AI Assignment 1

-Asmita Limaye 2018B5A70881G

1. Graphs of Final Improved Algorithm

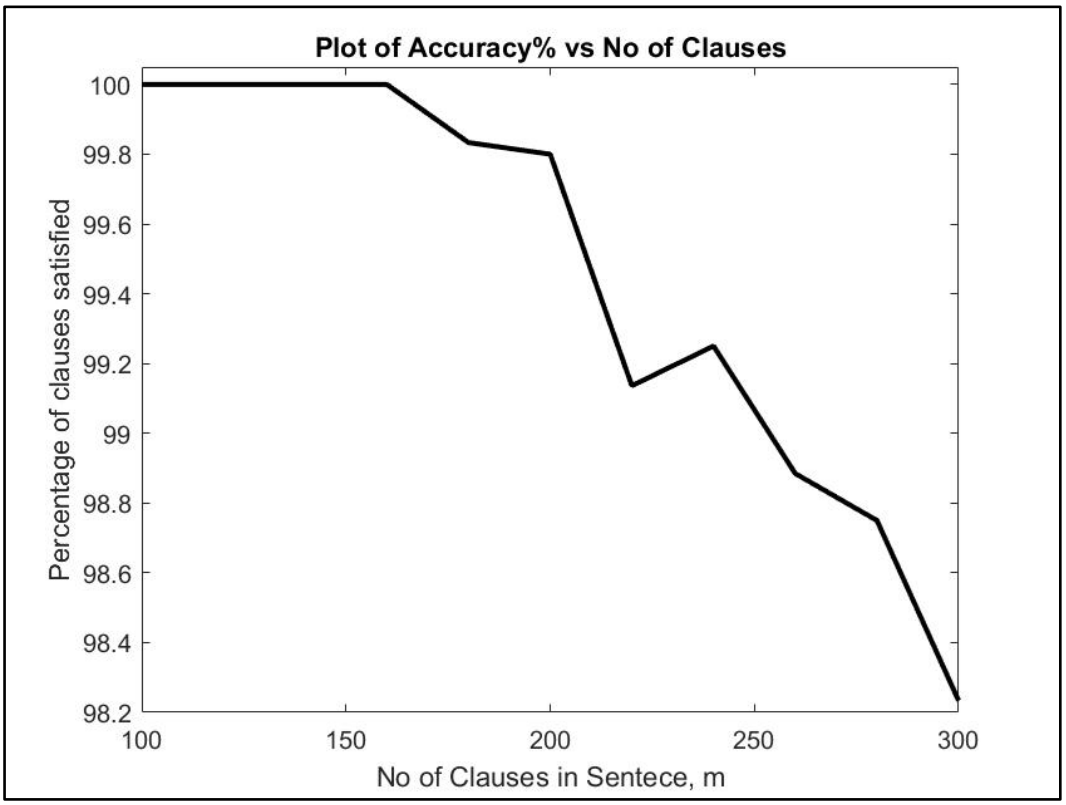

Fig 1: Average fitness(percentage of clauses satisfied) in a 3-CNF sentence, averaged over 10 sentences vs no of clauses(m) in the sentence

