Implementing the DPLL Algorithm and Evaluating Advanced Heuristics for Boolean Satisfiability

Gauthami Arun and Yatharth Agarwal

Source Code

***Abstract*—Electronic Design Automation (EDA) is pivotal in modern digital circuit design, ensuring efficiency, reliability, and scalability. Among its core components, the Boolean Satisfiability (SAT) problem is a fundamental challenge, with implications spanning from verification to synthesis. In this paper, we delve into the significance of SAT solvers within the realm of EDA, particularly emphasizing the implementation of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. We integrate two advanced heuristics, Conflict-Driven Clause Learning (CDCL) and Variable State Independent Decaying Sum (VSIDS), into the DPLL framework to enhance solver efficiency and efficacy. Through a comprehensive evaluation across diverse benchmarks, we scrutinize the performance of these heuristics, shedding light on their respective strengths and limitations. Our study underscores the critical role of SAT solvers in advancing EDA methodologies and provides valuable insights for further refinement and optimization in digital circuit design automation.**

***Index Terms*—Satisfiability Solver, DPLL Algorithm, Conflict Driven Clause Learning, Variable State Independent**

I. Introduction

In the realm of Very Large Scale Integration (VLSI) Electronic Design Automation (EDA), the complexity of designing intricate digital circuits continues to escalate with the growing demand for faster, smaller, and more power-efficient devices. As the intricacy of these designs increases, so does the necessity for efficient tools and methodologies to ensure correctness and optimize performance within tight timeframes. Satisfiability (SAT) solvers emerge as indispensable instruments in this landscape, playing a pivotal role in verifying the correctness of digital designs and optimizing various stages of the design flow.

SAT solvers are algorithms employed to determine the satisfiability of a propositional logic formula, i.e., whether an assignment of truth values exists to the variables that satisfy the formula. In the VLSI EDA industry, SAT solvers find applications in diverse areas such as equivalence checking, formal verification, synthesis, and placement and routing. These tools enable designers to address complex design challenges by automating the process of verifying logical properties, identifying design errors, and optimizing design parameters.

One of the fundamental algorithms underlying modern SAT solvers is the DPLL algorithm. Introduced in the 1960s, the DPLL algorithm laid the groundwork for subsequent advancements in SAT-solving techniques. Its recursive, backtracking-based approach and unit propagation form the basis of many contemporary SAT solvers. Despite its effectiveness, the performance of the basic DPLL algorithm can be limited when dealing with large-scale, real-world instances.

To address the scalability challenges inherent in SAT solving, researchers have developed many heuristic techniques to enhance the efficiency and effectiveness of the DPLL algorithm. Heuristics such as conflict-driven clause learning (CDCL), variable and clause elimination strategies, activity-based variable selection, and restart policies have significantly augmented the performance of SAT solvers, enabling them to tackle instances of unprecedented complexity within reasonable time and resource constraints.

This project report aims to delve into the intricacies of SAT solvers, focusing on the DPLL algorithm and the role of heuristic enhancements in improving its performance. Through a comprehensive examination of underlying principles, algorithmic intricacies, and empirical evaluations, this report sheds light on the significance of SAT solvers in the VLSI EDA industry and the ongoing efforts to push the boundaries of efficiency and scalability through heuristic innovation.

We demonstrate the following in our project

1) We have successfully implemented the DPLL algorithm as part of our project.
2) We have successfully confirmed its functionality by incorporating two heuristic methods, CDCL and VSIDS.
3) We have comprehensively analyzed the performance across various benchmarks. This includes evaluating the impact of altering the decay rate of VSIDS and the effects of enabling restarts.

II. Background and Motivation

Satisfiability (SAT) is a fundamental problem in computer science, with wide-ranging applications in artificial intelligence, formal verification, and combinatorial optimization. At its core, the SAT problem involves determining whether a given Boolean formula can be satisfied by assigning truth values to its variables. Despite its seemingly simple formulation, SAT is notorious for its computational complexity, among the first problems proven to be NP-complete by Cook and Levin in 1971.

