HW3 Report - SAT

Roșu Cristian-Mihai A2
Manolache Mihai A2

January 3, 2020

Abstract

This research paper analyzes the performance of a Genetic Algorithm against the Simulated Annealing heuristic search algorithm applied on the *Boolean Satisfiability Problem*(SAT).

1 Introduction

In logic and computer science, the Boolean satisfiability problem is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE. If this is the case, the formula is called satisfiable. On the other hand, if no such assignment exists, the function expressed by the formula is FALSE for all possible variable assignments and the formula is unsatisfiable. [1]

This paper aims to solve this problem by employing two different methods:

- using a Genetic Algorithm
- using a Heuristic Search Method, specifically Simulated Annealing

It is expected that the Genetic Algorithm will yield the best results as it generally outperforms the alternatives by a great margin. This assumption is made in correlation with the results of the previous experiments.

1.1 Motivation

The purpose of this work is to explore the capabilities of a Genetic Algorithm against those of a heuristic search method.

In that sense, SAT was chosen due to its ease of implementation and of understanding on a theoretical level. And as for the heuristic search method, Simulated Annealing includes the abilities of alternatives such as Iterated Hill-climbing while minimizing the danger of getting stuck in a local minima. On top of that, it is equally easy to understand and implement. As such, it is the perfect candidate.

2 Method

The CNF formulas which represent the input of SAT are being parsed into the algorithm from the DIMACS CNF format. [2]

The candidate solution for SAT has been represented identically in both algorithms: a bitstring of length equal to the number of variables in the CNF formula, the values of 1 and 0 representing TRUE and FALSE.

Since it would normally be impossible to compare SAT solutions, given the fact that they either satisfy the formula or not, we choose to evaluate a solution based on its number of TRUE clauses.

2.1 Genetic Algorithm

The genetic operators used are mutation and crossover, and as for selection, the **Roulette Wheel** method was used.

Since the solution is represented using a bitstring, the mutation and crossover operators suffered no modification in their implementation in the case of SAT.

The selection, however, was modified slightly due to the method used to calculate the fitness function. Specifically, the fitness of a cromozome is equal to the number of satisfied clauses.

Following experiment results, a maximum **number of generations of 500** was chosen since there was no real improvement in the solutions with bigger numbers. As for the other values, a **mutation rate of 0.01** and a **crossover rate of 0.3** were used.

2.2 Simulated Annealing

The function to maximize is derived from the input from the DIMACS CNF file, and the solutions are compared based on the number of TRUE clauses. Apart from that, the functionality of the algorithm remains the same.

In this case, a starting temperature of **100** was chosen with a cooling rate of **0.999**.

3 Experiment

The experiment consists in running the Genetic Algorithm and Simulated Annealing through 8 benchmarks of increasingly higher size [3], over 30 iterations in order to get a sample big enough to compare them by looking at the minimum, maximum, average and standard deviation of the solution produced, as well as the time needed to produce them.

The code for this can be found on Github [4].

4 Results

Below are 8 tables corresponding to the experiment results for each benchmark, and each benchmark has a different number of variables and clauses:

- *frb30-15-1.cnf* with **450 variables** and **19084 clauses**

Method	**Min**	**Max**	**Average**	**Sd**	**Time**
Genetic Algorithm	0	16513	14702.9756	436.2184	1702s
Simulated Annealing	13411	19077	18874.8352	696.4361	322s

Figure 1: frb30-15-1

Here, Simulated Annealing arrives at the better result, and faster, with a big margin compared to the Genetic Algorithm, being quite close to 19084.

- *frb35-17-1.cnf* with **595 variables** and **29707 clauses**

Method	**Min**	**Max**	**Average**	**Sd**	**Time**
Genetic Algorithm	0	25595	22555.34106	546.1537	2246s
Simulated Annealing	21527	29696	29474.2374	833.5987	404s

Figure 2: frb35-17-1

In Figure 2, Simulate Annealing proves once again to be the better match.

- *frb40-19-1.cnf* with **760 variables** and **43780 clauses**

Method	**Min**	**Max**	**Average**	**Sd**	**Time**
Genetic Algorithm	0	36655	33576.6104	788.1297	3632s
Simulated Annealing	7571	10673	10590.2980	312.9464	213s

Figure 3: frb40-19-1

From this point on, the Genetic Algorithm starts providing better and better results while Simulated Annealing drifts further and further away from the goal.

