EXPERIMENTAL EVALUATION OF THE ALGORITHMS PROBSAT AND GSAT

Andrei Shchapaniak

October 27, 2023

Contents

Executive summary

In this experiment, the GSAT and probSAT algorithms were compared on sets of instances with variables ranging from 20 to 75 and clauses between 91 and 325. The attention was paid to key metrics, including the number of instances solved, steps taken for solved problems, and penalized steps for all instances.

After comparing these measured metrics between the algorithms, it became clear that the probSAT algorithm is better than the GSAT algorithm in all evaluated scenarios.

1 Introduction

In the context of the (Boolean satisfiability problem) SAT problem, this experiment compares the GSAT and probSAT algorithms on 3-SAT instances with 20-75 variables to determine which converges faster. Further details are provided in the assignment here[1].

The main motivation for this research was to gain a deeper understanding of the efficiency and effectiveness of these algorithms, especially when faced with problems of different complexities.

The hypothesis was centered on the idea that notable performance differences might exist between the two algorithms across various problem instances.

2 Methods

The implementation of the GSAT algorithm was utilized from the study materials available at `https://courses.fit.cvut.cz/NI-KOP/download/index.html`. As per the requirements, the algorithm is run with fixed parameters, specifically **p** = 0.4, which determines the probability that a random step will be taken in a given iteration.

The probSAT algorithm's implementation was obtained by modifying the aforementioned GSAT algorithm. In each iteration, this algorithm chooses a random literal from an unsatisfied clause at random. The probability of selecting a literal is based on the number of clauses that would be newly satisfied or unsatisfied upon toggling the value of the literal. The algorithm is executed with fixed parameters: **cm** = 0 and **cb** = 2.3. Here, the parameter cm determines the significance of the count of newly satisfied clauses after the literal's value is flipped, while cb affects the significance of the count of newly unsatisfied clauses after the flip [1].

Input data

The 3-SAT problem instances for measurement were obtained from the study materials available on the page[2]. The following sets from the SATLIB library were used in the experiment.

- `uf20-91R` - 20 variables, 91 clauses, 100 instances.
- `uf50-218R` - 50 variables, 218 clauses, 100 instances.
- `uf75-325` - 75 variables, 325 clauses, 100 instances.

[1] `https://courses.fit.cvut.cz/NI-KOP/homeworks/files/task1.html`
[2] `https://courses.fit.cvut.cz/NI-KOP/download/index.html`

Metrics

In the realm of algorithm analysis, the selection of appropriate metrics is crucial. These metrics offer a quantitative measure of an algorithm's performance. By evaluating algorithms using these metrics, we can make informed decisions about which algorithm solves instances more efficiently.

The following are the key metrics have been employed to compare the algorithms:

- `Number of solved problems.`
- `Average number of fined steps across all instances.`
- `Average number of steps for solved problems.`

Experiments

Both algorithms were run 500 times for each combination of parameters. The input parameters are shown in Table 1 below:

	UF20-91R		**UF50-218R**		**UF75-325**	
	repetitions -T	**iterations -i**	**repetitions -T**	**iterations -i**	**repetitions -T**	**iterations -i**
1.	1	300	5	300	5	500
2.	3	300	10	300	10	500
3.	5	300	1	500	3	1000
4.	1	500	3	500	5	1000
5.	3	500	5	500	1	3000
6.	1	1000	1	1000	3	3000
7.	-		3	1000	1	5000

Table 1: Combinations of parameters for each set of instances.

The choice of parameters was informed by insights gained from university practical lessons and small local tests, aiming to emphasize the outcomes of the algorithm runs.

The entire process, from executing the two algorithms to calculating statistics and plotting graphs, was automated using Python. This approach offers flexibility in making adjustments to the programs. All output files were saved in CSV format, making them easily accessible for analysis in applications like Excel.

