BITS Pilani, K. K. Birla Goa Campus

Department of Computer Science & Information Systems

CS F407 - Artificial Intelligence: Programming Assignment 1

Nimish Wadekar

2019A7PS1004G

September 29, 2021

1 Aim

This project aimed to find a *model* (valuation) of logical truth values of N variables, that satisfied a given *sentence* (boolean expression), also comprising N variables, using a genetic algorithm. The sentence was in conjunctive normal form (CNF) and consisted of M *clauses*. Each clause had 3 disjunct terms. A sentence with $N = 5$ and $M = 5$ thus looks like:

$$(a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee \neg e \vee \neg d) \wedge (\neg b \vee c \vee d) \wedge (\neg c \vee \neg d \vee e)$$

2 Method

Python 3.9.7 was used to implement the genetic algorithm. A model was represented as a *boolean* array of size N. A variable was represented as an integer in the range $[1, N]$, with its sign representing non-negated $(+)$ or negated $(-)$. A 3-CNF sentence was represented as an array of 3-tuples of such variables, of size M. For the purposes of gathering data, N was fixed at 50, and M was varied in the range $[100, 300]$, in steps of 20. The *fitness value* of a model with respect to a sentence was calculated as the percentage of clauses from the sentence that it satisfied, over the total number of clauses in the sentence.

The initial version of the algorithm implemented had the following features:

- A randomly generated *population* (set of sentences) of size 20.
- Selection of two individuals for reproduction by randomly selecting two positions in the population, each of them with equal probability.
- Crossing over by randomly selecting a crossover point on a parent. Of the two offspring formed, one is randomly selected as part of the next generation.
- Mutation by negating the truth value of each variable based on a fixed probability equal to 0.04.
- In each iteration, a generated offspring was returned as a satisfactory state if it had a fitness value of 100%. Otherwise, the model with the best fitness value was being kept track of, and was returned after a timeout of 45 seconds.

2.1 Modifications

2.1.1 Improving Modifications

The aforementioned algorithm produced models with a maximum fitness value of 98%, for $M = 100$. The algorithm was then updated and tested with the following modifications:

- The selection of individuals for reproduction was modified to select one individual based on a probability that is proportional to its fitness value, while the other individual is selected from the opposite side of the first individual, treating the list of individuals as a circular list, i.e. if the index of the first selected individual in the population is i, the index of the other individual would be $(i + PopulationSize)\ \%\ PopulationSize$, where $\%$ is the *modulo* operator. This ensured sufficient diversity in the next generation.
- Reproduction was modified in two ways:
  - Two crossover points were selected instead of one. The crossing over thus looked like this:

$$\left|\begin{matrix}1&2&3&4&5&6\\1&2&3&4&5&6\end{matrix}\right| \Rightarrow \left|\begin{matrix}1&2&3&4&5&6\\1&2&3&4&5&6\end{matrix}\right|$$

  - Of the two offspring, the one with the better fitness value was selected to proceed.
- During the mutation stage, a mutant was only allowed to proceed if its fitness value was higher than the original, unmutated individual.
- A form of elitism was implemented such that an offspring is allowed to become a part of the next generation only if it has a higher fitness value than both of its parents. Otherwise, the parent with the highest fitness value is chosen to proceed unmodified.

- The algorithm keeps track of stagnation γ, i.e. the number of consecutive generations that have all resulted in a best state with the same fitness level. It uses this information to update the probability of mutation according to the formula:

$$P\left(mutation\right) = \sigma\left(\frac{8 \cdot \gamma}{50 \cdot (\lfloor \frac{\gamma}{50} \rfloor + 1)^{1.5}} - 4\right), \quad \gamma \geq 0;$$

where $\lfloor x \rfloor$ is the greatest integer less than or equal to x, and $\sigma\left(z\right) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

