Experimental analysis and comparison of GSAT and ProbSAT algorithms

Narek Vardanjan

October 28, 2022

1 Executive summary

The experiment descibed in this document compares 2 different algorithms for solving the 3-SAT problem. These algorithms are `prob_sat` and `gsat` and they are from the group of non-complete algorithms. It tries to answer question.

Is ProbSAT better algorithm for solving 3-SAT hard instances than GSAT for the given parameters p, c_m, c_b?

2 Introduction

The experiment is trying to answer if `prob_sat` is faster than `gsat` given 3-SAT instances that are hard (have the clauses-to-variables ration near `4.3`). These instances were downloaded from the `SATLIB` benchmarks. Parameters used to answer the experiment were as follows.

- `MAX_TRIES` - $(300, 400, 500)$
- `MAX_FLIPS` - $(20, 50, 100, 120)$
- `gsat` $p = 0.4$
- `prob_sat` $c_m = 0, c_b = 2.3$

To get rid of the bias, multiple values for `MAX_TRIES` and `MAX_FLIPS` were measured. Additionally, a new dataset was created from `SATLIB` benchmarks that has a mixture of variable and clause numbers to further decrease possible bias in the data. Dataset is available in the `dataset/` folder.

3 Material

`GSAT` implementation given on the lectures was used to generate solutions to instances. `Prob_sat` implementation was written in python from scratch using the same API. Both implementations ran instances from the `dataset`, each one for 15 times with every combination of `MAX_TRIES` and `MAX_FLIPS`, so that it is possible to average anomalies. Script for running the `csv` generation is called `gen_csv.sh` and is part of the source code. Results were then saved as a csv and sample from that csv could be seen bellow. Full data is available in the `res.csv` file.

iter	filename	alg	max_tries	max_flips	solved	num_vars
15	uf50-0853.cnf	gsat	300	20	True	50
39	uf50-0851.cnf	prob_sat	500	20	True	50
177	uf50-0369.cnf	prob_sat	400	20	True	50

iter	filename	alg	max_tries	max_flips	solved	num_vars
10	uf20-0433.cnf	prob_sat	300	20	True	20
31	uf50-0636.cnf	gsat	500	50	True	50
2385	uf75-057.cnf	prob_sat	400	120	True	75
672	uf20-0897.cnf	gsat	400	100	True	20
6199	uf75-059.cnf	gsat	300	120	True	75
56	uf20-0433.cnf	prob_sat	400	50	True	20
16	uf20-0772.cnf	prob_sat	300	50	True	20
1048	uf75-066.cnf	gsat	300	120	True	75
994	uf20-0533.cnf	gsat	300	100	True	20
1350	uf75-033.cnf	prob_sat	300	50	True	75
18	uf50-043.cnf	gsat	500	100	True	50
59	uf20-0969.cnf	prob_sat	400	100	True	20

Some runs didn't find solution to the problem. Those runs were penalized by multiplying their number of iterations by 10. Sample of runs that did not finish after a number of tries specified can be seen below.

`number of unsolved runs 306`

iter	filename	alg	max_tries	max_flips	solved	num_vars
1000000000	uf75-089.cnf	prob_sat	500	20	False	75
600000000	uf75-042.cnf	prob_sat	300	20	False	75
800000000	uf75-070.cnf	prob_sat	400	20	False	75
600000000	uf75-042.cnf	prob_sat	300	20	False	75
800000000	uf75-070.cnf	gsat	400	20	False	75

Data acquired from the gathering step were then averaged accross `filename`, `algorithm`, `max_tries`, `max_flips` and `num_vars` columns. Resulting dataset has only 240 rows total, with `iter` equal to the average of the 15 runs that were done per configuration.

4 Results

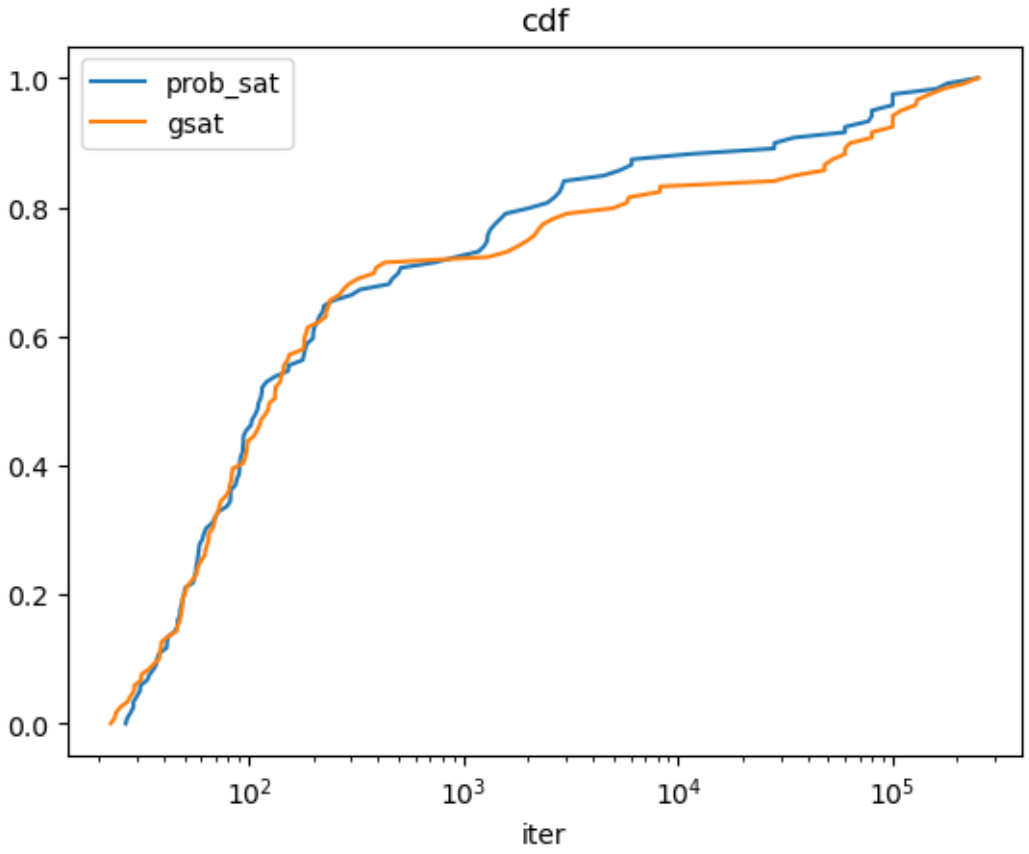
In this section preprocessed data is going to be presented in the human interpretable form, using metrics.

