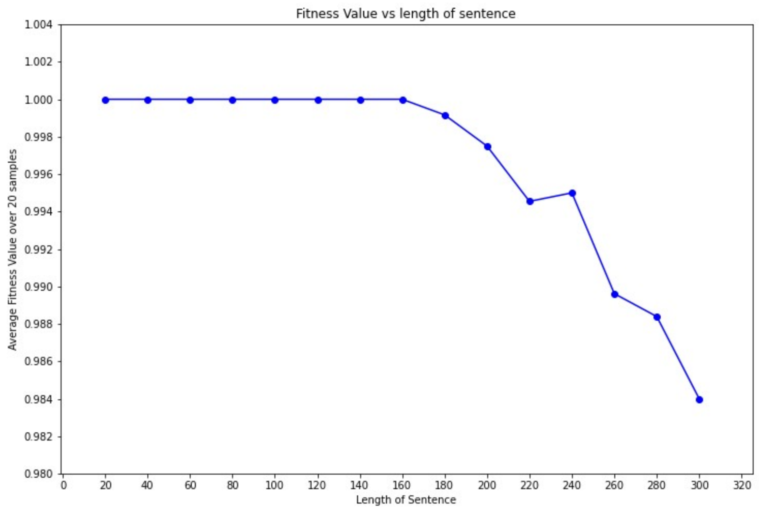
Output of the code on Python 3.9.7:



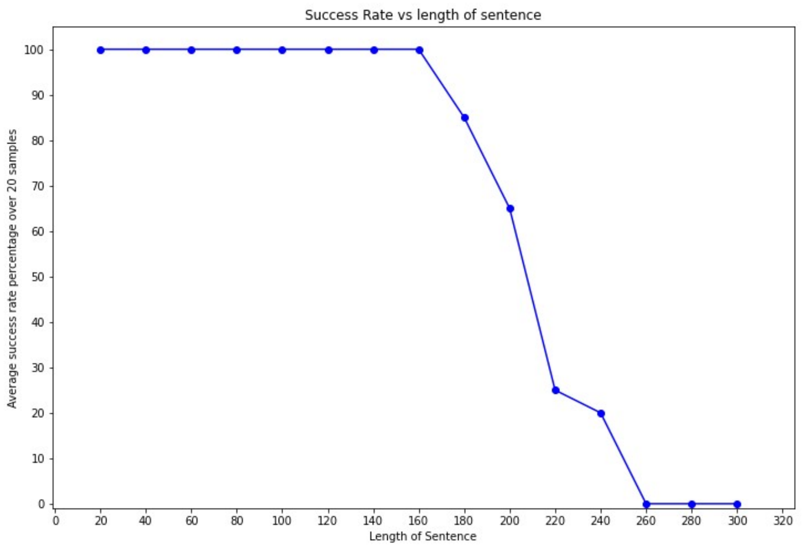
```
[susmit@ardra Assignment1]$ python3 --version
Python 3.9.7
[susmit@ardra Assignment1]$ python3 2018A7PS0116G_SUSMIT.py



Roll No : 2018A7PS0116G
Number of clauses in CSV file :  100
Best model :  [-1, -2, 3, 4, 5, 6, -7, -8, -9, -10, 11, 12, 13, -14, 15, 16, 17, 18, 19, -20, 21, -22, 23, -24, -25
, 26, 27, 28, 29, 30, 31, -32, -33, 34, -35, -36, 37, -38, 39, -40, 41, -42, 43, -44, 45, 46, 47, -48, -49, 50]
Fitness value of best model : 100.0%
Time taken : 0.232743501663208 seconds
```

Visualising Results: A total of 20 random samples were generated for sentence length from 20 upto 300 with step size of 20. The following data was gathered from the runs. Interpretation of the data will be made in the upcoming sections.

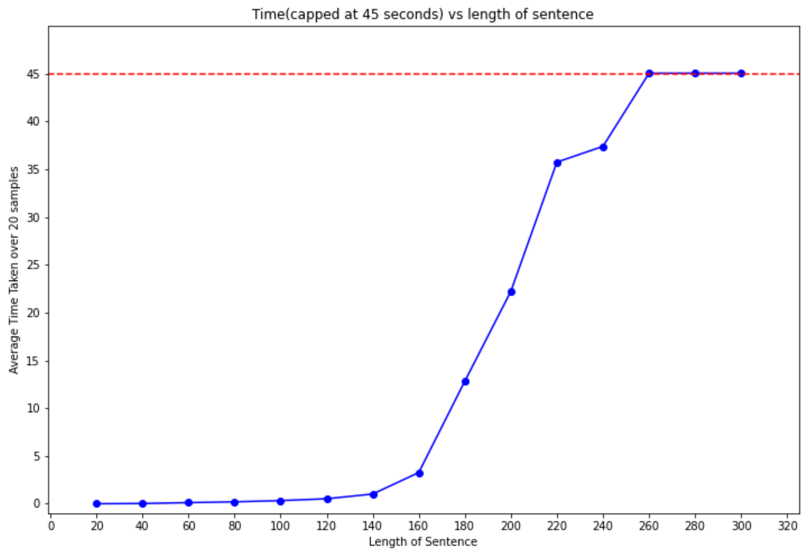
1. *Average Fitness vs Length of sentence:* As we can see, while the sentence length is less than 180, we get 100% fitness. After this, there is considerable drop in average fitness value. This is because the sentence becomes unsatisfiable and some clauses can never be satisfied. Thus these clauses remain unsatisfied and on the whole, fitness value decreases.