- *frb45-21-1.cnf* with **945 variables** and **61855 clauses**

Method	**Min**	**Max**	**Average**	**Sd**	**Time**
Genetic Algorithm	0	51173	47209.8882	896.0659	4401s
Simulated Annealing	20707	28504	28225.1825	907.1837	411s

Figure 4: frb45-21-1

- *frb50-23-1.cnf* with **1150 variables** and **84508 clauses**

Method	**Min**	**Max**	**Average**	**Sd**	**Time**
Genetic Algorithm	0	69589	64072.9148	1145.0336	6672s
Simulated Annealing	12797	18012	17767.8844	642.8460	347s

Figure 5: frb50-23-1

- *frb53-24-1.cnf* with **1270 variables** and **98921 clauses**

Method	**Min**	**Max**	**Average**	**Sd**	**Time**
Genetic Algorithm	0	81104	74952.3112	1450.3403	7772s
Simulated Annealing	23473	32215	31518.4353	1140.3133	646s

Figure 6: frb53-24-1

- *frb56-25-1.cnf* with **1400 variables** and **114668 clauses**

Method	**Min**	**Max**	**Average**	**Sd**	**Time**
Genetic Algorithm	0	94095	87121.6556	1430.3610	8763s
Simulated Annealing	10503	14879	14660.4627	532.6347	297s

Figure 7: frb56-25-1

- *frb59-26-1.cnf* with **1534 variables** and **132295 clauses**

Method	**Min**	**Max**	**Average**	**Sd**	**Time**
Genetic Algorithm	0	107362	100082.6476	1627.0163	9914s
Simulated Annealing	23400	31878	31087.7561	1160.8581	559s

Figure 8: frb59-26-1

And below are 2 pairs of graphs highlighting the evolution of the best solution produced by each of the 2 methods, which coresponds to the maximum value at each iteration, in the case of Simulated Annealing, and at each generation, in the case of the Genetic Algorithm.

The first pair showcases their behaviour on a small number of variables and clauses (frb30-15-1.cnf in Figure 1), while the second pair presents a benchmark with a significantly bigger number of variables and clauses (frb59-26-1.cnf in Figure 8).

The Y axis represents the range of values found and the X axis represents the temperature's evolution, specifically the number of times it took for it to sufficiently cool down, in the case of Simulated Annealing, and it represents the generation's number in the case of the Genetic Algorithm.