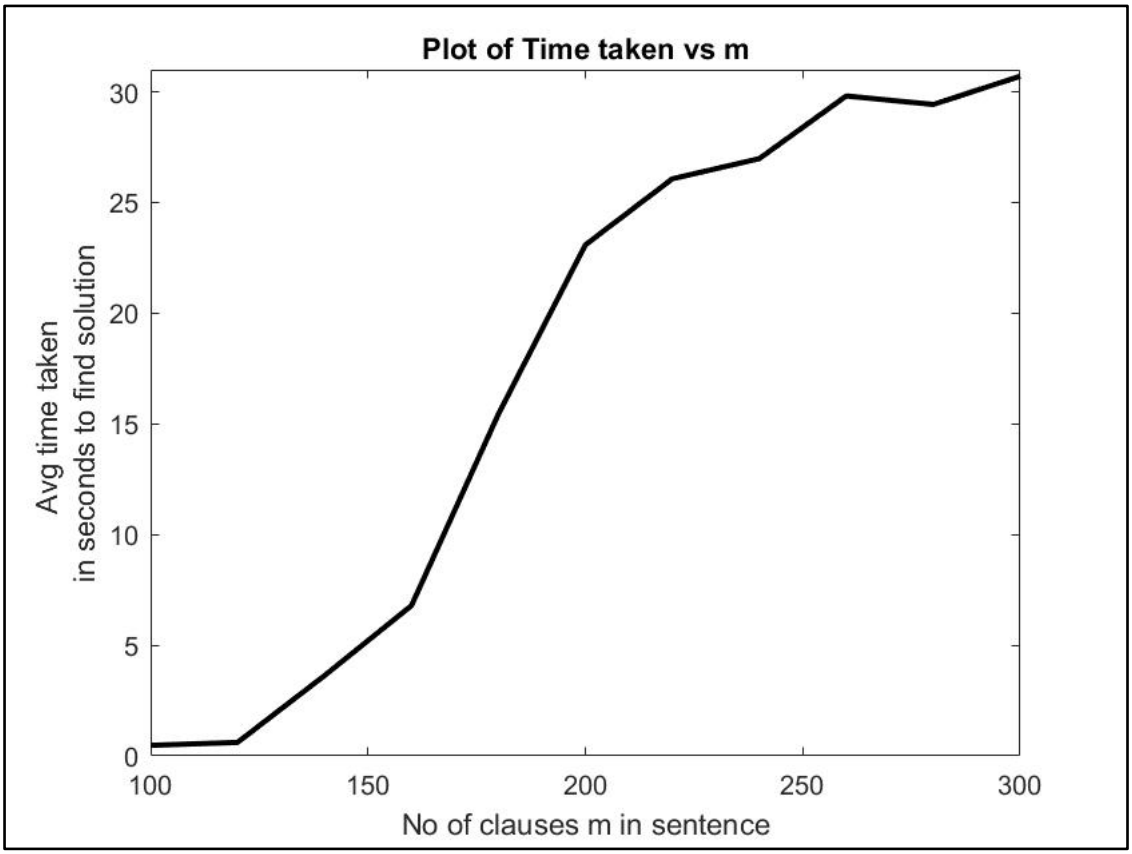

Fig 2.Average time taken in seconds to find solution by the genetic algorithm, averaged over 10 3-CNF sentence vs no of clauses (m) in the sentence

2. Attempts at Improvement from Original

I ran the vanilla version of the GA given in the class notes, with a 45 second time limit, 1 variable mutation of the model and a population size of 20. I decided to use a sentence with 350 clauses (from the CNF5.csv). I then changed the population size to 50 as the convergence was too slow otherwise.

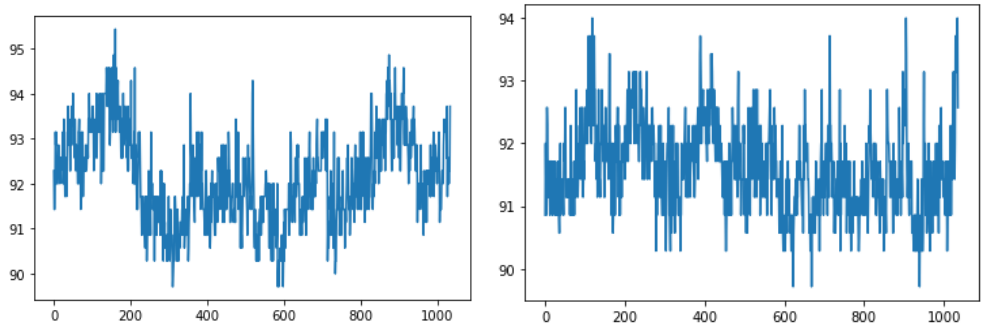

Fig 3 and 4. Plot of best fitness function in a generation vs no of iterations of 2 runs of the VANILLA algorithm

Random restart:

I saw that that the fitness percentage was often decreasing and there was no consistent improvement. I decided to implement a random restart strategy, which would check if the best fitness in a generation was less than the max fitness so far for more than 10 seconds (plateau or decreasing values for >10 seconds), and if so restart. The results were not very impressive.

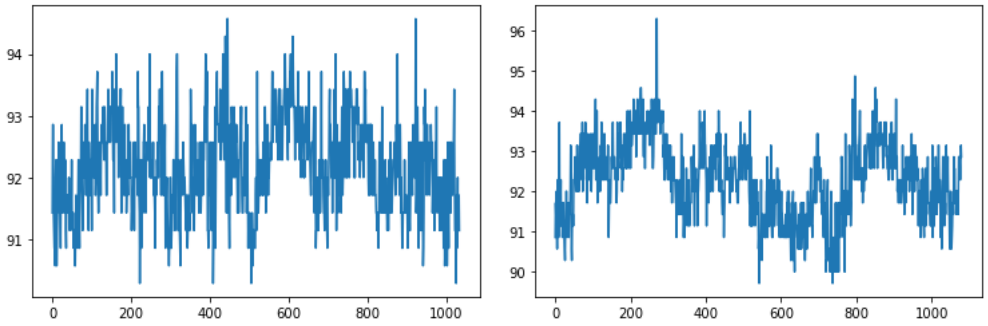

Fig 5 and 6. Plot of best fitness function in a generation vs no of iterations of 2 runs of the random restart algorithm (10s)

Elitism:

The previous strategy did not give me great results, so I decided to implement elitism by taking half of the next population from the parents, and the other half from the children on the basis of their fitness functions. This showed me very great improvement.