3 Results

UF20-91R

Configuration	Solved		Total	Awg fined steps		Awg steps for solved	
	probSAT	GSAT		probSAT	GSAT	probSAT	GSAT
-T 1 -i 300	47303	44522	50000	216	383	60	64
-T 3 -i 300	49857	49321	50000	103	220	78	101
-T 5 -i 300	49987	49839	50000	83	158	79	110
-T 1 -i 500	49093	47033	50000	158	371	70	83
-T 3 -i 500	49980	49727	50000	87	196	81	116
-T 1 -i 1000	49874	49019	50000	104	298	79	107

Table 2: Merged statistics for uf20-91R set.

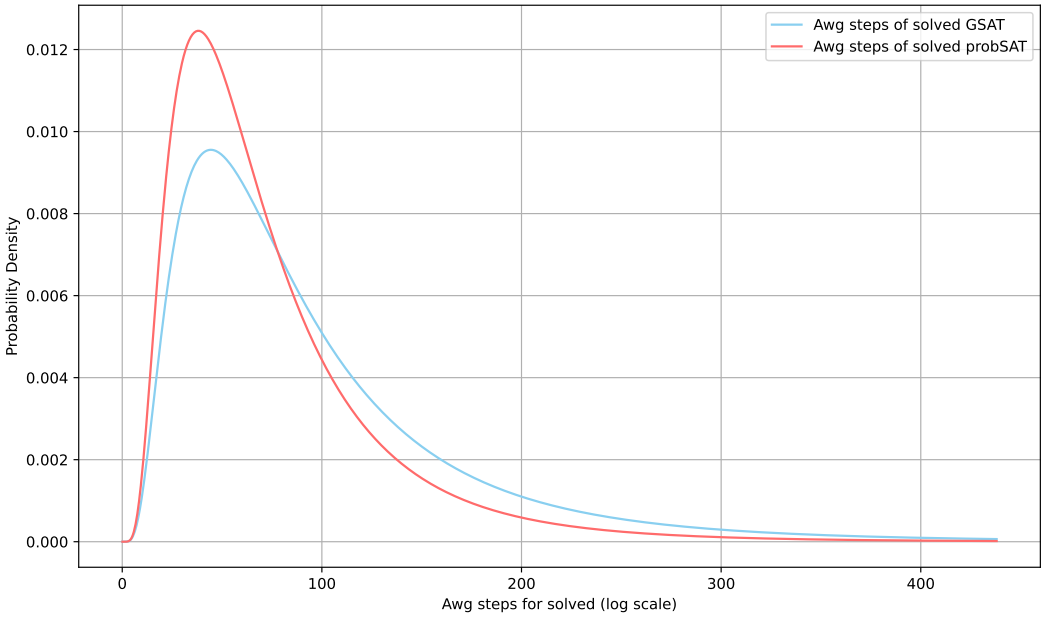

Figure 1: Log-normal distribution of average steps for solved instances from the uf20-91R set.