First metric that's going to be considered is the `mean` number of iterations. It's available in the table below. `Prob_sat` is takes lesser number of iterations than `gsat` on average.

algorithm	mean
prob_sat	11737.7
gsat	17255.9

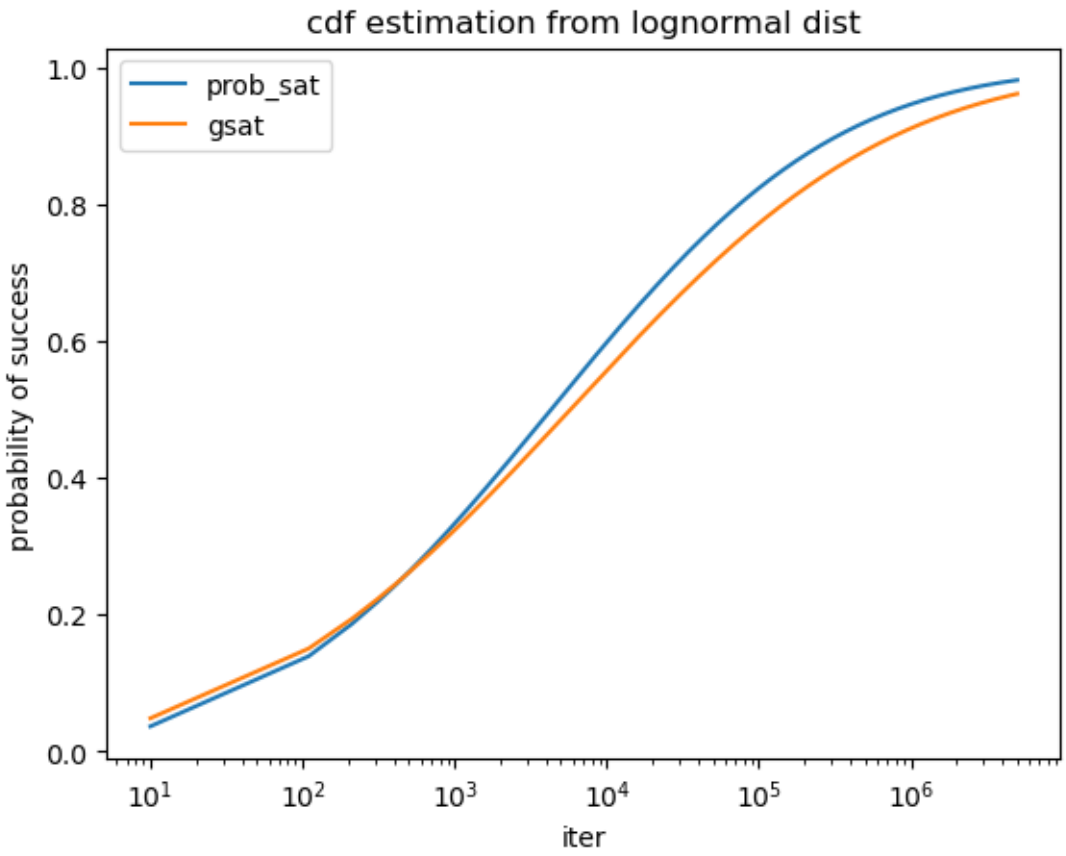
Next let's look at the cumulative distribution function (`cdf`) for both algorithms. X axis is converted to logarithmic scale to enhance clarity. No algorithm is systematically better, which is apparent

from multiple intersections points of the probabilities at the start. From one point onward it seems that `prob_sat` is more succesful in finding solutions given enough number of iterations.



Last metric that was not discussed in the lectures is expected log normal distribution `cdf`. The μ and σ values of log normal distribution were used to plot empirical `cdf`. Estimated moments are shown in the table and from the `cdf` we can again see that `prob_sat` is better than `gsat`.

algorithm	mu	sigma
gsat	8.661	3.828
prob_sat	8.371	3.385



5 Discussion

There is not a systematically better algorithm, which is apparent from the intersections in the `cdf`s. Given enough iterations it seems that `prob_sat` is better at finding solutions. Conclusion is that empirically `prob_sat` seems to be better than `gsat` at solving hard SAT-3 instances. So the final verdict to the question is. **Yes, ProbSAT seems to be better at solving hard 3-SAT instances given parameters** p, c_m, c_b.