Late Breaking Results: Differential and Massively Parallel Sampling of SAT Formulas

Arash Ardakani*, Minwoo Kang*, Kevin He, Vighnesh Iyer, Suhong Moon, John Wawrzynek
University of California, Berkeley
{arash.ardakani, minwoo_kang}@berkeley.edu

***Abstract*—Diverse solutions to the Boolean satisfiability (SAT) problem are essential for thorough testing and verification of software and hardware designs, ensuring reliability and applicability to real-world scenarios. We introduce a novel differentiable sampling method, called DIFFSAMPLER, which employs gradient descent (GD) to learn diverse solutions to the SAT problem. By formulating SAT as a supervised multi-output regression task and minimizing its loss function using GD, our approach enables performing the learning operations in parallel, leading to GPU-accelerated sampling and comparable run time performance w.r.t. heuristic samplers. We demonstrate that DIFFSAMPLER can generate diverse uniform-like solutions similar to conventional samplers.**

I. Introduction

Boolean satisfiability (SAT) problem solving is a critical technique in software and hardware design verification, addressing the challenges posed by the complexity and scale of modern systems. Uniform sampling is a core challenge for SAT samplers with applications in a diverse range of areas such as constrained-random simulation, constraint-based fuzzing, configuration testing, and bug synthesis [1]. Modern SAT problem samplers leverage sophisticated algorithms and heuristics to efficiently explore vast solution spaces and provide uniform satisfying solutions to complex logical formulas.

High-throughput sampling stands as a cornerstone for SAT samplers, serving a multitude of essential purposes. Primarily, it enhances efficiency and scalability by facilitating rapid exploration of extensive solution spaces, particularly crucial when tackling real-world problems with numerous variables and constraints. Furthermore, it increases coverage across solution spaces, aiding in identifying rare solutions and nuanced edge cases. Additionally, high-throughput sampling elevates statistical confidence through the generation of larger sample sizes, thus mitigating sampling variance.

While GPU acceleration can offer significant performance benefits across various applications, current state-of-the-art (SOTA) SAT solvers and sampling algorithms are typically executed on CPUs. This is because these solvers rely heavily on sequential processes, which are better suited for CPUs, particularly due to their reliance on branching and backtracking. Instead, to enable GPU acceleration for SAT sampling, we propose a fundamentally different approach. Our method introduces a novel technique that utilizes gradient descent (GD) for learning diverse solutions to the SAT problem. We re-frame the task of sampling formulas for a SAT instance as a supervised multi-output regression task and employ GD to minimize its loss function. This differentiable optimization task resembles typical machine learning problems, is inherently parallel, and can be accelerated using GPUs, allowing for the independent generation of satisfying solutions through a learning process. Due to GPU-accelerated training, our differentiable SAT sampling method achieves a comparable throughput to that of SOTA sampling algorithms, despite implemented in a higher-level programming language (Python), while also generating uniform-like solutions across various benchmarks featuring different numbers of variables and clauses. We refer to our sampling technique as DIFFSAMPLER in this paper. The code of DIFFSAMPLER is available at https://github.com/LBR-DAC-2024/DiffSampler.

*Both authors contributed equally.

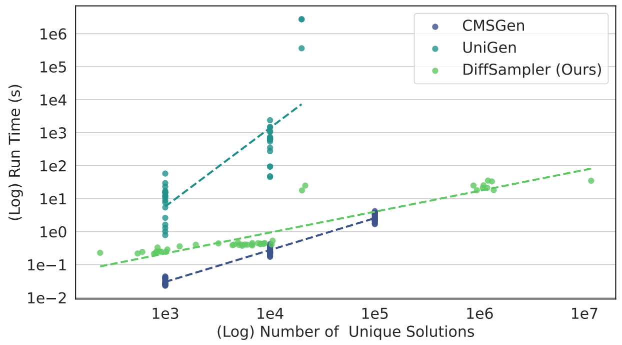

Fig. 1: Log-Log plot of sampler run time in microseconds against the count of unique satisfying solutions found within that run time. A representative subset of 18 SAT problems from the evaluation benchmark are used (or-50-10-7, or-60-20, or-70-5-5, and or-100-20-8).

