Bima Journal of Science and Technology, Vol. 8(4B) Jan, 2025 ISSN: 2536-6041

DOI: 10.56892/bima.v8i4B.1166

Comparison of the Empirical Performance of Greedy, Hill Climb and Simulated Annealing Algorithms in GSAT Solver on DIMAC and Aloul Benchmarks

A. J. Kawu[1*], G. M. Wajiga[2] ,Y. M. Malgwi[2] and Usman Mohammed[2]

[1]Department of Computer Science, Faculty of Science, Gombe State University, Gombe, Nigeria
[2]Department of Computer Science, Modibbo Adama University Yola, Adamawa State, Nigeria

Corresponding Author: ahmadkawujibir@gsu.edu.ng

ABSTRACT

Boolean satisfiability solvers have made a significant impact in different areas of research such as Mathematics, Computer Science (artificial Intelligence, Automatic Test Pattern Generation, and so on). The problem to determine whether a given Boolean formula has a model was proved to be NP complete by Cook in 1971. This area witness a number of research over the past 5 decades. A number of algorithms were developed for solving SAT problem. These algorithms showed varying performance on different categories of SAT instance (random, industrial and handcrafted). This paper proposed to compare three (3) options of the Stochastic Local Search solver GSAT(Greedy SAT Solver). The options are Greedy, Hill Climb and Simulated Annealing. This work was inspired by the need to provide an insight on their performance on different problem Instances to serve as a guide for researchers who are interested in developing hybrid decision heuristic, initialization of variable weights in a decision heuristic and portfolio SAT solver. We empirically compared the performance on the three options on a Dell latitude E7470 laptop. Our result showed that Hill Climb option outperformed the other options in a number of problem instances solved while Simulated Annealing performed large number of downward and sideways moves. This implies that Hill Climb is the best choice for a hybrid solver while Simulated Annealing is preferable for initialization of variable weights.

Keywords: Satisfiability, Hill Climb, Simulated Annealing, Stochastic Local search, Systematic search

INTRODUCTION

Boolean Satisfiability (SAT) is one of the main issues in mathematics and computer science that is generally thought to be untractable. Since Cook demonstrated in 1971 that this problem is NP-complete (Cook, 2023), theorists have examined it in great detail (Ganesh & Vardi, 2020). Many intriguing SAT examples can be solved effectively in practice utilizing heuristic algorithms, despite the general belief that SAT is difficult in the worst situation. The research community has found a wide range of distinct heuristic methods over the years, essentially by discovering heuristics that perform well on various SAT instances, but also in part by significantly improving on earlier concepts (Ghanem, & Siniora, 2021). The goal of the Boolean satisfiability problem, a traditional combinatorial optimization problem, is to ascertain if a given Boolean formula can be made true by a set of variable assignments (Reifenstein etal, 2023).The major algorithms used for solving Boolean satisfiability problems are the complete algorithms (based on DPLL) and incomplete algorithms (Stochastic local search algorithms (SLS)). Local search algorithms are an attractive choice for solving the boolean satisfiability problem because they scale better with input size than systematic algorithms (Cohen et al, 2021). Among the most well-known techniques for SAT problem solving at the moment are SLS algorithms. Some incomplete SAT solvers are surprisingly more

DOI: 10.56892/bima.v8i4B.1166

successful than state-of-the-art systematic solvers at finding models of satisfiable formulae for random k-SAT cases, despite the fact that they cannot guarantee either finding the answers or proving a particular Boolean formula unsatisfiable (Hoos and Stutzle as cited in Fu et al, 2021). According Mengshoel, et al (2020) SLS algorithms are greedy optimizers that avoid becoming stuck in local but non-global optima by making random moves. Since there are many complicated combinatorial optimization and decision issues in various scientific and industrial domains where computer methods have been comprehensively used, the research community and industry are very interested in finding solutions. SLS is a crucial and effective approach for resolving challenging combinatorial optimization issues (Kastrati & Biba, 2021). We were inspired to examine the performance of three options of GSAT solver (Greedy, Hill Climb, and Simulated Annealing) on various SAT instances, by the success that GSAT demonstrated in solving difficult combinatorial problems more quickly than systematic solvers based on DPLL (Muhammad and Stuckey, 2006). This paper is an extension of our earlier work by Kawu et al (2024) where we compared the performance of the Minisat solver and the Simulated Annealing option of the GSAT solver on the SATLIB benchmark. Our result indicated that Minisat outperformed SA in most instances. This result contradicts the earlier published result (Selman & Kautz, 1993; Hoos & Stützle, 2000). Motivated by this observation, we propose to compare three options of the GSAT solver version 35 on DIMACS (a subset of SATLIB) and Aloul Benchmarks.