The NP-hardness of SAT means that no known algorithm can solve all instances of the problem efficiently in polynomial time. However, researchers have developed increasingly sophisticated algorithms and techniques to tackle SAT instances effec-

This work was conducted as part of the ECE 51216 (Digital Systems Design Automation) course project.

Gauthami Arun (e-mail: arung@purdue.edu) and Yatharth Agarwal (e-mail: agarw414@purdue.edu) are with the Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA.

tively. One notable milestone in this journey is the development of SAT solvers.

A. Evolution of SAT Solvers:

SAT solvers have undergone significant evolution since their inception. Early approaches focused on exhaustive search methods, quickly becoming impractical for larger problem instances due to their exponential time complexity. However, advancements in algorithmic design and optimization strategies paved the way for more efficient solvers.

A pivotal development in the evolution of SAT solvers came with the introduction of the DPLL algorithm. Proposed in the 1960s and refined over subsequent decades, DPLL provided a systematic backtracking search framework for solving SAT instances. DPLL significantly improved SAT-solving algorithms' efficiency by leveraging a combination of unit propagation and CDCL.

B. DPLL Algorithm:

The DPLL algorithm operates by recursively selecting variables, assigning truth values to them, and simplifying the problem until a satisfying assignment is found or a contradiction is detected. Key to its effective quickly becoming unit propagation, where unit clauses are identified and propagated to derive additional variable assignments. Moreover, the CDCL technique enhances DPLL by dynamically learning conflict clauses during the search process, enabling more informed decisions and efficient backtracking.

C. Heuristic Innovations:

In recent years, further enhancements to SAT solvers have been driven by heuristic strategies to guide the search process more intelligently. CDCL and VSIDS are notable heuristics.

CDCL prioritizes exploring conflicting paths by learning from encountered conflicts and leveraging learned clauses to guide future search decisions. This heuristic effectively prunes the search space and directs the solver towards promising solution regions.

VSIDS, on the other hand, assigns scores to variables based on their involvement in conflicts and dynamically adjusts these scores over time. By focusing the search on variables with high conflict activity, VSIDS directs the solver toward potential solutions more efficiently.

III. Design Methodology

A. DPLL Algorithm

The DPLL algorithm is a fundamental approach to solving the Boolean Satisfiability Problem (SAT), a classic problem in computer science and mathematical logic. SAT involves determining whether there exists an assignment of truth values (true or false) to variables in a propositional formula such that the formula evaluates to true.

The algorithm starts by setting up its initial state, which involves establishing data structures to describe the input CNF formula. This process entails creating and organizing lists that store information about literals, including their frequencies and polarities within the formula.

Unit propagation is a critical component of the DPLL algorithm. It involves iteratively identifying unit clauses, which are clauses containing only one literal. If such a clause is found, the truth value of the literal is determined and propagated through the formula. This process continues until no more unit clauses can be identified.

After unit propagation, the algorithm applies transformations to simplify the formula based on the assigned truth values. This involves removing clauses satisfied by the assigned literals and simplifying clauses that contain negated literals. These transformations reduce the size of the formula and facilitate further exploration.

If the formula is not fully satisfied after unit propagation and transformation, the algorithm selects an unassigned variable based on a heuristic. A common heuristic is to choose the variable with the highest frequency in the remaining formula. The algorithm then recursively explores two branches: one where the selected variable is assigned true and another where it is assigned false. This branching continues until a satisfying assignment is found or the search space is exhausted.

The recursion terminates when either a satisfying assignment is found, or it is determined that no such assignment exists. If a satisfying assignment is found, the algorithm reports "SAT" along with the assignment. If no satisfying assignment is found, the algorithm reports "UNSAT" to indicate that the formula is unsatisfiable.

The DPLL algorithm combines efficient data structures, heuristic search, and backtracking to explore the solution space of SAT instances systematically. Despite being a basic algorithm, DPLL forms the basis for more sophisticated SAT solvers used in various applications, including formal verification, planning, and scheduling.