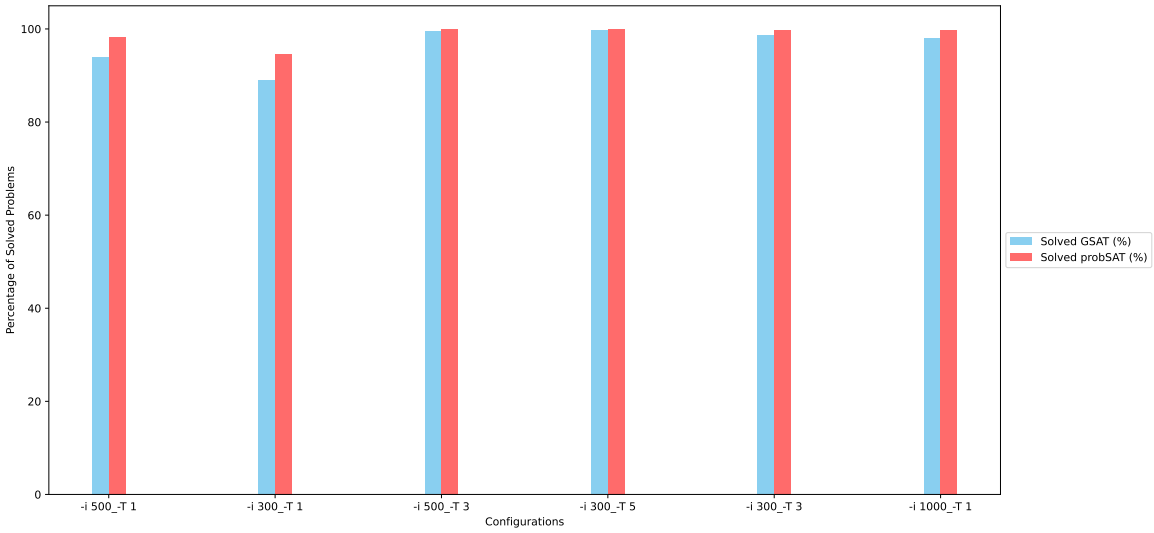

Figure 2: Histogram representing the ratio of solved to total instances for each configuration in the uf20-91R set.

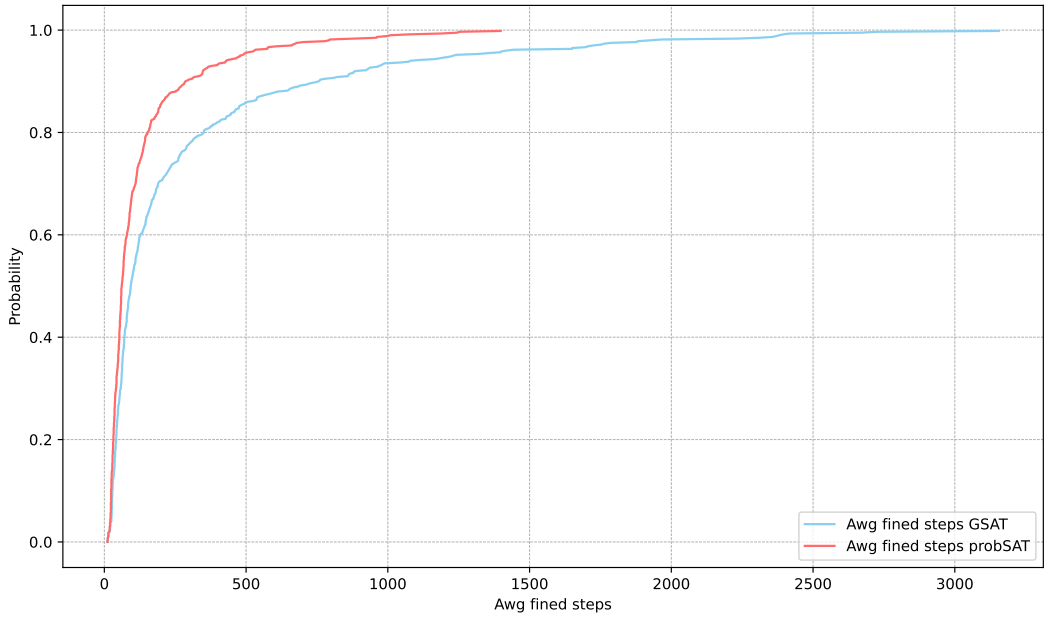

Figure 3: Empirical cumulative distribution function for the average number of fined steps from the uf20-91R set.

UF50-218R

Configuration	Solved		Total	Awg fined steps		Awg steps for solved	
	probSAT	GSAT		probSAT	GSAT	probSAT	GSAT
-T 5 -i 300	43902	40249	50000	2173	3280	418	475
-T 10 -i 300	47896	45967	50000	1773	3019	557	691
-T 1 -i 500	31732	24750	50000	1952	2620	200	210
-T 3 -i 500	43959	39182	50000	2152	3592	413	477
-T 5 -i 500	47221	43961	50000	1856	3553	520	652
-T 1 -i 1000	40117	32055	50000	2222	3789	327	334
-T 3 -i 1000	47746	44140	50000	1850	4119	547	734

Table 3: Merged statistics for uf50-218R set.

