Experimental analysis and comparison of GSAT and ProbSAT algorithms

Narek Vardanjan

October 29, 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 not complete algorithms. It tries to answer a question.

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

Repository with the source code is available at https://github.com/vnarek/prob-sat-vs-gsat

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 the 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 `create_dataset.sh` was used for `dataset` creation. Both scrips are available in 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
6198	uf75-080.cnf	gsat	400	50	True	75
115	uf20-0564.cnf	prob_sat	500	50	True	20
10000	uf75-094.cnf	gsat	500	20	False	75
22	uf20-0392.cnf	prob_sat	300	120	True	20
22	uf20-0897.cnf	prob_sat	400	100	True	20
10000	uf75-089.cnf	gsat	500	20	False	75
95	uf50-0369.cnf	gsat	300	50	True	50
324	uf20-0468.cnf	gsat	500	100	True	20
179	uf20-0897.cnf	gsat	400	100	True	20
7	uf20-0327.cnf	prob_sat	500	100	True	20
10	uf20-0738.cnf	prob_sat	300	20	True	20
9	uf20-0897.cnf	gsat	300	120	True	20
80	uf20-0130.cnf	prob_sat	400	120	True	20
145	uf50-0369.cnf	gsat	300	50	True	50
113	uf20-0339.cnf	prob_sat	300	100	True	20

Not every run found a solution to the problem. Those that didn't 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
300000	uf75-070.cnf	gsat	300	100	False	75
80000	uf75-070.cnf	gsat	400	20	False	75
480000	uf75-012.cnf	prob_sat	400	120	False	75
480000	uf75-012.cnf	gsat	400	120	False	75
100000	uf75-022.cnf	prob_sat	500	20	False	75

Runs acquired from the gathering step were then grouped by `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.

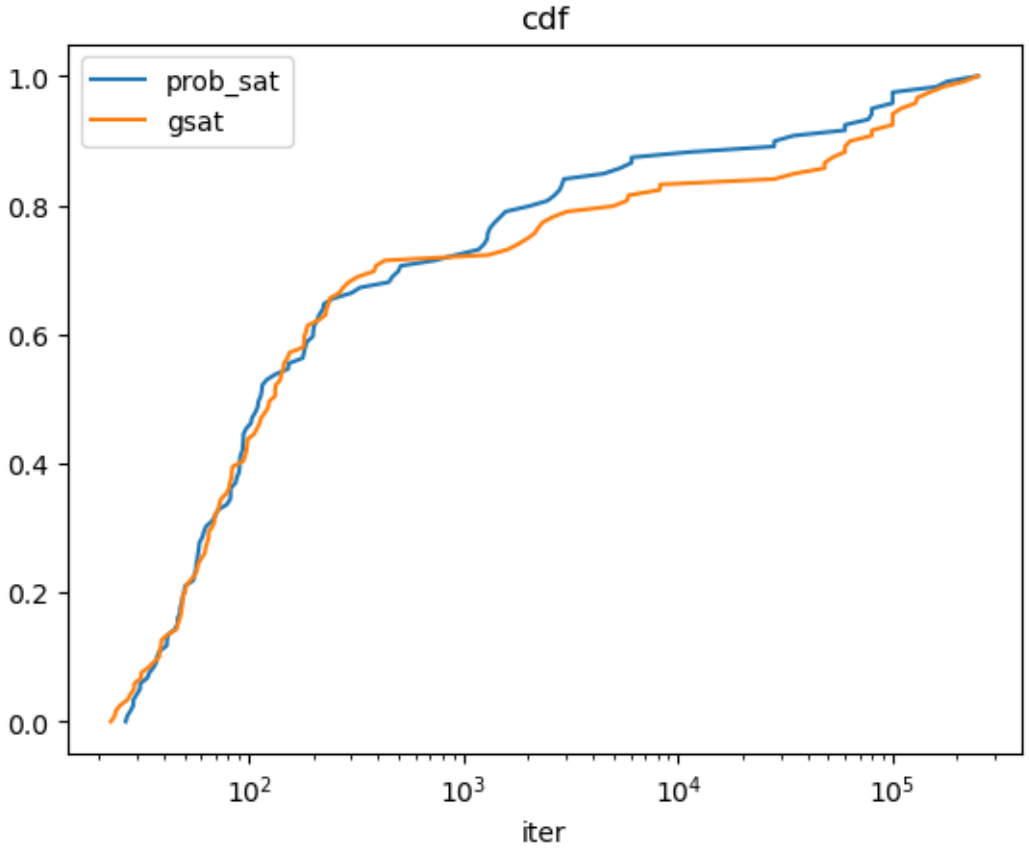
4 Results

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. The reason for picking this metric is it's simplicity and ease of explainability. It shows, that `Prob_sat` 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 intersection points of the probabilities at the start. From one point onward it seems that there is a higher probability of finding solutions given number of iterations using the `prob_sat` algorithm. This metric was chosen, because it can be interesting to look at the development of the `cdf`.