B. Relevant Data structures

Our code employs a variety of data structures to streamline SAT-solving algorithms:

- A dictionary is utilized for recording the occurrences of each literal within the CNF clauses.
- Several lists are employed for different tasks:
  - To store the CNF clauses extracted from the file.
  - To keep track of modified clause sets during Boolean Constant Propagation (BCP).
  - A separate list manages assigned literals during unit propagation.
  - Propagation of literals during unit propagation.
  - Storage of learned clauses during conflict analysis.
  - Propagation of literals during two-literal watch propagation.
  - A list to hold assignments in the SAT solution.
- A tuple is used to encapsulate the result and statistics of the SAT solver.
- Multi-dimensional lists are integral for various functions:
  - A list of lists, retaining two watched literals per clause for 2-literal watch propagation.
  - A list of literals representing a new clause during clause creation.

These data structures play integral roles across the CDCL algorithm, facilitating clause manipulation, conflict resolution, decision-making, and backtracking tasks. They are instrumental in enabling the algorithm to efficiently search for satisfying assignments or identify unsatisfiability through effective learning mechanisms.

C. Conflict Driven Clause Learning

CDCL is a highly effective algorithm for solving the Boolean Satisfiability Problem (SAT), a fundamental problem in computer science and mathematics. CDCL builds upon the DPLL algorithm and incorporates several enhancements for efficiency.

The CDCL algorithm begins by initially leaving all variables unassigned. It then performs unit propagation, which assigns truth values to variables based on unit clauses. If a variable gets a truth value assigned during unit propagation, the implications of that assignment are propagated to other variables in the formula.

During propagation, if a conflict arises where a clause becomes unsatisfiable given the current variable assignments, CDCL detects this conflict. A conflict signals that the current set of assignments is invalid and needs to be revised. Upon detecting a conflict, CDCL analyzes the sequence of decisions and assignments that led to the conflict. This analysis identifies a set of clauses, called the "conflict clause," that were responsible for causing the conflict.

CDCL learns from these conflicts by adding the conflict clause to the formula as an additional constraint. This learned clause prevents the solver from revisiting the same assignment path that previously caused the conflict. After learning the conflict clause, CDCL backtracks to a previous decision level.

Backtracking involves undoing some of the most recent variable assignments and decisions and returning to a prior state where an alternative decision can be explored.

Periodically, CDCL may restart the entire search process from scratch to explore completely different branches of the search space. Restarting helps escape local minima and can improve the chances of finding global solutions.

CDCL repeatedly executes this main loop of making decisions on variables, propagating the implications, detecting conflicts, analyzing conflicts to learn new clauses, backtracking, and potentially restarting the search. This loop continues until either a satisfying assignment is found (SAT case) or the problem is proven unsatisfiable (UNSAT case). After termination, CDCL verifies that any claimed satisfying assignment satisfies all the original clauses.

CDCL's effectiveness lies in its ability to efficiently navigate the vast solution space of SAT instances by learning from conflicts and adjusting its search strategy dynamically. As a result, CDCL is widely used in various applications requiring Boolean satisfiability solving, such as hardware and software verification, planning, scheduling, and optimization.

D. Variable State Independent Decay Sum - VSIDS

VSIDS is a heuristic used in CDCL solvers, often employed in SAT solvers. The VSIDS heuristic is designed to guide the solver in selecting which variable to assign a value to during the search process.

In CDCL solvers, each variable is assigned a score representing its relevance or importance in searching for a satisfying assignment. Initially, all variable scores are set to the same value. Whenever a variable appears in a clause that causes a conflict, the score of that variable is increased. This reflects that variables involved in recent conflicts are potentially more crucial for finding a solution.

Periodically, the solver applies a decay factor to all variable scores, reducing their values. This decay mechanism prevents variables in very old conflicts from indefinitely dominating the decision-making process. It allows variables that have not been involved in conflicts recently to gain more prominence over time.

When the solver needs to decide which variable to branch on (assign a truth value to), it typically selects the variable with the highest score. This heuristic prioritizes variables involved in recent conflicts, as they are considered more likely to contribute to finding a satisfying assignment.

If the solver encounters a conflict during the search process, it analyzes it to learn from it. This conflict analysis identifies the variables responsible for the conflict and updates their scores accordingly, increasing the scores of the relevant variables.

In CDCL solvers, the VSIDS heuristic is implemented efficiently using data structures like priority queues or heaps to maintain the variable scores. These data structures allow fast retrieval of the variable with the highest score when selecting the following variable for branching.