Our work differs from algorithm configuration, where some parameters are chosen at run time to improve performance. All configuration are set at the onset of the program execution. The main objective of this paper is to test the performance of the three algorithms on different Benchmarks to gain and insight of their strengths and make recommendation for their use either in a hybrid SAT solver or variable activity initialization or initial weights. This work is to give an insight into the performance of each algorithm on different problem instances to serve as a guide for researchers who may like to build a hybrid solver using two stochastic local search algorithms or a local search and a systematic search algorithm. Our work will also serve as a guide for detecting stagnation in a local search. Stagnation occurs when a search algorithm is trapped in a local optima. This may be detected by high sideway moves.

GSAT

GSAT and WalkSAT are SLS solvers in which the variable to flip is chosen more carefully. For these types of solvers, the algorithm has two types of moves to choose from randomly. A “steepest descent” move where it flips the variable which minimizes the objective function. This step is not always possible because the algorithm can sometimes be at a local minimum (Reifenstein et al, 2023). GSAT is a greedy algorithm, that tries to flip variables so that as many clauses as possible are satisfied. If the chosen variable v is such that DIFF[v] > 0, the total number of unsatisfied clauses decreases. This is call this a "downward" move. If DIFF[v] = 0, then the total number of satisfied clauses remains constant; this is call a "sideways" move. Finally, if the flipped variable has DIFF[v] < 0, then an "upwards" move is performed, in which the number of satisfied clauses decreases. Each iteration of the inner loop is referred to as a "flip", and each iteration of the outer loop as a "try".

Hiil-Climbing

Hill climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by

DOI: 10.56892/bima.v8i4B.1166

making an incremental change to the solution. If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found. The solution obtained by this method may not be the global optimum; it will only be a local optimum. In hill-climbing search, we select any local change that improves the current value of the objective function. Greedy local search is a form of hill-climbing search where we select the local move that leads to the largest improvement of the objective function. Traditionally, one would end hill-climbing and greedy search procedures when no local move could further improve the objective function. Upon termination, the search would have reached a local, but not necessarily global, optimum of the objective function. (Selman & Gomes, 2006).

Simulated Annealing

According to Kirkpatrick et al (1983) Simulated annealing is a local search technique that comprises of sideway and downhill moves. In simulated annealing, downhill moves are accepted with a probability based on the size of the change in the objective function, with the `worst' moves becoming the least likely. The upward move is accepted based on the temperature level; at higher temperature the probability of accepting upward move is high while at lower temperature the algorithm return to the greedy approach. According to Selman and Gomes (2006), in simulated annealing, the temperature starts high and is slowly lowered during the search process. For more information on application of Simulated Annealing in SAT problems and how to improve its performance, see (Reifenstein et al, 2023).

Related Work

Stochastic Local Search algorithms have been a competitor for systematic search algorithms in solving satisfiability problems. Comparing the performance of different SAT solvers (systematic and incomplete) has received attention of a number of researchers. This is to discover their strengths and weaknesses and highlight areas that need improvement. Selman and Kautz(1993) compared GSAT with Simulated Annealing and showed that GSAT can be used as a efficient method for executing the low-temperature tail of annealing schedule. Mitchell et al. (1992), empirically found that the most difficult CNF expressions were created when clause to variable ratio = 4.30, and conjectured that the ratio approached 4.25 for large variables. Below 4.25, the expression has increasingly many solutions and is correspondingly easier to satisfy. Above 4.25, the expression is increasingly likely to have no solutions and a contradiction can be found more easily. At 4.25, about half the randomly generated CNF expressions are satistifiable, and few are sufficiently under or over-constrained to permit an easy answer. This property has been observed in a number of AI problems (Cheesman et al., 1991). Hao and Dorne (1996) compared the performance of their proposed heuristic local search template with Simulated Annealing. Their heuristics showed similar performance with SA on constraint satisfaction problems. Hoos and Stützle (2020) conducted a thorough comparative analysis of the performance of GSAT and WalkSAT across a range of benchmark instances, which included both randomized distributions and SAT-encoded problems from various domains. The results indicated that these algorithms are capable of effectively solving challenging SAT problems with a large number of variables, often surpassing the performance of conventional systematic SAT algorithms. Dutertre (2020) evaluated the performance of 16 state-of-the-

