Massively Parallel Continuous Local Search for Hybrid SAT Solving on GPUs

Yunuo Cen[1], **Zhiwei Zhang**[2], **Xuanyao Fong**[1*]

[1]National University of Singapore
[2]Rice University
cenyunuo@u.nus.edu, zhiwei@rice.edu, kelvin.xy.fong@nus.edu.sg

Abstract

Although state-of-the-art (SOTA) SAT solvers based on conflict-driven clause learning (CDCL) have achieved remarkable engineering success, their sequential nature limits the parallelism that may be extracted for acceleration on platforms such as the graphics processing unit (GPU). In this work, we propose FastFourierSAT, a highly parallel hybrid SAT solver based on gradient-driven continuous local search (CLS). This is achieved by a parallel algorithm inspired by the fast Fourier transform (FFT)-based convolution for computing the elementary symmetric polynomials (ESPs), which is the major computational task in previous CLS methods. The complexity of our algorithm matches the best previous result. Furthermore, the substantial parallelism inherent in our algorithm can leverage the GPU for acceleration, demonstrating significant improvement over the previous CLS approaches. FastFourierSAT is compared with a wide set of SOTA parallel SAT solvers on extensive benchmarks including combinatorial and industrial problems. Results show that FastFourierSAT computes the gradient 100+ times faster than previous prototypes on CPU. Moreover, FastFourierSAT solves most instances and demonstrates promising performance on larger-size instances.

1 Introduction

Constraint-satisfaction problems (CSPs) are fundamental in mathematics, physics, and computer science. The Boolean satisfiability (SAT) problem is a paradigmatic class of CSPs, where each variable takes values from the binary set {`True`, `False`}. Solving SAT efficiently is of utmost significance in computer science, both from a theoretical and a practical perspective (Kyrillidis et al. 2020). The dominating technique of SAT has evolved from local search (Mitchell, Selman, and Leveque 1992) to DPLL (Davis and Putnam 1960; Davis, Logemann, and Loveland 1962) and CDCL (Marques-Silva and Sakallah 1999). CDCL (Marques-Silva and Sakallah 1999) SAT solvers have been highly successful, constantly solving industrial benchmarks with millions of variables. Numerous problems in various domains are encoded and tackled by SAT solving, e.g., information theory (Golia, Juba, and Meel 2022), VLSI design (Wang, Liu, and Young 2023), and quantum computing (Vardi and Zhang 2023), etc.

Despite the domination of CDCL SAT solvers, most of them rely on a sequential process, where each decision and propagation step depends on the history (Hamadi and Wintersteiger 2013). The sequential nature of CDCL solvers makes it challenging to natively parallelize them and leverage advanced computational resources such as multicore CPU, GPU, and TPU (Jouppi et al. 2017). Most parallel SAT solvers are based on divide-and-conquer and portfolio principles, which only exploit *thread-level* parallelism (Martins, Manquinho, and Lynce 2012). A guiding path is usually used to divide the search space into disjoint subspaces, which are individually searched in parallel for a solution using multiple threads (Zhang, Bonacina, and Hsiang 1996). Different single-thread solvers use different search configurations to diversify the search trajectories (Hamadi, Jabbour, and Sais 2010). In modern SAT solvers, threads may communicate with each other to exchange information (Le Frioux et al. 2017). However, the sequential dependency limits the opportunities for instruction-level parallelism.

Some SAT solvers achieve *instruction-level* parallelism to some extent by incorporating data structures that allow for efficient clause database management, or conflict analysis. CUD@SAT (Dal Palù et al. 2015) proposed a parallel unit propagation algorithm on GPU but gained limited acceleration. GPUShareSat (Prevot, Soos, and Meel 2021) uses GPU for parallel clause checking, thereby minimizing the shared database of a portfolio approach. ParaFROST (Osama, Wijs, and Biere 2021) uses GPU to implement the parallel inprocessing to simplify the clause database.