Overall, VSIDS is a powerful heuristic that guides CDCL solvers in exploring the search space efficiently by prioritizing variables likely to contribute to finding a satisfying assignment based on their recent involvement in conflicts during the solving process.

IV. Experimental methodology

We assessed the performance of the satisfiability solver by benchmarking various CNF formulae and tracking metrics such as the number of restarts, solution time, learned clauses, decisions, and implications. This evaluation aimed to understand the effects of implementing different heuristics and how the solver's performance varies with the number of literals.

A. Scripts and test setup

We created two scripts to simplify the evaluation process. The first script, $run.sh$, ran multiple benchmarks, and stored their results in text files. It had the provision to terminate computations if the execution time exceeded 2 hours. Second, we implemented $collect\_results.py$, which stores the relevant details from the above-generated text files in a $.csv$ file, making evaluations of results effortless.

B. Input CNF Formulas

We employ two sets of benchmarks in CNF format to assess the satisfiability solver. Firstly, we utilize instances from the DIMACS Benchmark set with 50, 100, and 200 variables. Secondly, we select benchmarks with varying numbers of variables (50, 75, 100, 125, 150, 175) from Uniform Random-3-SAT. For each dataset, we choose three instances of both SAT and UNSAT, providing coverage across a total of 52 instances.

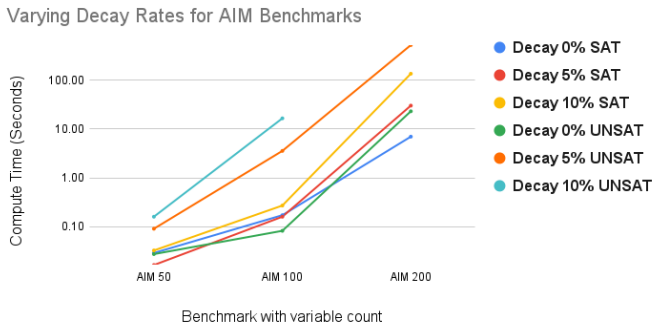

Fig. 1. Performance analysis with and without restarts on AIM benchmarks (50,100,200 variables).Computation time on Y-Axis in log scale and benchmarks on X-Axis

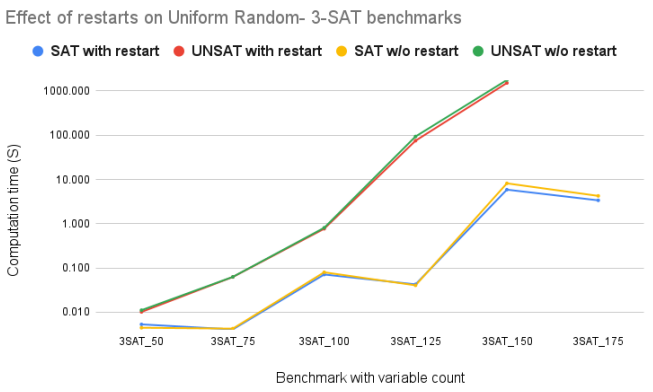

Fig. 2. Performance analysis with and without Restarts on Uniform Random-3-SAT benchmarks (50,75,100,125,150,175 variables).Computation time on Y-Axis in log scale and benchmarks on X-Axis

C. Evaluation of Heuristics

To outline our experimental approach to evaluate heuristics in the DPLL SAT solver with CDCL and VSIDS, we first looked at how restarting the search process affected solver efficiency. We assessed the above benchmarks with and without restarts, keeping the VSIDS decay rate constant at 5%. This allowed us to determine the impact of restarting the search process on the solver's effectiveness in locating solutions methodically.

Next, we investigated how altering the VSIDS decay rate alters solver performance. To do this, the benchmarks were evaluated at varied decay rates of 0%, 5%, and 10%. By applying this experimental design, we want to understand how the decay rate impacts the performance of the satisfiability solver.

V. Results and Discussion

In our project, we implemented the DPLL algorithm augmented with the Variable VSIDS heuristic and CDCL mechanism. Through rigorous evaluation of CNF formulas, we observed notable trends. The results for all benchmarks are listed in Fig. 5. The average computation time across three instances was averaged and is plotted in Fig. 1 for the DIMAC benchmarks and Fig. 2 for Uniform Random-3-SAT.