II. Related Work

Several SAT formula samplers have been developed in literature, including UNIGEN3 [2] with approximate guarantees of uniformity, and CMSGEN[1] and QUICKSAMPLER [3] that emphasize sampling efficiency. Prior work have also used data-parallel hardware for SAT solving, but have largely been limited to parallelizing conflict-driven clause-learning (CDCL) or other heuristic-based SAT solving algorithms [4], [5]. Our method best aligns with the formulation of a SAT instance as a constrained numerical optimization problem as in UNISAT [6], which predates the advent of today's massively parallel hardware. While some work [7] have similarly formulated a relaxed, differential approach to SAT solving, our is the first to effectively showcase the utility of GPU-accelerated formula sampling on standard benchmarks that scale beyond the small, random instances considered in prior work.

III. Methodology

SAT problems are typically expressed in conjunctive normal form (CNF) formulas, comprising m clauses connected by AND operators, with each clause containing l literals connected by OR operators. A literal represents either a variable or the negation of a variable. Finding a satisfying solution to the SAT problem entails assigning values to variables such that all OR gates yield an output of 1. This enables us to reframe the SAT problem as a supervised multi-output regression task, where a solution is derived through a learning process. To facilitate learning, we initially assign random soft values to each variable. Subsequently, we conduct OR operations among literals in all clauses simultaneously, utilizing a probabilistic model for the OR gate. Following the computation of OR gate outputs in terms of probability, we construct a loss function to penalize variables based on the deviation of their associated clauses from generating a 1 output in OR operations. Variables are then updated iteratively, and the process is repeated until convergence is achieved.

To formulate our learning approach, we encode n variables of the SAT instance as parameters (i.e., $\mathbf{V} \in \mathbb{R}^{b\times n}$) of an embedding layer where b denotes the batch size. Let us express m clauses as the matrix $\mathbf{C} \in \mathbb{Z}^{m\times l_{\max}}$ where $l_{\max}$ denotes the maximum number of literals in any single clause in the SAT instance. The clause matrix $\mathbf{C}$ contains indices to the variables where the positive and negative indices denote variables in their true and complementary forms, respectively. We use padding with 0 to achieve consistency in the size of all clauses. To ensure the soft values of variables are represented as a probability value between zero and one, we use the sigmoid function σ. As such, the embedded clauses can be expressed as

$$\mathbf{E} = \sigma(\mathbf{V})\big[\mathbf{C}\big] \in \big[0,1\big]^{b\times m\times l_{\max}}, \tag{1}$$

where the embedded element $e_{ijk} \in \mathbf{E}$ is equal to $v_{it} \in \mathbf{V}$ when its corresponding index is positive; otherwise $1 - v_{it}$ is assigned to e_{ijk}. We reserve the padding index 0 for the noncontributory 0-valued e_{ijk} in the OR operation. The OR operations are computed by

$$\mathbf{Y} = 1 - \prod_{l_{\max}} (1 - \mathbf{E}) \in \big[0,1\big]^{b\times m}, \tag{2}$$

and accordingly the ℓ_2-loss function $\mathcal{L}$ is obtained by

$$\mathcal{L} = \sum_{b,m} ||1 - \mathbf{Y}||_2^2 \,. \tag{3}$$

By computing the loss function which measures the distance between the matrix $\mathbf{Y}$ and expected output 1 for all the clauses across all batches, the variables $\mathbf{V}$ can be updated using GD in an iterative manner. Upon convergence, the batch of soft values (i.e., $\mathbf{V}$) are converted to hard values (i.e., $\widetilde{\mathbf{V}} \in \{0,1\}^{b,n}$) based on their distance from binary values as b solutions to the SAT problem.

IV. Experimental Results

Based on our re-formulation of the SAT problem above, we demonstrate a prototype for DiffSampler, implemented in Python and using a high-performance numerical computing library (JAX). In this Section, we compare our sampler implementation against SOTA baselines (UniGen3 and CMSGen), assessing both in terms of the run time performance and uniformity of the solutions. For comprehensive evaluation. we use a public domain benchmark suite utilized in prior work [1]. Both baseline samplers were executed on server-grade Intel Xeon Gold 6134 CPU with 3.2GHz clock rate and 1TB RAM; DiffSampler results are from running on a system equipped with an Intel Xeon E5-2698 with 2.2GHz clock rate and 8 32GB NVIDIA V100 GPUs.

A. Run time Performance

We selected 60 instances from the uniform sampling benchmark used in prior work, which includes instances from applications as probabilistic reasoning and bounded model checking. Figure 1 summarizes the scaling trends of run time performance against number of unique formulas sampled. We see that (1) DiffSampler is overall much more efficient than UniGen and (2) even compared to CMSGen, our method scales more efficiently to sampling greater numbers of solutions. Direct run time performance, measured in terms of throughput, is also depicted in Table I. For a representative subset of 10 benchmarks, we measured the throughput of each sampler when producing 1000 unique solutions. Although the baselines are highly optimized C++ implementations, resulting in improved run time performance, our method offers comparable run time performance and, in certain cases, even surpasses the baselines for specific instances.