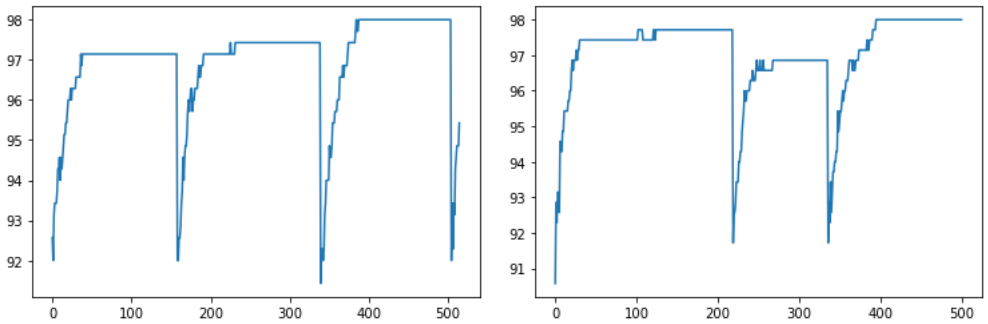

Fig 7 and 8. Plot of best fitness function in a generation vs no of iterations of 2 runs of the random restart algorithm (10s) with elitism

Culling:

I then decided to try culling by generating a children population of twice the size that was needed and then taking the best children of the desired population size. This showed improved similar graphs to elitism, but needed to restart more often. Changing the size of the children population to three times the required one and then taking the best children did not improve it significantly.

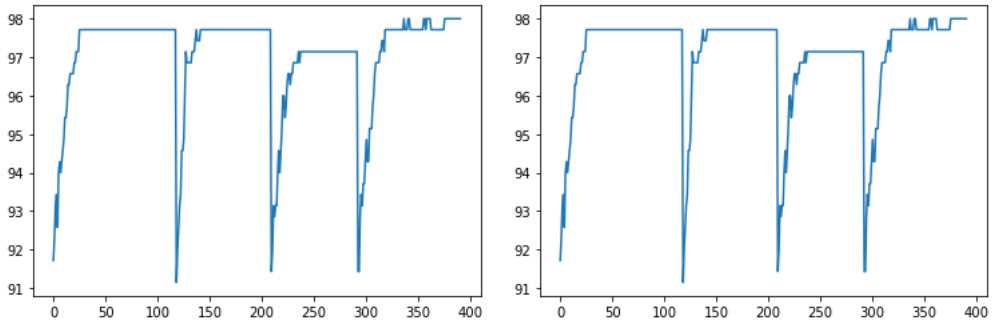

Fig 9 and 10. Plot of best fitness function in a generation vs no of iterations of 2 runs of the random restart algorithm (10s) with culling

My final strategy involved calculating desired population size(n) of children, then sorting **both parents and children** on the basis of their fitness function and taking the best n out of those. This ensured that the best model would not be lost as parents had children, and the graphs converged to higher accuracies quicker, and **they did not fluctuate as much.**

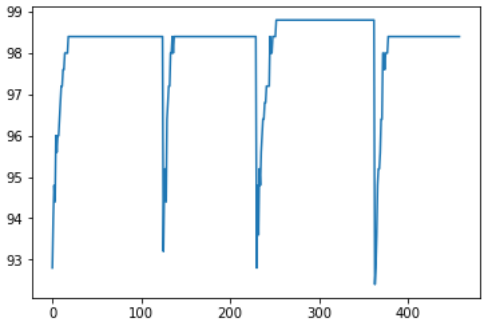

Fig 11. Plot of best fitness function in a generation vs no of iterations of the random restart algorithm (10s) with elitism

Changing random restart time to 5 seconds:

Noting that there were on average 4 10 second plateaus in 45 seconds, I decided to decrease the random restart time to 5 seconds so there was more opportunities to get to a favourable outcome.