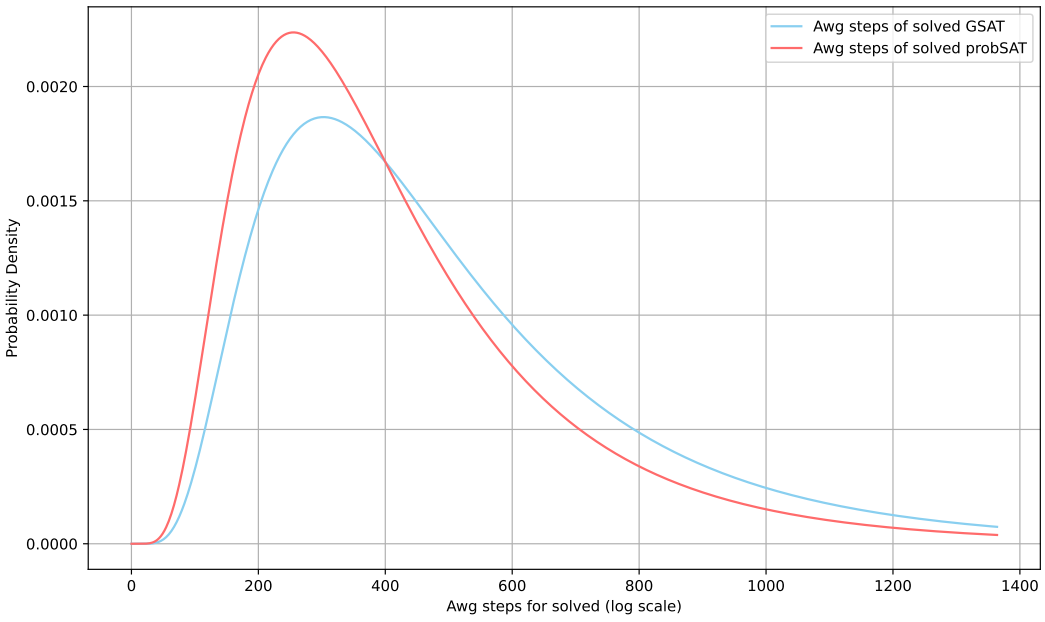

Figure 4: Log-normal distribution of average steps for solved instances from the uf50-218R set.

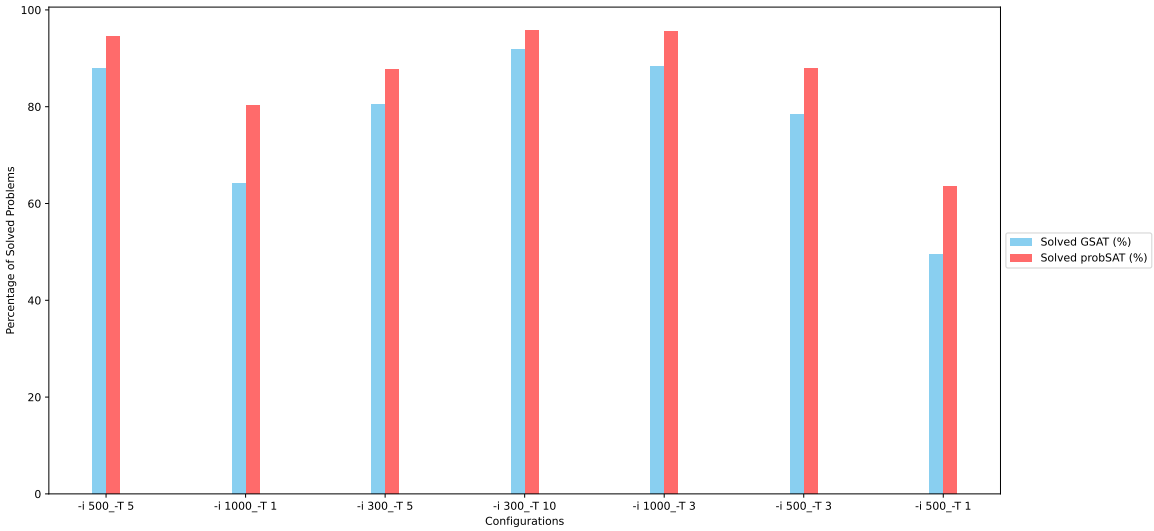

Figure 5: Histogram representing the ratio of solved to total instances for each configuration in the uf50-218R set.