DOI: 10.56892/bima.v8i4B.1166

art SAT solvers on SMT_LIB repository Benchmark. Their result showed that CaDical outperformed the other solvers. Chu et al (2023) proposed a new clause weighted scheme for SLS solvers to remedy the bad effect initial soft clause weight has on the performance of SLS in solving weighted partial MaxSat problems. They compared their new solver (NuWLS) with four state-of-the-art solvers. Their results showed that their solver outperformed its competitors. They further investigated the complimentary of their solver improving hybrid solvers. Their result showed favorable improvement. FyalSAT, a Field Programmable Gate Array-based SLS solution, was proposed by Choi and Kim (2024). The authors compared their solution with others that were comparable and based on different SLS, such as WalkSAT. For benchmarks with a wide range of literals, their solver showed a high throughput. The new solver performed better than the existing ones. Li and Huang (2005) combined the deterministic variable selection heuristic of GSAT and the random heuristic of WalkSAT to remedy the weaknesses of the two heuristics. The hybrid heuristic showed improved performance in their new solver, G2 W SAT. Peng et al. (2020) compared their novel SLS heuristic for solving SAT instances (AspiSAT) with other SLS heuristics (Sparrow and Swqcc refer to the paper for details) on random 3-SAT instances. The AspiSAT heuristic outperformed the two heuristics.

MATERIALS AND METHOD

The three GSAT options tested in this work are Greedy, Hill-climb and Annealing. The parameters set for all the options are: maximum flip and maximum try. Annealing option has in addition the two parameters about the temperature schedule. We run several experiments with different settings. The first experiment was run with a maximum flips of five times the number of variables (which is the default) and five iterations (here the default is one iteration) for all the three options. More experiments were run with maximum flips of ten times number of variables, fifteen times, twenty times and twenty five times. For annealing option the default temperature schedule is 100 50 to 20 by .8 (see Algorithm 1 for detail explanation of the parameters). More experiments were run with different schedules, for example 200 100 to 20 by 0.8, 200 100 to 20 by 0.4.

After each experiment, the following information is recorded: number of variables, number of clauses, best flips, down ward moves, sideways moves, execution time, number of tries, and finally whether a model is found (SAT) or a failure is returned (UNSAT). The results obtained were presented in tabular form and graphs.

We compared the average execution time of the three options on different problem instances. We also compared their ability to solve problems.

GSAT algorithm as contained in the user's manual of the GSAT solver version 35. The algorithm is the same for the three options. The major differences are highlighted below the algorithm.

Algorithm1: The GSAT procedure

Procedure GSAT

Input: a set of clauses C, and integers MAX-FLIPS, and MAX-TRIES.

Output: a satisfying truth assignment of C, if any is found.

```
begin
  for i := 1 to MAX-TRIES:
    T := a randomly generated truth assignment;
    For j := 1 to MAX-FLIPS:
      if p satisfies C then return p;
```

DOI: 10.56892/bima.v8i4B.1166

for each variable v:

let MAKE[p] = the number of clauses currently unsatisfied

by T that would become satisfied if the truth value of

p were reversed ("flipped").

let BREAK[v] = the number of clauses currently satisfied

by T that would become unsatisfied if the truth value of

p were flipped.

let DIFF[v] = MAKE[v] - BREAK[v];

end for

let MAX_DIFF_LIST = list of variables with the greatest DIFF;

v := a random member of MAX_DIFF_LIST;

p := p with the truth assignment of v flipped;

end for

end for

return "no satisfying assignment found";

end.

The option "hillclimb" is available in GSAT version 26 (and later), which modifies the above algorithm by considering all variables with DIFF >=1 to be equally good, and all with DIFF <= -1 to be equally bad. It is not quite as greedy. Because of this change the implementation is able to store all the variables in three buckets (up, down, and sideways), and quickly shuffle variables between the buckets after each flip. This leads to about a 20 fold speedup for well form formulas with very large numbers of variables (10,000 or up).