Firstly, we noted an exponential increase in computation time with an increase in the number of literals within the formulas(Fig. 1 & 2). This phenomenon suggests that the algorithm's complexity multiplies with the input size, likely due to the combinatorial explosion of possible assignments to be explored.

Moreover, our evaluation revealed that the time required for processing unsatisfiable (UNSAT) instances exceeds that of satisfiable (SAT) instances, which can be attributed to the additional effort in conflict analysis and resolution inherent in CDCL-based solvers when encountering unsatisfiability.

Additionally, we looked into how restart tactics affected the solver's overall performance. We also investigated how the VSIDS decay rate affected the solver's behavior.

A. Improvement due to restarts

Fig. 1 and Fig. 2 illustrate that restarts have a limited impact on performance for instances with up to 200 variables in our implementation. This outcome can be attributed to several factors. Primarily, the modest problem size might not necessitate extensive backtracking, thus diminishing the potential benefits of restarts. It is observed that the gap between computation times starts to diverge slowly for instances having greater than 175 variables. Additionally, the nature of the problem instances may lack intricate search landscapes or deep local optima, reducing the occasions where restarts could aid in navigating challenging regions of the solution space. Furthermore, the efficacy of restarts hinges on the chosen restart strategy and the algorithm's configuration, where sub-optimal choices or excessive overhead could undermine their effectiveness. By analyzing these facets within our experimental setup, we provide a nuanced understanding of why restarts do not significantly improve the performance of our DPLL algorithm implementation for smaller-scale instances.

B. Impact of VSIDS decay rate

The computation times for VSIDS decay rates of 0%, 5%, and 10% are presented in Table 6. Subsequently, the average times across three instances are plotted in Fig. 3 for the DIMACS AIM benchmarks and in Fig. 4 for Uniform Random 3-SAT benchmarks.

Surprisingly, the best performance was consistently achieved when the decay rate was set to 0%, with deteriorating performance observed as the decay rate increased to 5% and 10%. One potential explanation for this observation lies in the delicate balance between exploration and exploitation within the search space. When the decay rate is set to 0%, the VSIDS heuristic maintains high activity scores for variables over prolonged periods, allowing the solver to explore promising branches of the search tree thoroughly. However, as the decay rate increases to 5% and 10%, the influence of past decisions diminishes more rapidly, potentially leading to premature down-prioritization of critical variables and limiting the solver's ability to exploit valuable information gathered during the search process. This imbalance may result in the solver being less effective at efficiently navigating the search space and resolving conflicts, ultimately leading to the observed deterioration in performance. Further analysis and benchmarking are required to understand the underlying mechanisms driving this trend entirely and optimize the performance of the VSIDS heuristic

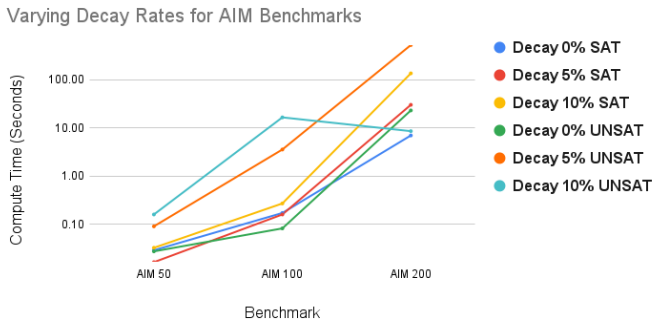

Fig. 3. Performance analysis for VSIDS decay rate at 0%, 5% and 10% on AIM benchmarks (50,100,200 variables). Computation time on Y-Axis in log scale and benchmarks on X-Axis