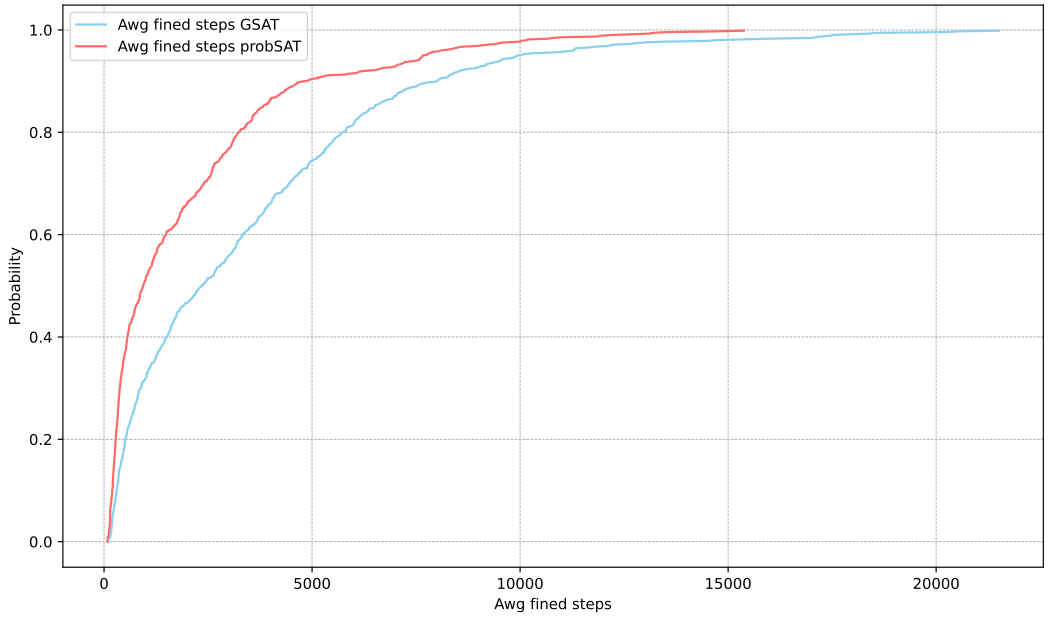

Figure 6: Empirical cumulative distribution function for the average number of fined steps from the uf50-218R set.

UF75-325

Configuration	Solved		Total	Awg fined steps		Awg steps for solved	
	probSAT	GSAT		probSAT	GSAT	probSAT	GSAT
-T 5 -i 500	39264	33824	50000	5974	8678	840	946
-T 10 -i 500	45437	41930	50000	5592	9239	1208	1500
-T 3 -i 1000	41825	35101	50000	5607	9631	906	1070
-T 5 -i 1000	45817	41348	50000	5197	9812	1176	1597
-T 1 -i 3000	41775	32056	50000	5617	11351	878	987
-T 3 -i 3000	48278	44115	50000	4430	12391	1439	2181
-T 1 -i 5000	45542	36870	50000	5440	14092	1153	1412

Table 4: Merged statistics for uf75-325 set.