Motivated by the success in efficiently training neural networks, it is promising to exploit the potential of hardware advancement in machine learning (ML), such as GPU, in SAT solving at thread, instruction, and *data-level*. In this paper, we propose a highly parallelized SAT solver based on continuous local search (CLS). Initiated by FourierSAT (Kyrillidis et al. 2020) and followed by GradSAT (Kyrillidis, Vardi, and Zhang 2021), CLS-based SAT solvers have been recently actively studied as a novel line of SAT framework. These solvers transform SAT into a polynomial optimization problem over the real domain and apply gradient-based optimizers. Compared with Discrete local search (DLS) approaches, CLS has the advantages of high

[*]Corresponding Author: Xuanyao Fong.
Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

convergence quality and native support for non-CNF, i.e., *hybrid* constraints (Kyrillidis et al. 2020). The main performance bottleneck of FourierSAT and GradSAT is, however, the slow gradient computation on the real domain. FourierSAT needs $O(k^3)$ to compute the gradient for a constraint of length k. GradSAT improves this complexity to $O(k^2)$. The computational nature of CLS, especially gradient computation, seems amenable to parallelization, which makes combining CLS and ML-motivated hardware a promising direction. Nevertheless, we reveal that the ideal parallel execution time of FourierSAT and GradSAT is still linear w.r.t the length of constraints, i.e., $O^*(k)$, which is unsatisfactory.

Contributions The theoretical contribution of this paper is a highly parallel algorithm for computing the elementary symmetric polynomials (ESPs), which is the major computation task in CLS approaches. Given a Boolean formula as the conjunction of Boolean constraints, FourierSAT needs to evaluate the Walsh expansion (WE) of the constraint. If the constraints are symmetric, their WEs can be represented by ESPs. Our algorithm is inspired by the FFT-based convolution. We show that the evaluation of WEs of hybrid Boolean constraints can be vectorized. Hence, the gradient computation in FourierSAT can be efficiently implemented by packages frequently used in ML, e.g., JAX (Bradbury et al. 2018). More specifically, we construct the evaluation of the WE as an evaluation trace. By reverse traversal of the trace, the gradient can be obtained due to the chain rule, yielding an algorithm for gradient computation with complexity $O(k^2)$, which matches the best-known complexity in GradSAT. Moreover, our algorithm can be efficiently implemented with parallel computing, with sublinear ideal execution time[1] $O^*(\log k)$ due to the data-level parallelism. By further leveraging the instruction- and thread-level parallelism, the WEs of different constraints can be evaluated on different assignments concurrently. The three-level parallelism maps to the GPU hierarchy of *threads* (data-level), *warps* (instruction-level), and *streaming multiprocessors* (thread-level), enabling ease of GPU acceleration.

We implemented our approach as FastFourierSAT, with massively parallel computation on the GPU platform. The benchmark instances include a variety of combinatorial optimization problems encoded by hybrid (Max)SAT formulas, including *parity learning with errors*, *graph problem*, and instances from SAT competition, MaxSAT evaluation 2023. FastFourierSAT with GPU implementation computes the gradient 100+ times faster than previous CLS algorithms implemented with CPUs, including FourierSAT and GradSAT. We also compare FastFourierSAT with SOTA (parallel) SAT solvers. Results indicate that FastFourierSAT solves most instances and demonstrates promising scalability.

[1]Let $T = T_s + T_p/N$ be the execution time, where T_s, T_p are the serial and parallel components, N is the computational resources. We denote the ideal execution time for $N = \infty$ as $O^*(T)$.

2 Preliminaries

2.1 Boolean Constraints and Formulas

A Boolean constraint c maps all assignments of a set of n Boolean variables to a truth value, i.e., $c : \mathbb{B}^n \to \{\texttt{True}, \texttt{False}\}$. Boolean constraints are often represented by variables and Boolean connectives, e.g., $\wedge$, $\vee$, $\neg$, and $\oplus$.

A Boolean formula $f : \mathbb{B}^n \to \mathbb{B}$ is defined by the conjunction of a set of Boolean constraints $C = \{c_i\}_{i=1}^m$, i.e., $f = \bigwedge_{i=1}^m c_m$. A formula is in conjunctive normal form (CNF) when each constraint is a disjunction of literals. Otherwise, the formula is said to be *hybrid*, which may contain non-CNF constraints, e.g., XOR and cardinality constraint.