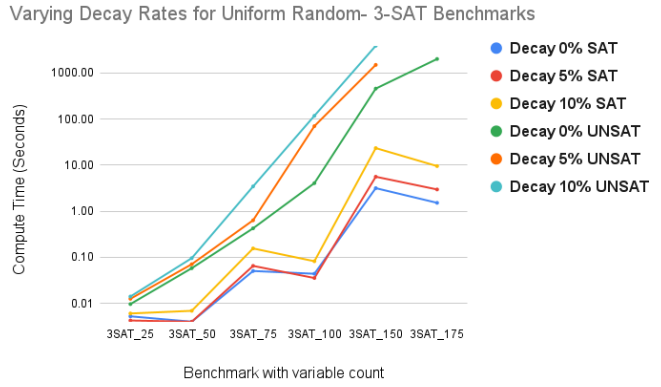

Fig. 4. Performance analysis for VSIDS decay rate at 0%, 5% and 10% on Uniform Random- 3-SAT benchmarks (50,75,100,125,150,175 variables).Computation time on Y-Axis in log scale and benchmarks on X-Axis

within our implementation.

VI. Conclusion

Our project successfully implemented and benchmarked an SAT solver leveraging the DPLL algorithm with CDCL and VSIDS heuristics. We observed limited performance improvement by enabling restarts and reduction in performance by increasing the VSIDS decay rate even by small percentages through extensive testing, particularly for instances with up to 200 variables. Further, scalability challenges emerged for larger instances, suggesting the need for optimization strategies. Our findings highlight the effectiveness of advanced techniques in SAT solving and underscore opportunities for further research to enhance scalability and efficiency. Possible improvements include implementing learned clause deletion, Unique Implication-based clause learning, and exploring hybrid approaches with machine learning. These enhancements promise to advance state-of-the-art SAT solving and address real-world scalability challenges.

VII. Acknowledgements

We want to express our sincere gratitude to Professor Anand Raghunathan for his invaluable guidance and insightful input throughout this project. His expertise and mentorship have been instrumental in shaping the direction of this work.

We extend our heartfelt thanks to our Teaching Assistant, Sujay Pandit, for his unwavering support and patience in clarifying doubts during various project stages. His dedication has contributed significantly to the successful completion of this project.

VIII. Contributions

Gauthami Arun and Yatharth Agarwal collaborated seamlessly. Gauthami took the lead in implementing CDCL and VSID heuristics, pivotal components enhancing algorithmic efficiency. At the same time, Yatharth developed the base DPLL algorithm and conducted comprehensive analyses across various benchmarks. Their joint efforts ensured the algorithm's robustness and scalability. Yatharth's analytical insights provided critical guidance for refining the algorithm's design, complementing Gauthami's meticulous implementation. Together, their dynamic teamwork propelled the project's success, advancing its objectives and the broader field of study.