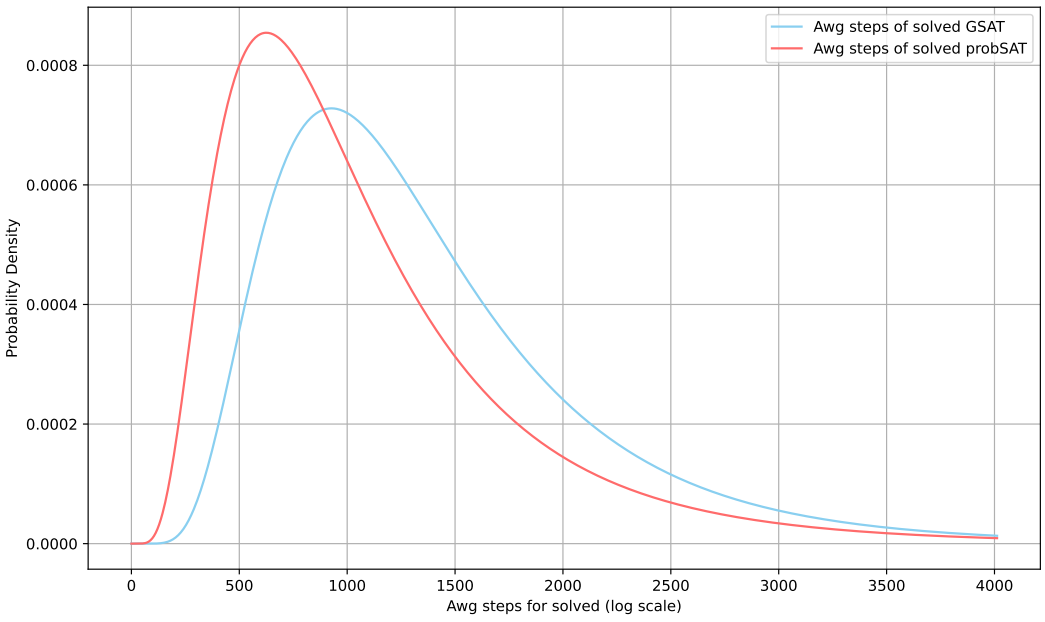

Figure 7: Log-normal distribution of average steps for solved instances from the uf75-325 set.

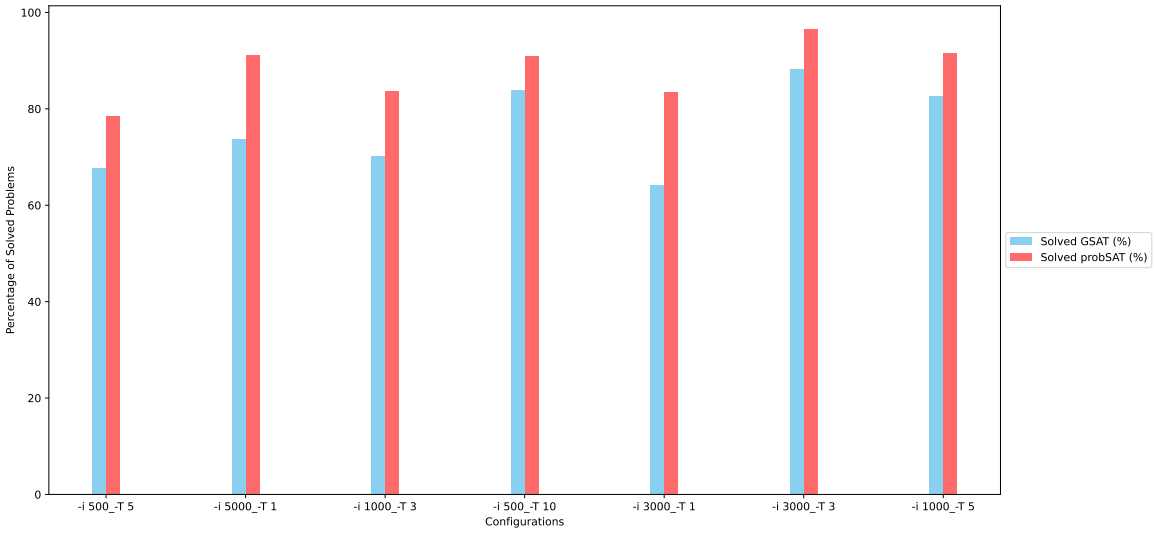

Figure 8: Histogram representing the ratio of solved to total instances for each configuration in the uf75-325 set.