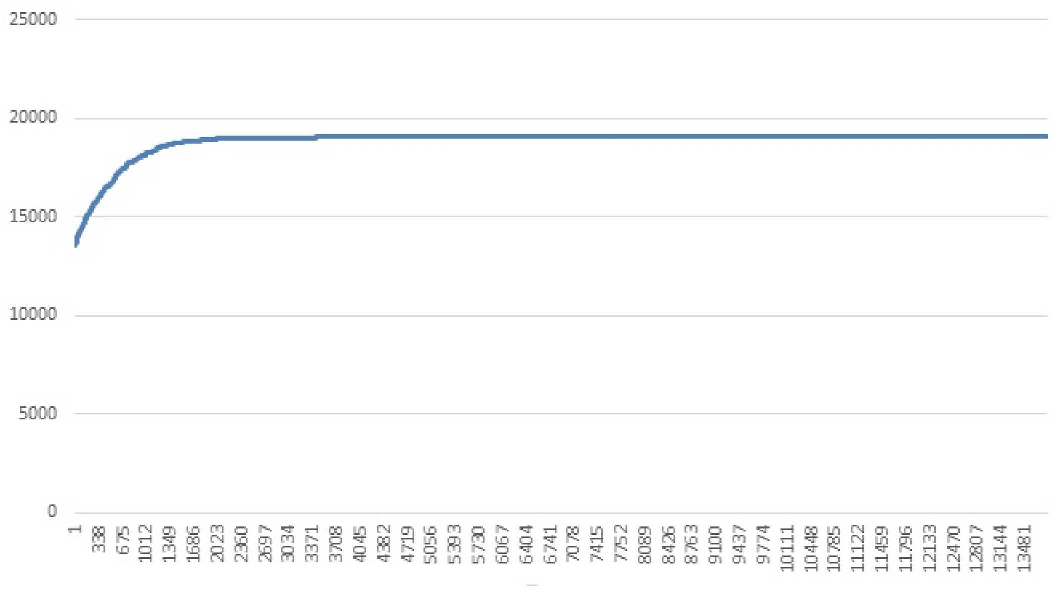

Figure 9: SA frb30-15-1.cnf

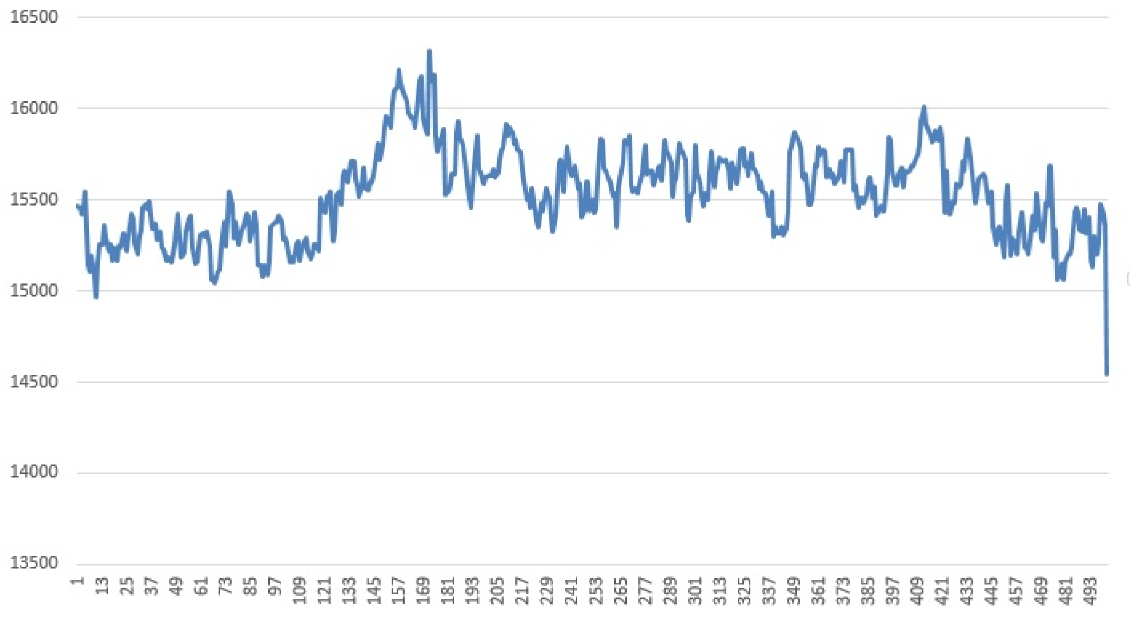

Figure 10: GA frb30-15-1.cnf

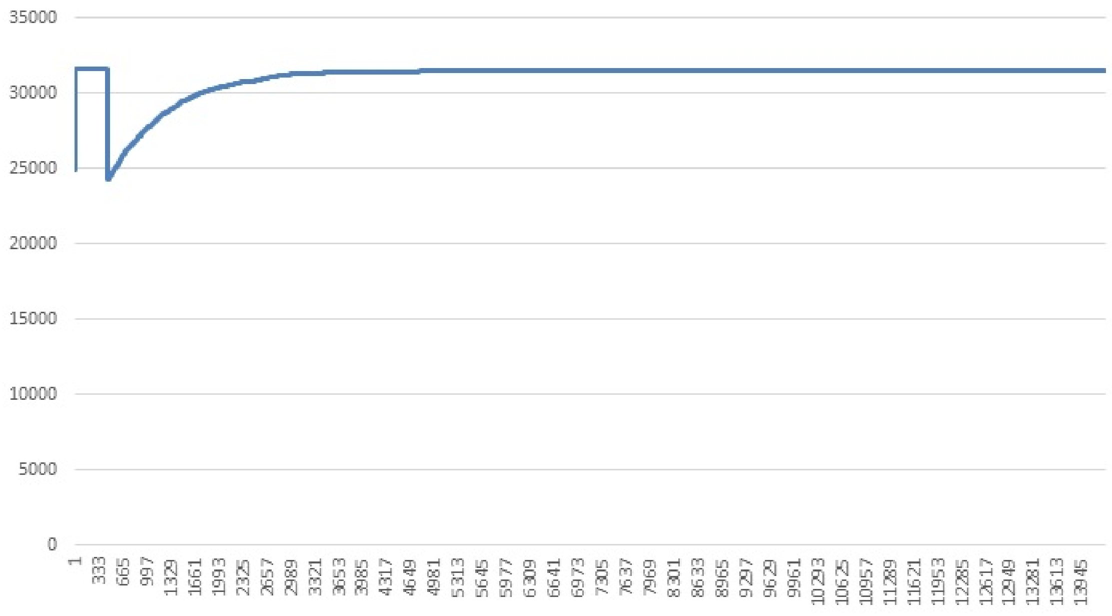

Figure 11: SA frb59-26-1.cnf

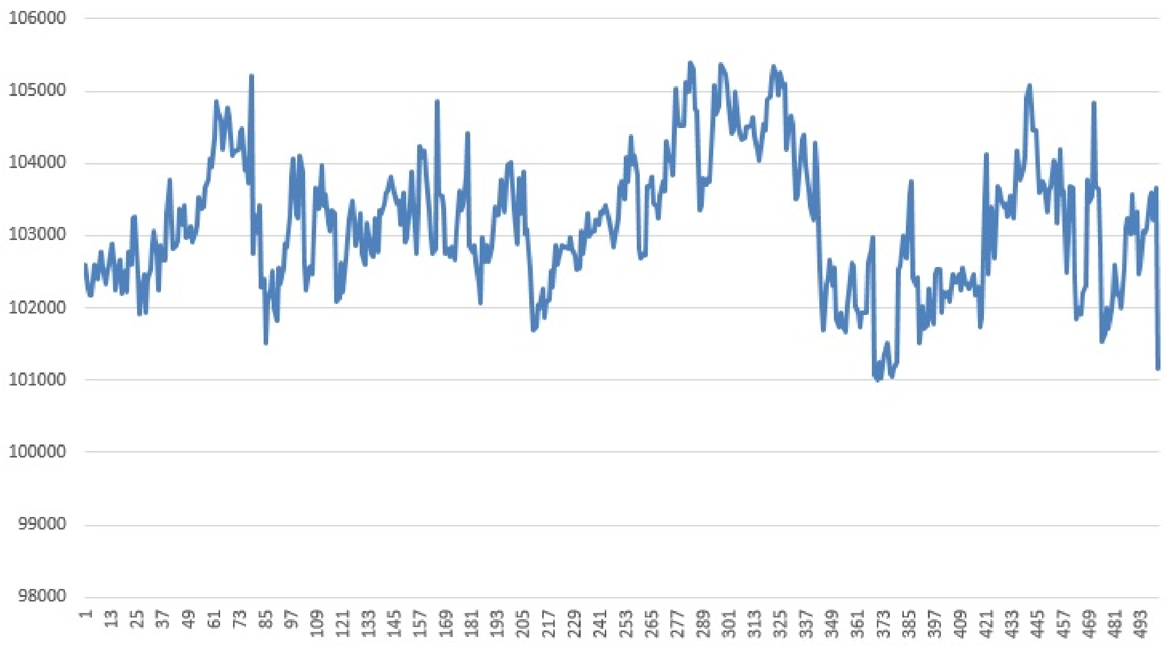

Figure 12: GA frb59-26-1.cnf

5 Conclusions

Seeing the results, Simulated Annealing outperforms the Genetic Algorithm when it comes to smaller samples. It arrives at a near-perfect solution in a very short time and with quite a margin, as seen in Figure 1 and 2.

However, from Figure 3 onwards, the Genetic Algorithm consistently outperforms Simulated Annealing with bigger and bigger differences.

From this, we can draw the conclusion that the Genetic Algorithm is the better alternative to this problem as it scales upwards.

It is worth mentioning that both methods could be improved in an unforeseen way that could potentially produce better results.

References

[1] Wikipedia page for SAT
`https://en.wikipedia.org/wiki/Boolean_satisfiability_problem`

[2] DIMACS CNF format definition
`https://logic.pdmi.ras.ru/~basolver/dimacs.html`

[3] Site with benchmarks used
`http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/benchmarks.htm`

[4] Github repository for the project
`https://github.com/Nenma/ga-hw3`