GSAT also includes a simple simulated annealing option that may be run instead of or in conjunction with greedy local search. The syntax for supplying the annealing parameters in the interactive prompt of the GSAT solver is given as:

STEPS START_TEMP to END_TEMP by FACTOR

For example,

100 50 to 20 by .8

this mean anneal at temperature 50 for 100 steps, then at 50*0.8 for 100 steps, then 50*0.8*0.8 steps, etc, stopping AFTER a run in which the temperature is 20 or less. (For example, the last 100 steps may be at19.89.).

The experiments in the study were conducted on a Dell Latitude E7470 laptop equipped with a 2.40GHz Core i5 CPU and 16GB of RAM. The operating system used for the experiments was Ubuntu 22.04.3 LTS.

RESULTS AND DISCUSSION

The results obtained from different runs of the 3 options with different parameter settings are presented in the tables below:

Table 1: Comparison of the three algorithms on fpga problem instance

Algorithm	Average Down move	Average Sideways	Best flips
Greedy	9.8	1057.75	96.91
Hill Cimbing	11.42	799.58	182.33
Simulated Annealing	63.75	1086.58	709

The Greedy algorithm performed less down ward moves with higher sideway moves as shown in Table 1. It is followed by the Hill-Climbing algorithm where the average sideway moves reduces compared to the greedy option while both downward and best

DOI: 10.56892/bima.v8i4B.1166

flips increased. The Simulated Annealing option performed much downward and sideway moves. These results showed the greedy nature of the Greedy and Hill Climb option as compared to the exploratory nature of SA option.

Table 2: Comparison of the performance of the three algorithms on ii problem instance

Algorithm	Average clause/variable ratio	Percentage of instance solved
Greedy	11.30	28.57
Hill Cimbing	11.30	53.66
Simulated Annealing	11.30	4.88

Table 2 depicts that Hill Climbing algorithm outperformed the other two options on the ii problem instances with an average clause to variable ratio of 11.30. Simulated annealing solved the least number of problem instances.

Table 3: Execution time of the Greedy algorithm on aim problem instance based on number of variables.

Number of variables	Execution time in seconds	Number of Instances	Solved instances
50	0.016004	14	6
100	0.037406	13	2
200	0.04372	12	0

as shown in table 3, the execution time of the greedy algorithm increases as the number of variables in the proble instance increase. However the number of solved instances decreases as the number of variables increases.

Table 4: Execution time of the Hill Climbing algorithm on aim problem instance based on number of variables.

Number of variables	Execution time in seconds	Number of Instances	Solved instances
50	0.016481	14	2
100	0.043216	13	1
200	0.038865	12	1

Table 4 shows that Hill Climbing algorithm showed better performance in terms of execution time and number of solved instances as the number of number of variables increases. However, unlike the Greedy algorithm, Hill Climbing doesn't show a consistent increase in time with problem size, suggesting that the algorithm sometimes reaches solutions (or local optima) quickly, depending on the landscape of the problem instance. The Hill Climbing algorithm is similar to the Greedy algorithm in terms of execution time for 50 and 100 variables, but slightly slower for 200 variables. Both algorithms are efficient in terms of time.

DOI: 10.56892/bima.v8i4B.1166

Table 5: Execution time of the simulated Annealing algorithm on aim problem instance based on number of variables.

Number of variables	Execution time in seconds	Number of Instances	Solved instances
50	0.04899	14	6
100	0.057679	13	2
200	0.059043	12	0

Table 5 depicts the relative increase of the execution time of the Simulated Annealing algorithm as the number of variables increases. The algorithm could not solve any instance in 200 variables category of the problem instance.

Table 6: comparison of the performance of Greedy algorithm on different problem instances (x5).

Problem Instance	No. of Instances	Solved	Unsolved	Total Time	Max Try	Modal Try
Fpga	11	11	0	0.003536	1	1
Aim	39	7	32	0.004157	3	1
II	41	12	29	0.062851	3	1
Par	30	0	30	-	-	-

As shown in table 6, the Greedy algorithm solved all the instances in the fpga category. The SA algorithm solved only 18% and 29% from the aim and ii categories respectively. The algorithm could not solve a single instance from the par category in the specified number of tries and flips.

Table 7: comparison of the performance of Hill Climbing algorithm on different problem instances(x5).