Appendix

Variables	Clauses	File Name	Result	With Restarts					Without Restarts				
				Restarts	Solve time	Learned Clauses	Decisions	Implications	Restarts	Solve time	Learned Clauses	Decisions	Implications
DIMACS AIM Benchmarks													
50	100	aim-50-2_0-yes1-1	SAT	7	0.017	282	290	4769	0	0.018	282	290	4769
		aim-50-2_0-yes1-2	SAT	9	0.019	333	341	4943	0	0.019	333	341	4943
		aim-50-2_0-yes1-3	SAT	7	0.018	296	304	3919	0	0.018	296	304	3919
50	100	aim-50-2_0-no-1	UNSAT	9	0.037	455	454	5247	0	0.036	455	454	5247
		aim-50-2_0-no-2	UNSAT	11	0.108	894	893	12379	0	0.111	894	893	12379
		aim-50-2_0-no-3	UNSAT	11	0.143	1762	1761	24030	0	0.154	1762	1761	24030
100	200	aim-100-2_0-yes1-1	SAT	8	0.055	532	547	12742	0	0.048	532	547	12742
		aim-100-2_0-yes1-2	SAT	11	0.324	2311	2322	67311	0	0.226	2311	2322	67311
		aim-100-2_0-yes1-3	SAT	11	0.293	2676	2685	78063	0	0.262	2676	2685	78063
100	200	aim-100-2_0-no-1	UNSAT	11	0.098	1139	1138	15095	0	0.080	1139	1138	15095
		aim-100-2_0-no-2	UNSAT	11	13.809	8718	8717	147464	0	13.220	8718	8717	147464
		aim-100-2_0-no-3	UNSAT	9	0.515	1716	1715	19023	0	0.518	1716	1715	19023
200	400	aim-200-2_0-yes1-1	SAT	11	8.524	68729	68763	3158541	0	11.225	68729	68763	3158541
		aim-200-2_0-yes1-2	***			***					***		
		aim-200-2_0-yes1-3	SAT	11	57.525	16007	16022	583519	0	71.673	16007	16022	583519
200	400	aim-200-2_0-no-1	***			***					***		
		aim-200-2_0-no-2	UNSAT	11	1042.16	154456	154455	3255606	0	434.278	154456	154455	3255606
		aim-200-2_0-no-3	UNSAT	11	6.454	9488	9487	160983	0	8.587	9488	9487	160983
UNIFORM RANDOM 3SAT BENCHMARKS													
50	218	uf50-01	SAT	5	0.009	41	54	887	0	0.006	41	54	887
		uf50-02	SAT	3	0.002	12	22	345	0	0.003	12	22	345
		uf50-03	SAT	4	0.004	21	42	514	0	0.005	21	42	514
50	218	uuf50-01	UNSAT	6	0.012	100	99	2089	0	0.015	100	99	2089
		uuf50-02	UNSAT	5	0.010	62	61	1355	0	0.011	62	61	1355
		uuf50-03	UNSAT	5	0.008	50	49	1175	0	0.008	50	49	1175
75	325	uf75-01	SAT	4	0.004	13	37	513	0	0.004	13	37	513
		uf75-02	SAT	1	0.002	1	21	41	0	0.002	1	21	41
		uf75-03	SAT	4	0.006	24	36	744	0	0.007	24	36	744
75	325	uuf75-01	UNSAT	7	0.042	210	209	6310	0	0.046	210	209	6310
		uuf75-02	UNSAT	8	0.057	260	259	8120	0	0.060	260	259	8120
		uuf75-03	UNSAT	7	0.089	365	364	10795	0	0.083	365	364	10795
100	430	uf100-01	SAT	7	0.070	251	266	9919	0	0.079	251	266	9919
		uf100-02	SAT	9	0.127	498	511	20738	0	0.147	498	511	20738
		uf100-03	SAT	4	0.018	33	51	1266	0	0.014	33	51	1266
100	430	uuf100-01	UNSAT	9	0.375	1068	1067	41000	0	0.446	1068	1067	41000
		uuf100-02	UNSAT	11	1.188	2127	2126	85939	0	1.178	2127	2126	85939
		uuf100-03	UNSAT	11	0.746	1454	1453	58635	0	0.804	1454	1453	58635
125	538	uf125-01	SAT	6	0.022	42	70	1907	0	0.019	42	70	1907
		uf125-02	SAT	8	0.079	206	228	9486	0	0.079	206	228	9486
		uf125-03	SAT	6	0.027	54	78	2337	0	0.023	54	78	2337
125	538	uuf125-01	UNSAT	11	4.652	3367	3366	162400	0	5.128	3367	3366	162400
		uuf125-02	UNSAT	11	196.640	11847	11846	572008	0	246.086	11847	11846	572008
		uuf125-03	UNSAT	11	23.919	5586	5585	265514	0	29.220	5586	5585	265514
150	645	uf150-01	SAT	11	1.952	4978	5006	286218	0	2.364	4978	5006	286218
		uf150-02	SAT	7	0.165	359	393	21037	0	0.199	359	393	21037
		uf150-03	SAT	11	15.535	40206	40227	2415938	0	21.931	40206	40227	2415938
150	645	uuf150-01	UNSAT	11	1107.15	25256	25255	1433135	0	1396.00	25256	25255	1433135
		uuf150-02	UNSAT	11	2145.61	24994	24993	1421462	0	2445.52	24994	24993	1421462
		uuf150-03	UNSAT	11	1281.02	22941	22940	1307028	0	1453.26	22941	22940	1307028
175	753	uf175-01	SAT	11	9.201	20310	20344	1308386	0	11.724	20310	20344	1308386
		uf175-02	SAT	9	0.506	952	988	60499	0	0.575	952	988	60499
		uf175-03	SAT	11	0.421	791	826	52171	0	0.464	791	826	52171
175	753	uuf175-01	***			***					***		
		uuf175-02	***			***					***		
		uuf175-03	***			***					***		

*** Evaluation not completed as execution stoped at 2Hrs

Fig. 5. Results of all performed benchmarks comparing DPLL with and without restarts. The table includes the number of restarts, solve time, number of learned clauses, decisions, and implications. Benchmarks include three instances each of SAT and UNSAT for the DIMACS benchmark set with 50, 100, and 200 variables and Uniform Random-3-SAT benchmarks with 50, 75, 100, 125, 150, and 175 variables.