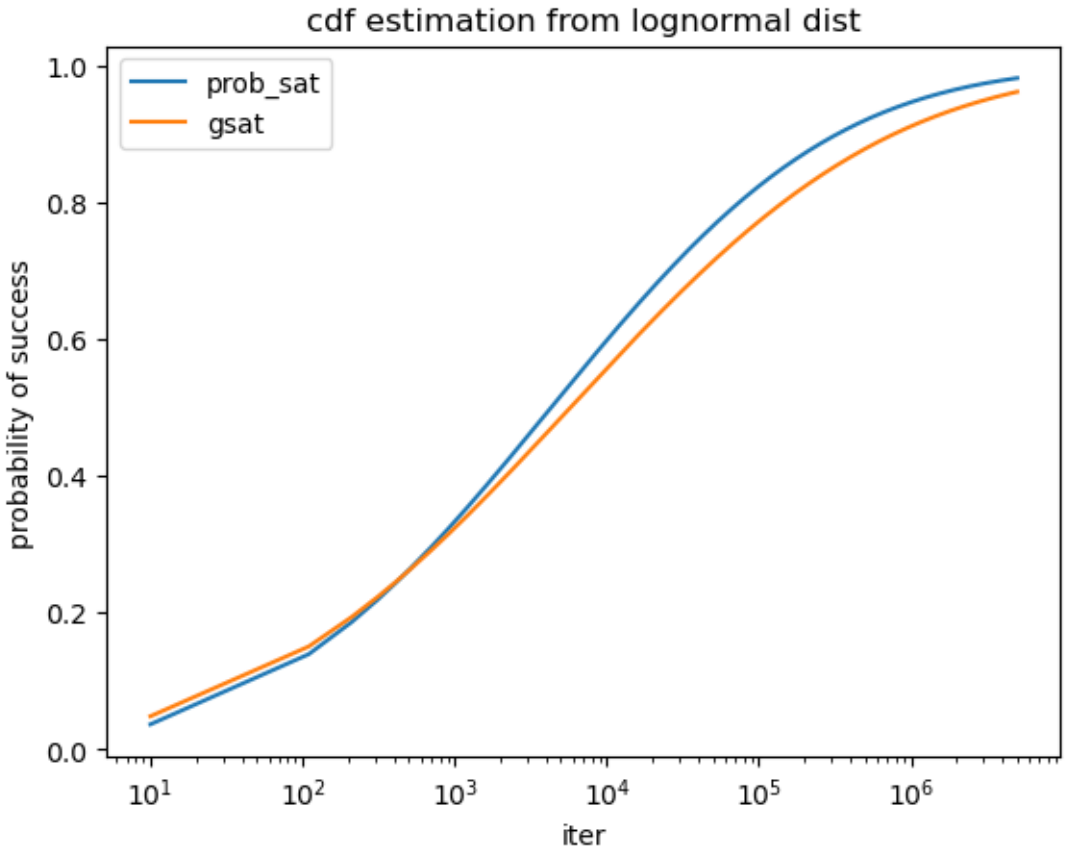
Last metric, that was not discussed in the lectures is expected log normal distributions `cdf`. The μ and σ values of log normal distribution were used to plot the theoretical `cdf`. They were computed using these formulas.

$$\hat{\mu} = \frac{\sum_{i=1}^{n} log\ x_i}{n}$$

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n} (log\ x_i - \mu)^2}{n-1}}$$

Estimated moments are shown in the table and from the `cdf` we can again see that `prob_sat` is better than `gsat` in the long run. This metric is similar to the normal `cdf` shown above, but there is more clarity in the results.

algorithm	mu	sigma
prob_sat	8.371	3.385
gsat	8.661	3.828



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. 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.