Problem Instance	No. of Instances	Solved	Unsolved	Total Time	Max Try	Modal Try
Fpga	11	11	0	0.002986	1	1
Aim	39	4	35	0.004471	3	1
II	41	23	18	0.151586	5	1
Par	30	0	30	-	-	-

Table 7 shows that Hill Climbing algorithm performed similar but faster than the Greedy algorithm on the fpga problem instances. However, it solved les number of instances in the aim category and solved more than 50%of the instances in the ii category with a higher maximum try of 5 and higher execution time. The algorithm performed similar to the Greedy algorithm.

Table 8: comparison of the performance of Simulated Annealing algorithm on different problem instances (x5 flips and default temperature schedule).

Problem Instance	No. of Instances	Solved	Unsolved	Total Time	Max Try	Modal Try
Fpga	11	9	2	0.025627	5	1
Aim	39	8	31	0.0051054	4	1
II	41	2	39	0.002957	2	2
Par	30	0	30	-	-	-

Bima Journal of Science and Technology, Vol. 8(4B) Jan, 2025 ISSN: 2536-6041

DOI: 10.56892/bima.v8i4B.1166

Table 8 shows that Simulated Annealing algorihm performed less that both the greedy and Hill Climbing algorithms in terms of number of solved instances and maximum number of try. However, the algorithm showed better execution time on the ii category.

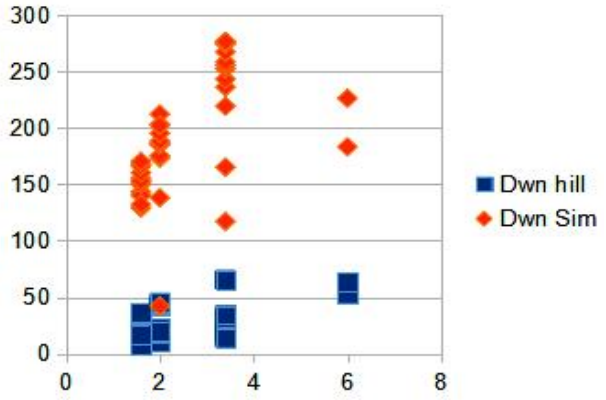

Figure 1: Comparison of the Average Downward moves of the Hill Climbing and Simulated Annealing algorithms on fpga problem instances.

Fig 1 shows that the Hill Climbing algorithm performed higher downward moves compared to the Simulated Annealing across all ratios.

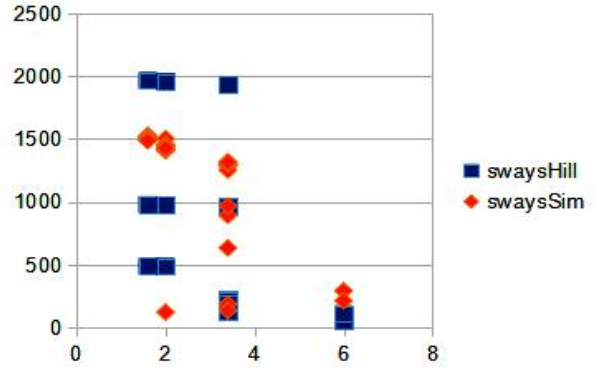

Figure 2: Comparison of the Average Sideways moves of the Hill Climbing and Simulated Annealing algorithms on fpga problem instances.

As shown in fig 2 the Hill Climbing algorithm performed higher sideways moves for a clause to variable ratio below 4 while the Simulated Annealing algorithm performed higher sideways move as the ratio increases above 4.

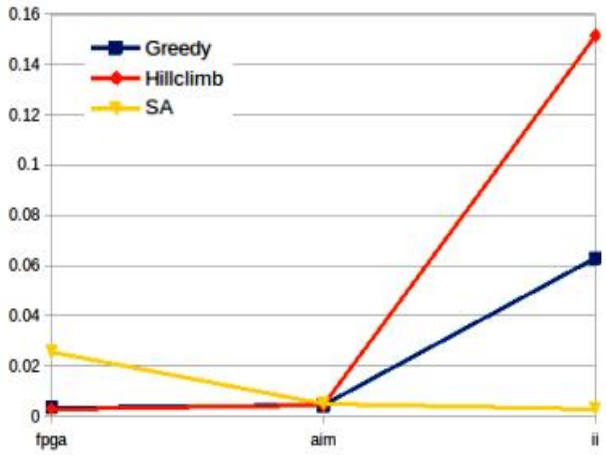

Figure 3: Comparison of the Average execution time of the Greedy, Hill Climbing and Simulated Annealing algorithms on fpga, aim and ii problem instances.