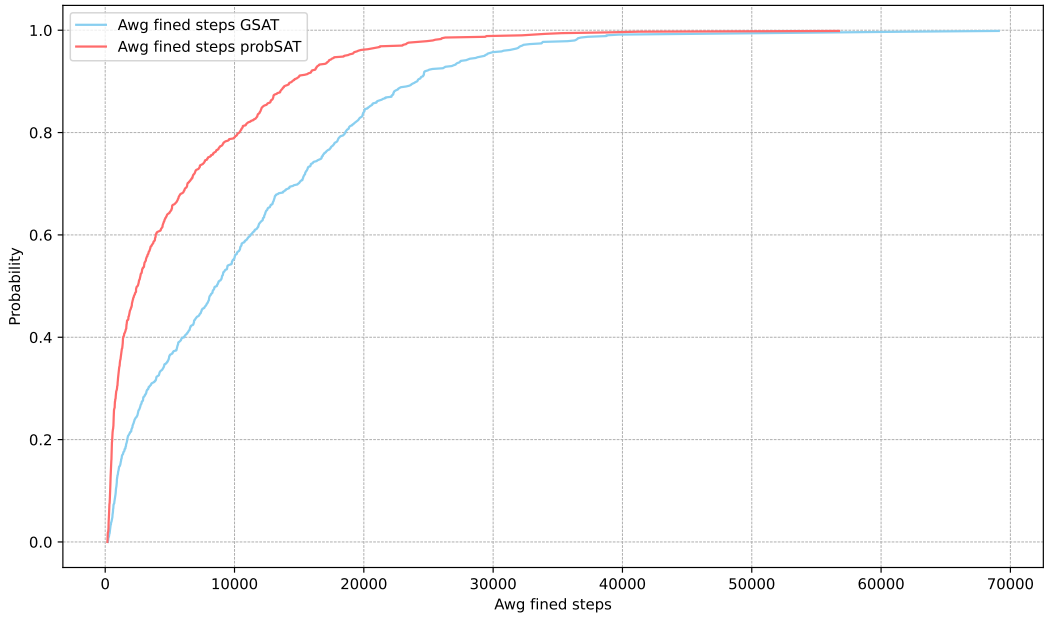

Figure 9: Empirical cumulative distribution function for the average number of fined steps from the uf75-325 set.

4 Discussion

In each table, when focusing on pairs of columns where both algorithms were executed with the same parameters on the same dataset, there's a noticeable difference in their metrics. It shows that the probSAT algorithm outperforms the GSAT algorithm.

Furthermore, as the complexity of the tasks escalates, the difference in the results of many table columns grows. This underscores that probSAT handles problems of various complexities effectively, while GSAT finds it more challenging.

Graphs

The figures 3, 6, and 9 present the empirical cumulative distribution functions for the average iteration counts of the algorithms. The penalty is counted for unsolved problems multiplying the maximum possible steps by 10. In all three figures, probSAT consistently converges faster across all dataset instances. The absence of curve intersections further confirms that probSAT consistently outperforms GSAT.

The histograms in 2, 5, and 8 highlight the number of problems each algorithm solved. Clearly, probSAT solves more instances across all parameter combinations for the three datasets.

Lastly, the graphs 1, 4, and 7 depict the distribution of average step counts for only the solved problems. These distributions appear to follow a log-normal pattern.

Summary

Based on all the measured metrics, whether from tables or graphs, it's clear that in this experiment with certain parameters mentioned in Section 2, the probSAT algorithm is superior to the GSAT algorithm.

While we could explore additional metrics for deeper insights, it's worth noting that there may be scenarios where GSAT could excel. However, in our current observations, probSAT has shown to be more efficient, especially with harder tasks, solving them faster.

It's crucial to recognize the inherent differences between GSAT and probSAT. Their distinct mechanisms mean that future tests should account for both the specific parameters and the fundamental contrasts between the two. This understanding will guide more precise evaluations in subsequent research.

References

[1] A. Balint and U. Schöning, "Choosing probability distributions for stochastic local search and the role of make versus break," in *Theory and Applications of Satisfiability Testing – SAT 2012*, A. Cimatti and R. Sebastiani, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 16–29. [Online]. Available: https://doi.org/10.1007/978-3-642-31612-8_3