SAT solver using genetic programming

Nguyen Xuan Thang
CTU - FIT
nguyexu2@fit.cvut.cz

4. ledna 2023

1 Introduction

The goal of the work was to implement and study the options of an efficient SAT solver using genetic programming. The program needs to find a binary evaluation of the variables, such that the given SAT formula is satisfied when using this evaluation. All formulas will be in CNF, meaning we have an NP-complete problem that we will try to solve using an iterative heuristic method. To map an iterative method to solve a decision problem, we will formulate the SAT problem as MaxSAT, meaning find values of variables such, that the maximum number of clauses are satisfied.

2 Data

All data are taken from the SATLIB library[1]. This library of different SAT formulas contains many hard and easy instances to test the solver. It also contains many instances that encode real-life problems and are not limited to only randomly generated data, such as planning problem or graph coloring. This library is widely used to compare the strength of different SAT solvers and is often included in many SAT solving competitions.

All instances are encoded in DIMACs format, for this reason, no further preprocessing was needed on the data itself. To train the models, the uf20, uf50, uf100 and uf200 instances were chosen. The number represents the number of variables in the formula. To test the model, instances from section DIMACS Benchmark instances were used.

3 Methods

To solve the SAT instances, a genetic algorithm was implemented inspired by [2]. Each chromosome is defined as a binary bit vector, where each element represents the binary truth value of a variable. The population is then a set of such bit vectors and to select the next generation, a roulette wheel selection approach was chosen. The better individuals will have a larger probability of being chosen and the fitness of an individual is calculated as the number of clauses being solved using this evaluation.

To explore the search space, 2 operators were implemented. The binary operator takes 2 chromosomes and applies 1-point crossover. The mutation process randomly flips k-bit of the chromosome, where k is a parameter.

The main parameters that we will train are the population size and the number of generations allowed to search the search space. While having bigger values for these parameters, we can expect better results, but the time needed to find the solution will also be larger. Another parameter that will be tuned is the elitism rate and the mutation rate. A high elitism rate should enforce convergence toward the solution whereas a high mutation rate will allow more diversification.

4 Results

The parameters were first tuned on very simple data to see the correlation between the size of the population and the number of generations, compared to the time needed to find the solution.

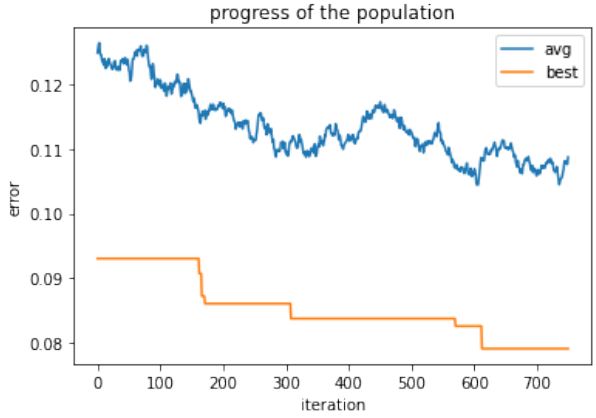
As we can predict, with more allowed iterations and a more diverse population, the solution gets better. Though after 500 allowed generations, the improvement was not as significant, but the time needed to process the iterations was still rising. Compared to the extra time needed to process all iterations, the error improvement was not noticeable. The optimal number of iterations was around 750.

Another parameter that was tested was the elitism rate. Surprisingly, the elitism rate did not have a significant impact on the result. The average error stayed almost the same no matter the elitism rate. As for the mutation rate, through measuring, it was found out, that with a slower more careful approach of flipping bits (meaning lower mutation rate), better results could be yielded. This should not be surprising, as we want to enforce more local search around the optimum instead of jumping too far away.

Finally, we can see the improvement of the final result through each iteration in graph 1. The initial best result is quite good but the improvement over

instance	variables	clauses	error	time[s]
uf20	20	91	0.001	0.78
uf50	50	218	0.026	3.52
uf100	100	430	0.05	6.96
uf200	200	860	0.072	15.59
CBS	100	423	0.04	6.90
hanoi4	718	4932	0.130	59.82
hanoi5	1931	14468	0.16	181.47
ii	-	-	0.419	69.72

Tabulka 1: Time needed to solve instances. ii instances have different number of variables and clauses.



Obrázek 1: Progress of instance uf200-860 over time.

time is not as noticeable. It would be worth it to study better operators that could guarantee the satisfaction of some chosen clauses. The average error of the whole population as expected is decreasing. Some spikes in the average is also desired, as we want to allow the algorithm to break out from a local optimum and allow wider search space.

5 Závěr

A variation of genetic algorithm was implemented for SAT. It yielded relatively good results, with the error being only around 5%. There is still some room for improvement regarding the implementation of the selection process. It would be interesting to see, if an island model would help with this process. Lastly, it would be very interesting to see if there could be some ways to create a model that could reduce the formula size while keeping solveability.

Reference

[1] Satlib library. [cit. 2022–26–12] `https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`.

[2] Arunava Bhattacharjee and Prabal Chauhan. Solving the SAT problem using genetic algorithm. *Advances in Science, Technology and Engineering Systems Journal*, 2(4):115–120, August 2017.