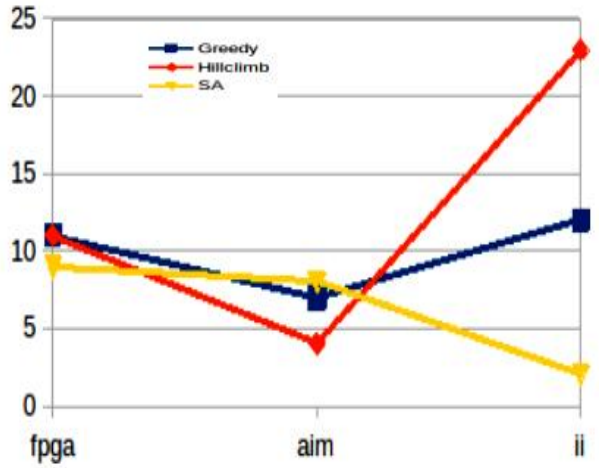

Figure 4: Comparison of the number of instances solved by the Greedy, Hill Climbing and Simulated Annealing algorithms on fpga, aim and ii problem instances.

DOI: 10.56892/bima.v8i4B.1166

Figures 3 and 4 compares the average execution time and number of instances solved respectively by the three algorithms on fpga , aim and ii problem instances . Greedy and Hill Climb were more efficient on both fpga and aim instances. However they spent longer time in ii category. This may be due to the higher number of instances they solved. The Greedy and Hill Climb algorithms have been able to solve all the instances in the fpga category while the SA algorithm solved only 82%. Despite the low performance of SA, it has however make higher number of downward moves and fewer sideway moves. This shows it suitability for use in variable activity initialization. This result is extracted from the result presented in table x and y to emphasis the execution time variation among the three algorithms and their ability finding a model.

We recommend using Simulated Annealing algorithm for initialization of variable weight of conflict-driven algorithms such as VSIDS heuristics this is due to the high average downward and sideways move of the algorithm on different sat problem instances more especially the fpga category of the Aloul Benchmark.

The results presented provide a number of insights. Table 1 and Fig 1 and 2 showed that Simulated annealing perform many downwards and sideways moves on industrial problem instances. The algorithm performed a number of exploration and exploitation of the search space. This algorithm can be used in a hybrid approach to assign weights to variables during the initial running phase of a solver for use by another algorithm or heuristic either systematic or stochastic. Table 6 to 8 showed that Hill Climb algorithm solved the highest number of problem instances followed by greedy approach. This shows that Hill Climb can be used in a Hybrid solver where two or more algorithms are use in a portfolio. Here the Hill Climbing algorithm will be selected to solved problem instances from random problems category.

CONCLUSION

in this work we compared the performance of three options of the GSAT solver on DIMACS and Aloul Benchmarks to provide an insight on their strengths on different problem categories (industrial and randomly generated problem instances) to serve as a guide for building a hybrid decision heuristic or SAT solver from a combination of Stochastic local search algorithms or a stochastic and systematic (DPLL or CDCL) algorithms. Our result showed that Hill Climbing is the best choice for Hybrid solver followed by the Greedy algorithm. However, Simulated Annealing may be preferable in initialization of variable weights or variable assignment (see Berend & Twitto, 2020 for initial value assignment for SLS). SA can also be applied in a hybrid decision heuristic of a systematic algorithm. We recommend using SA to guide Conflict-Driven Clause Learning solvers (refer to Cai et al., 2022). All clauses that the SA has satisfied are deleted from the clause database during the preprocessing phase. It can also be applied to incremental SAT solver like Minisat, in which the CDCL solver adopts the clauses that the SA has already satisfied. SA can be used to assign scores to variables with higher make value because of its exploratory nature at the start of the search process. The variable activity of the CDCL decision heuristic, such as VSIDS (Variable State Independent Decaying Sum), can be initialized using these scores.