2.2 Walsh Expansions of Boolean Constraints

We define a Boolean function by $f : \{\pm 1\}^n \to \{\pm 1\}$ (with an overload to f) where -1 represents `True` and $+1$ for `False`. WE transforms a Boolean function into a multilinear polynomial, such that the polynomial agrees with the Boolean function on all Boolean assignments (O'Donnell 2014). The following theorem shows that every function defined on a Boolean hypercube has an equivalent WE.

Theorem 1 *((O'Donnell 2014), Walsh Transform) Given a function $f : \{\pm 1\}^n \to [-1, 1]$, there is a unique way of expressing f as a multilinear polynomial, called the WE, with at most 2^n terms in S, where each term corresponds to one subset of $[n]$ according to:*

$$f(x) = \sum_{S \subseteq [n]} \left(\hat{f}(S) \cdot \prod_{i \in S} x_i \right),$$

where $\hat{f}(S) \in \mathbb{R}^{2^n}$ is called Walsh coefficient, given S, and computed as:

$$\hat{f}(S) = \frac{1}{2^n} \sum_{x \in \{\pm 1\}^n} \left(f(x) \cdot \prod_{i \in S} x_i \right).$$

Example 1 *(Walsh Expansion, WE) Given a cardinality constraint $c : x_1 + x_2 + x_3 + x_4 \geq 2$, its WE is*

$$\begin{aligned} WE_c(x) = &-\frac{3x_1x_2x_3x_4}{8} - \frac{x_1x_2x_3}{8} - \frac{x_1x_2x_4}{8} - \frac{x_1x_3x_4}{8} \\ &-\frac{x_2x_3x_4}{8} + \frac{x_1x_2}{8} + \frac{x_1x_3}{8} + \frac{x_1x_4}{8} + \frac{x_2x_3}{8} + \frac{x_2x_4}{8} \\ &+\frac{x_3x_4}{8} + \frac{3x_1}{8} + \frac{3x_2}{8} + \frac{3x_3}{8} + \frac{3x_4}{8} - \frac{3}{8}. \end{aligned} \quad (1)$$

All literals in the constraint in Example 1 have the same coefficient. Such a constraint is called *symmetric*, where the Walsh coefficient $\hat{f}(S)$ only depends on $|S|$. Hence the WE can be compactly represented by ESPs.

Definition 1 *(Elementary Symmetric Polynomials, ESPs) The ESPs in n variables $x_1, \cdots, x_n$, denoted by $esp(x) = [e_n(x), \cdots, e_0(x)]$, where $e_0(x) = 1$. For $1 \leq j \leq n$,*

$$e_j(x) = \sum_{1 \leq i_1 < \cdots < i_j \leq n} x_{i_1} \cdots x_{i_j}.$$

Therefore, the WE in Example 1 can be concisely expressed as:

$$WE_c(x) = \hat{f}_c \cdot esp(x)^T, \quad (2)$$

where $\hat{f}_c \in \mathbb{R}^{n+1}$ is the vector of Fourier coefficients of the constraint c, with the i-th element as $\hat{f}(S), |S| = i$.

2.3 Convolution and Fourier Transform

Definition 2 *(Linear Convolution) With $g \in \mathbb{R}^n$ and $h \in \mathbb{R}^m$ as two one-dimension sequences. The linear convolution of g and h is a one-dimension sequence $(g * h) \in \mathbb{R}^{n+m-1}$. Each entry in sequence $(g * h)$ is defined as:*

$$(g * h)[i] = \sum_{j=0}^{i} g[i-j] \cdot h[j].$$

Definition 3 *(Fourier Transform) Let $\omega = e^{\left(-2\pi\sqrt{-1}/n\right)}$. $W \in \mathbb{C}^{n\times n}$ is the discrete Fourier transform (DFT) matrix, with each element as $W[i,j] = \omega^{i\cdot j}$. Fourier transform, denoted by $\mathcal{F}(\cdot)$, converts $x \in \mathbb{R}^n$ into $X \in \mathbb{C}^n$ in frequency domain:*

$$X = \mathcal{F}(x) = W \cdot x.$$

Conversely, the inverse Fourier transform is defined by:

$$x = \mathcal{F}^{-1}(X) = W^{-1} \cdot X.$$

Theorem 2 and Corollary 1 state that the linear convolution ($*$) in the time domain is equivalent to the Hadamard product ($\circ$) in the frequency domain.