2. *Average Success Rate vs Length of sentence*: Again, if we vary the sentence lenth from 20 to 160, we get a fully satisfiable 3-CNF 100% of the time. This can be seen from the previous graph as well. But more interestingly, we can see that the sentences in this test with lengths greater than or equal to 260 are not at all satisfiable. For intermediate values of sentence lengths, we can see a steady drop in success rate of our model from 100 to 0.

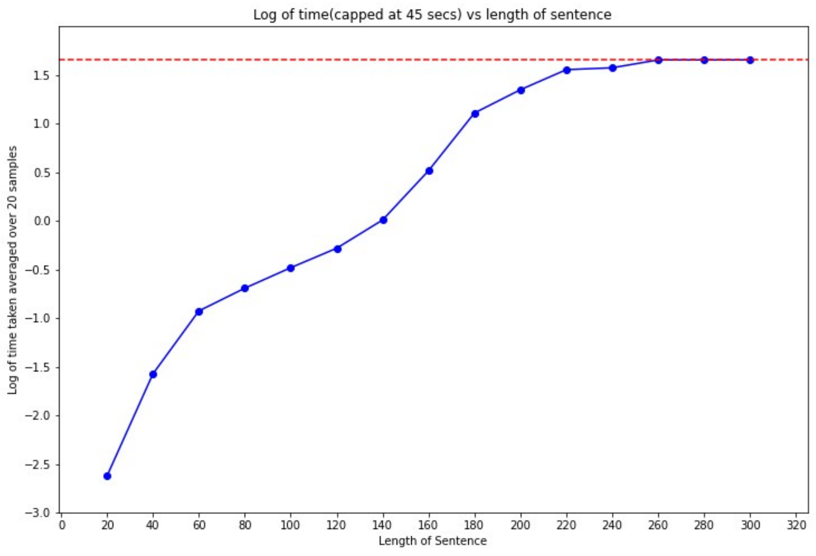


3. *Average Time vs Length of sentence*: Immediately, we can see that the graph follows kind of an exponential rise in time taken to find solution. The red line is drawn at y=45 seconds as the run terminates at 45 seconds. As expected, for sentence length greater or equal to 260, because out success rate was 0, average time taken is 45 seconds.

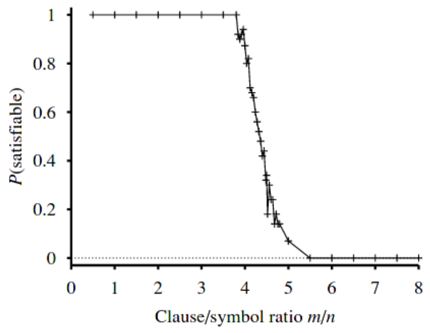


4. *Bonus Graph: Log of time vs Length of sentence*

I thought it would be interesting to plot the log of time vs length of sentnce since the previous graph appeared to follow an exponential pattern. As we can see we can sort of fit a linear line, of course with some degree of error, through the data points till sentence length 220.



Interpreting the results (on the difficulty in solving 3-CNF): As we can see from the graph of log of times needed, till a sentence length of 140, we can find the solution to the 3-CNF under 1 second. But after that, the times grow rapidly. Around 12 seconds for sentence length 180 and 35 seconds for sentence length 220. This could be because as the total number of clauses grow, the occurrences of variables increase in number and the probability that the sentence becomes unsatisfiable also increases, thus leading to overall non-satisfiability of the sentence.



According to the graph shown alongside (taken from Stuart Russell, Peter Norvig - Artificial Intelligence, A Modern Approach 3e), for a 3-CNF with 50 variables, the satisfiability percentage varies as shown where m is the number of clauses and n is the number of variables. The same thing is observed in the previous success rate vs length graph. From around 200 clauses to 275 clauses satisfiability drops from 100% to 0% which can be verified independently from the graph above.

Problems with Genetic Algorithm: In general, the main problem with the GA is that if the fitness function to the problem contains a lot of local minima, it will most likely get stuck there and will struggle to find the optimal solution. The genetic algorithm is kind of a greedy algorithm in the sense that it focusses more on reproducing the more 'fit' parents. So it may happen that a niche feature is not carried forward because other features make up for it but the optimal solution needs that niche feature to be present. Thus, we get stuck in a local minima due to this. This is a problem with all greedy algorithms. This could be solved by increasing the mutation rate so that it is possible to get that niche feature back into the generations to obtain the optimal solution.

END OF REPORT