REFERENCES

Berend, D., & Twitto, Y. (2020). Effect of initial assignment on local search performance for Max Sat. In 18th *International Symposium on Experimental Algorithms (SEA 2020)*. Schloss-Dagstuhl- Leibniz Zentrum für Informatik

DOI: 10.56892/bima.v8i4B.1166

Cai, S., Zhang, X., Fleury, M., & Biere, A. (2022). Better decision heuristics in CDCL through local search and target phases. *Journal of Artificial Intelligence Research*, 74, 1515-1563.

Choi, Y. K., & Kim, C. (2024). FYalSAT: High-Throughput Stochastic Local Search K-SAT Solver on FPGA. *IEEE* Access.

Cohen, A., Nadel, A., & Ryvchin, V. (2021). Local search with a SAT oracle for combinatorial optimization. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 87-104). Cham: Springer International Publishing.

Cook, S. A. (2023). The complexity of theorem-proving procedures. In Logic, automata, and computational complexity: The works of Stephen A. Cook (pp. 143-152).

Chu, Y., Luo, C., Hoos, H. H., & You, H. (2023). Improving the performance of stochastic local search for maximum vertex weight clique problem using programming by optimization. Expert Systems with Applications, 213, 118913.

Dutertre, B. (2020). An Empirical Evaluation of SAT Solvers on Bit-vector Problems. In *SMT* (pp. 15-25).

Fu, H., Wu, G., Liu, J., & Xu, Y. (2021). More efficient stochastic local search for satisfiability. Applied Intelligence, 51, 3996-4015.

Ganesh, V., & Vardi, M. Y. (2020). On the Unreasonable Effectiveness of SAT Solvers.

Ghanem, M., & Siniora, D. (2021). On Theoretical Complexity and Boolean Satisfiability. *arXiv preprint arXiv:2112.11769*

Hao, J. K., & Dorne, R. (1996). Empirical studies of heuristic local search for constraint solving. In Principles and Practice of Constraint Programming—CP96: Second International Conference, CP96 Cambridge, MA, USA, August 19–22, 1996 Proceedings 2 (pp. 194-208). Springer Berlin Heidelberg.

Hoos, H. H., & Stützle, T. (2000). Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, *24*(4), 421-481.

Kastrati, M., & Biba, M. (2021). Stochastic local search: a state-of-the-art review. International Journal of Electrical and Computer Engineering, 11(1), 716.

Kawu, A. J., Wajiga, G. M., & Malgwi, Y. M. (2024). Comparison of the Empirical Performance of Minisat and GSAT Solvers on SATLIB Benchmark. *BIMA JOURNAL OF SCIENCE AND TECHNOLOGY (2536-6041)*, *8*(1B), 197-205.

Li, C. M., & Huang, W. Q. (2005). Diversification and determinism in local search for satisfiability. In Theory and Applications of Satisfiability Testing: 8th International Conference, SAT 2005, St Andrews, UK, June 19-23, 2005. Proceedings 8 (pp. 158-172). Springer Berlin Heidelberg.

Mengshoel, O. J., Yu, T., & Zeng, M. (2020). Stochastic local search and machine learning: From theory to applications and vice versa. In ECAI 2020 (pp. 2919-2920). IOS Press.

Muhammad, R., & Stuckey, P. J. (2006). A stochastic non-CNF SAT solver. In PRICAI 2006: Trends in Artificial Intelligence: 9th Pacific Rim International Conference on Artificial Intelligence Guilin, China, August 7-11, 2006 Proceedings 9 (pp. 120-129). Springer Berlin Heidelberg.

Peng, C., Xu, Z., & Mei, M. (2020). Applying aspiration in local search for satisfiability. PloS one, 15(4),

DOI: 10.56892/bima.v8i4B.1166

e0231702.*https://doi.org/10.1371/journal.pone.0231702*

Reifenstein, S., Leleu, T., McKenna, T., Jankowski, M., Suh, M. G., Ng, E., ... & Yamamoto, Y. (2023). Coherent SAT solvers: a tutorial. *Advances in Optics and Photonics*, *15*(2), 385-441.

Selman, B., & Gomes, C. P. (2006). Hill-climbing search. *Encyclopedia of cognitive science*, *81*(333- 335), 10.

Selman, B., & Kautz, H. A. (1993). An empirical study of greedy local search for satisfiability testing. In *AAAI* (Vol. 93, pp.46-51).