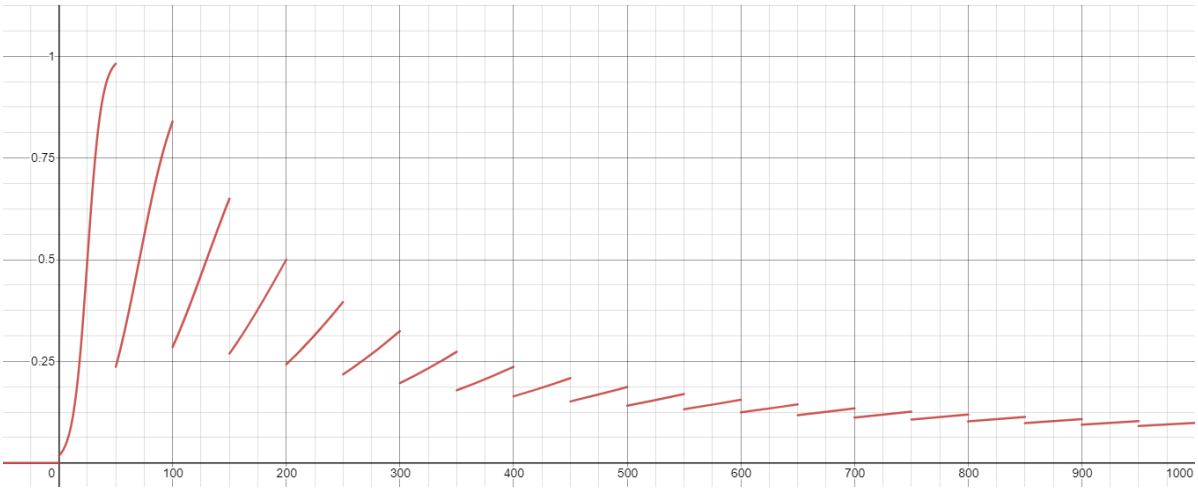

Figure 2.1: A graph of $P(mutation)$ on the $Y\text{-}axis$ versus γ on the $X\text{-}axis$

Figure 2.1 shows us the variation of $P(mutation)$ with increasing γ. For smaller γ, exploration is encouraged due to a high mutation rate, while after long periods of stagnation, the probability of mutation is small to prevent variation from the obtained high fitness values. The assumption made is that if mutations at lower levels of stagnation did not achieve a better fitness, it is unlikely that this changes after hundreds of iterations of further stagnation, and thus $P(mutation)$ is decreased to preserve the obtained models.

2.1.2 Unsuccessful Modifications

The above modifications improved the performance of the algorithm. However, there were other modifications made to the algorithms that did not improve its performance as expected. They were:

- Updating the mutation probability as:

$$P(mutation) = \frac{1}{fitness}$$

where $fitness$ is the fitness value of the unmutated offspring. This calculation was expected to produce a mutation rate where higher fitness individuals mutate less. However, this led to individuals with a fitness value of 99% and above stagnating at a local maximum.

- Choosing both parents independently based on a probability proportional to their fitness. However, his led to a rapid loss in diversity due to the higher fitness individuals being chosen repeatedly, and led to stagnation.

3 Results

Figure 3.1 and Figure 3.2 show the final average fitness values of models returned by the improved algorithm, and its average run times, for different values of M. The algorithm was run on 40 randomly generated sentences for each value of M.

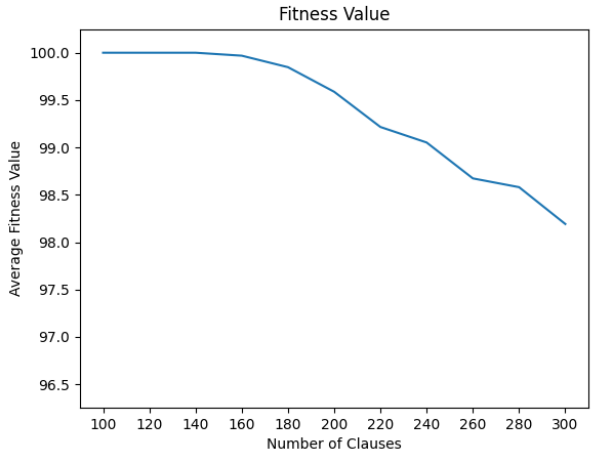

Figure 3.1: A graph of average fitness value versus M

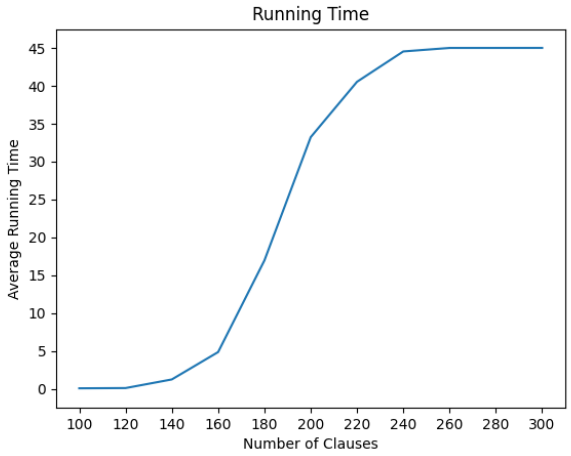

Figure 3.2: A graph of average run time versus M

From Figure 3.2, the average running time of the algorithm shows a steep increase after $M = 160$. Figure 3.1 shows that the average fitness value of the model obtained also dips after an almost constant value before $M = 160$. It can also be assumed that the average time remained constant at 45 seconds after $M = 240$ due to the timeout imposed on the algorithm.

4 Conclusion

It is evident that as the number of clauses increases linearly, the running time of the algorithm increases non-linearly with respect to M. It can be hypothesized that the algorithm finds it difficult to find a solution for a 3-CNF sentence of N variables because since each clause can be satisfied by 3 different variables, as the number of clauses increases, the number of states that satisfy k clauses, $0 < k < M$, increases more rapidly, thus increasing the difficulty of finding the best model and causing stagnation.

In general, from the above data, we can hypothesize that a genetic algorithm finds it difficult to find the best solution when the number of non-best solutions increases at a much faster rate than the number of best solutions.