Variables	Clauses	File Name	Solve time (Decay 0%)	Solve time (Decay 5%)	Solve time (Decay 10%)
DIMACS AIM BENCHMARKS					
50	100	aim-50-2_0-yes1-1	0.02	0.02	0.03
		aim-50-2_0-yes1-2	0.06	0.02	0.03
		aim-50-2_0-yes1-3	0.01	0.02	0.03
		aim-50-2_0-no-1	0.02	0.03	0.06
		aim-50-2_0-no-2	0.02	0.10	0.20
		aim-50-2_0-no-3	0.04	0.14	0.23
100	200	aim-100-2_0-yes1-1	0.03	0.05	0.06
		aim-100-2_0-yes1-2	0.09	0.20	0.44
		aim-100-2_0-yes1-3	0.40	0.23	0.31
		aim-100-2_0-no-1	0.03	0.08	0.12
		aim-100-2_0-no-2	0.15	10.21	48.07
		aim-100-2_0-no-3	0.07	0.44	1.63
200	400	aim-200-2_0-yes1-1	9.06	8.51	67.80
		aim-200-2_0-yes1-2	10.39	***	***
		aim-200-2_0-yes1-3	1.55	51.95	203.97
		aim-200-2_0-no-1	62.92	***	***
		aim-200-2_0-no-2	6.10	1042.16	***
		aim-200-2_0-no-3	0.41	6.33	9.09
UNIFORM RANDOM 3SAT BENCHMARKS					
50	218	uf50-01	0.01	0.01	0.01
		uf50-02	0.00	0.00	0.00
		uf50-03	0.00	0.00	0.01
		uuf50-01	0.01	0.02	0.02
		uuf50-02	0.01	0.01	0.01
		uuf50-03	0.01	0.01	0.01
75	325	uf75-01	0.00	0.00	0.01
		uf75-02	0.00	0.00	0.00
		uf75-03	0.01	0.01	0.01
		uuf75-01	0.04	0.05	0.08
		uuf75-02	0.04	0.06	0.09
		uuf75-03	0.10	0.10	0.12
100	430	uf100-01	0.04	0.06	0.10
		uf100-02	0.09	0.12	0.31
		uf100-03	0.02	0.01	0.06
		uuf100-01	0.17	0.36	0.60
		uuf100-02	0.70	0.92	3.92
		uuf100-03	0.41	0.63	5.86
125	538	uf125-01	0.04	0.02	0.12
		uf125-02	0.07	0.07	0.10
		uf125-03	0.03	0.02	0.03
		uuf125-01	1.03	4.08	9.27
		uuf125-02	10.15	184.68	303.67
		uuf125-03	1.06	23.06	42.75
150	645	uf150-01	0.64	1.73	5.59
		uf150-02	0.13	0.15	0.54
		uf150-03	8.75	14.98	64.42
		uuf150-01	203.95	1107.15	3929.56
		uuf150-02	590.32	2145.61	***
		uuf150-03	588.95	1281.02	***
175	753	uf175-01	3.67	8.17	26.05
		uf175-02	0.74	0.41	2.01
		uf175-03	0.17	0.35	0.63
		uuf175-01	2029.08	***	***
		uuf175-02	***	***	***
		uuf175-03	***	***	***
*** Evaluation not completed as execution stoped at 2Hrs					

Fig. 6. Results of all performed benchmarks comparing solving time for VSIDS decay rate at 0%, 5% and 10%. Benchmarks include three instances each of SAT and UNSAT for the DIMACS benchmark set with 50, 100, and 200 variables and Uniform Random-3-SAT benchmarks with 50, 75, 100, 125, 150, and 175 variables.