Limiting no of restarts:

For sentences with lower clauses, the algorithm did not need many random restarts, but for sentences with higher clauses, I saw the algorithm hit its maximum accuracy value within 5 restarts most of the time. These following graphs are a typical example:

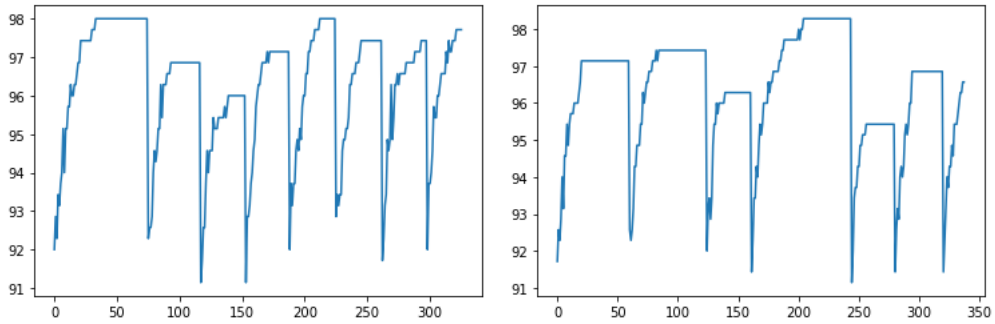

Fig 12. Plot of best fitness function in a generation vs no of iterations for 2 runs of the random restart algorithm (5s) with elitism.

Therefore, I fixed the **maximum no of restarts to 5**. This improved the running time of sentences with a higher m value considerably with no compromises in accuracy.

Changing population size:
A population size of <40 was too small and gave lower accuracy (but less time to for each iteration), while a population of greater than 150 took a long time to run but it converged with fewer iterations. A population of 500 consistently gave great accuracy and converged well, but took very long to run.
There was not much discernable difference between a population size in the 50-100 range which I deemed to be the ideal range for my algorithm to optimise on running time.

Varying mutation rate:
[These changes were tested in sentences with a lower no of clauses (m=100,150) which hit 100% fitness easily as it more apparent what led it to converge quickly.]
Changing the possibility of mutation to about 0.4-0.5 causes mutations to happen too quickly and lead to slower convergence rates. The same was true of lower mutation rates of 0.01-0.1 percent.
A mutation rate of about 0.2 or 0.25 was ideal.

Changing the number of mutating variables:
Changing the number of variables which change truth values in a mutation of the child beyond 15-20 led to slow convergence and lower initial accuracies because the child population was vastly different from the parent. It lead to a large number of low-accuracy plateaus when run without the random restart restriction.
Leaving the no of mutations as 1 or 2 was not very detrimental but convergence often happened faster with 5-8 mutated variables.
I finally selected 5 mutated variables.

3. Failed Approaches

1. **Population size too low** (<40) will lead to slow convergence and low accuracy, because of less variation in each generation and slow improvements.
Especially in sentences with a larger number of clauses, the average accuracy falls with lower population.
2. **Not having a limit on random restarts** led to sentences with higher number of clauses running for all 45 seconds with little improvements.
3. **Mutation rate too high** (>0.5) changed the generations too much to converge while a slow mutation rate (<0.2) did not introduce enough variety in the next generation.
4. Similarly, the **no of variables being mutated being too high** (>20) led to worse results and slow convergence.
5. **Changing the mixing number** or number of parents to 3 or 4 did not change the algorithm much.
6. **Changing the crossover point** or points had no effect on the algorithm.
7. **Randomly selecting the parents** (as opposed to a weighted random choice) led to slower convergence and worse accuracies.

4. Conclusions

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

A. The genetic algorithm failed to find a solution when the number of clauses become too high, compared to the number of variables.
This is because there are a decreasing number of correct solutions or models which satisfy the sentence as the number of clauses grow, despite the number of possible states being the same (2^50).
We conclude that the genetic algorithm fails or takes longer to find a solution when the **number of solutions are few** compared to the **total number of states.**

Q. 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? (Note: A CNF sentence is satisfied when all its clauses are satisfied.)

A. The graphs showed us that when m(number of clauses)>150 and n(number of variables)=50, the average accuracy of the fitness function was no longer 100% and fell quite quickly after that.
They also showed that as m increased while n stayed fixed, the algorithm took longer and longer to maximise the fitness function.
So as **m/n becomes >3**, the difficulty of satisfying the sentence increased sharply.

As the number of clauses grow while variables remain fixed, the probability of a model satisfying all clauses falls.

I conclude that for a 3-CNF sentence in n variables, **the difficulty of satisfying it increases** as the **ratio m/n increases**, where m is the no of clauses in the sentence.
Specifically, **as m/n increases past 3**, the difficulty of satisfying it increases quite sharply, ie, **satisfiability of the sentence falls from 1 to 0.**

5. References

1. Class Notes.
2. Russell, Stuart J., Peter Norvig, and Ernest Davis. Artificial Intelligence: A Modern Approach. 4th ed. pp. 129.