B. Uniformity Measurement

We employed the Barbarik sampler test framework [8] to assess the uniformity of the generated solutions for each benchmark. For evaluation using Barbarik, we adhered to the default parameter settings as recommended by the authors, setting the tolerance parameter ϵ to 0.3, the intolerance parameter η to 1.8, and the confidence parameter δ to 0.1. For all benchmarks in Section IV-A where DiffSampler discovered solutions, Barbarik returned "Accept", confirming the uniformity of the generated solutions. To further analyze uniformity in a quantitative way, we measured the entropy of the distribution between Hamming distances between the generated solutions. The entropy quantifies the degree of variability in the solutions. Higher entropy suggests that solutions are scattered throughout the search space, making it less likely for them to be uniformly distributed, whereas lower entropy suggests more structured and constrained solutions. For majority of benchmark instances, UniGen3 shows the lowest entropy suggesting more uniformity whereas CMSGen presents the highest entropy suggesting more diversity among the generated solutions. DiffSampler, on the other hand, is behaviorally closer to UniGen3. Table II summarizes the distribution statistics of Hamming distances between solutions for a representative subset of 3 benchmarks.

[1]https://zenodo.org/records/3793090

TABLE I: Run time performance, measured in terms of unique solution throughput. Throughput is measured under the case where each method is aimed to produce 1000 unique solutions.

Benchmark	DiffSampler	UniGen3	CMSGen
or-50-10-7-UC-10	75,040.1	64.7	36,693.5
or-70-5-5-UC-30	13,665.6	616.0	36,344.1
or-100-20-8-UC-50	33,728.7	84.4	26,888.6
blasted_1_b12_1	25.8	400.4	10,767.4
blasted_1_b14_3	88.8	97.8	16,495.0
tire-1	35.4	36.8	16,271.4
blasted_1_12_even2	0.7	12.2	1,246.9
blasted_1_14_even	3.3	15.9	4,288.2
modexp-8-4-1	NA	2.8	6.4
hash-02	NA	8.0	1.0

TABLE II: Hamming distance distribution statistics between satisfying solutions sampled by DiffSampler (DS), UniGen3(UG), and CMSGen(CG).

Benchmark	tire-2			blasted_case_1_b12_1			or-100-20-8-UC-10		
Sampler	DS	UG	CG	DS	UG	CG	DS	UG	CG
Range	[2,121]	[3,123]	[3,131]	[7, 157]	[3,30]	[6,200]	[20,75]	[27,91]	[34,111]
Avg	46.0	48.2	69.1	72.7	16.9	116.4	48.0	57.5	74.2
Std	13.1	12.3	19.7	15.5	3.3	24.7	5.9	7.1	8.2
Entropy	5.7	5.6	6.3	6.0	3.8	6.7	4.6	4.9	5.1

V. Conclusion

In this paper, we presented a differentiable sampling method, called DiffSampler, by reframing the SAT problem as a supervised multi-output regression task, allowing the independent generation of satisfying solutions using GD. We then demonstrated that DiffSampler can be accelerated using GPUs due to the parallel nature of its computing paradigm, enabling high-throughput generation of solutions. The experimental results our method show a comparable run time performance and uniformity compared to SOTA sampling techniques.

References

[1] P. Golia *et al.*, "Designing samplers is easy: The boon of testers," in *Proc. of Formal Methods in Computer-Aided Design (FMCAD)*, 2021.

[2] M. Soos *et al.*, "Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling," in *Proceedings of International Conference on Computer-Aided Verification (CAV)*, 2020.

[3] R. Dutra *et al.*, "Efficient sampling of sat solutions for testing," in *Proc. of the International Conference on Software Engineering*, 2018.

[4] C. Costa, "Parallelization of sat algorithms on gpus," Technical report, INESC-ID, Technical University of Lisbon, Tech. Rep., 2013.

[5] M. Osama *et al.*, "Sat solving with gpu accelerated inprocessing," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2021, pp. 133–151.

[6] J. Gu, "Global optimization for satisfiability (sat) problem," *IEEE Trans. on Knowledge and Data Engineering*, vol. 6, no. 3, pp. 361–381, 1994.

[7] T. Sato *et al.*, "Matsat: a matrix-based differentiable sat solver," *arXiv preprint arXiv:2108.06481*, 2021.

[8] Y. Pote *et al.*, "On scalable testing of samplers," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.