Theorem 2 *(Convolution Theorem) With g and h as two one-dimension sequences,*

$$g * h = \mathcal{F}^{-1}\left(\mathcal{F}(g) \circ \mathcal{F}(h)\right).$$

Corollary 1 *With $g_1, \cdots, g_k$ as k one-dimension sequences, and $\gamma_i = \mathcal{F}(g_i)$ for $1 \le i \le k$, we have*

$$g_1 * g_2 * \cdots * g_k = \mathcal{F}^{-1}\left(\gamma_1 \circ \gamma_2 \circ \cdots \circ \gamma_k\right). \quad (3)$$

3 Theoretical Framework

In this section, we first recap two CPU implementations of *Continuous Local Search* (CLS) for hybrid SAT solving and discuss why they are limited for parallelization. We then show how our proposed solver, named FastFourierSAT, can exploit GPU to accelerate gradient computation.

FastFourierSAT consists of three steps: *i.* Fourier transform, *ii.* multiplications in the frequency domain, *iii.* inverse Fourier transform. The complexity of our approach is $O(k^2)$ for computing the gradient of a constraint with length k, which matches previous works. Our algorithm is, moreover, highly parallelizable and the ideal execution time with unlimited resources can be reduced to $O^*(\log k)$. Due to the space limit, we delay most of the proofs to the supplemental materials[2].

[2] Supplemental materials including code, data, and appendix are available on (https://github.com/seeder-research/FastFourierSAT).

Algorithm 1: The CLS Framework for Hybrid SAT Solving

Input: Boolean formula f with a hybrid constraint set C
Output: An discrete assignment $x \in \{-1, 1\}^n$

1: Sample x from $[-1,1]^n$ and initialize weight $w = w_0$
2: **for** $j = 1, \cdots, \texttt{max_iter}$ **do**
3: Search for a local optimum x^* of $F_f(x, w)$
4: **if** $F_f(\texttt{sgn}(x^*), w) = -\sum_{c\in C} w_c$ **then**
5: **return** $\texttt{sgn}(x^*)$
6: $x, w \leftarrow \texttt{Restart}(x^*, w)$
7: **return** $\texttt{sgn}(x^*)$ with lowest $F_f(\texttt{sgn}(x^*), w_0)$

3.1 Recap of CLS-Based SAT Solving

CLS-based SAT solvers define a continuous objective function, the minima of which encode the solutions to the original Boolean formula.

Definition 4 *(Objective) For a formula f with constraint set C, the objective function associated with f is defined as:*

$$F_f(x, w) = \sum_{c\in C} w_c \cdot WE_c(x), \quad (4)$$

where w_c is the weight of c assigned by the CLS algorithm.

Theorem 3 *(Reduction) Given variables $x \in [-1,1]^n$, a Boolean formula f with constraint set C is satisfiable iff $\min F_f(x) = -\sum_{c\in C} w_c$.*

Based on Definition 4 and Theorem 3, previous CLS frameworks can be described in Algorithm 1, which searches for the ground state of the objective function F_f. Global optimization on non-convex functions is, however, NP-hard (Jain, Kar et al. 2017). Since global optima can be identified efficiently by Theorem 3, it is usually more practical to converge to local optima and check if any of them is global as in line 4 of Algorithm 1 (Gu 1994).

FourierSAT and GradSAT (Kyrillidis et al. 2020; Kyrillidis, Vardi, and Zhang 2021) are two gradient-based CLS variants of Algorithm 1. The majority of the computational workload in CLS approaches is line 3 of Algorithm 1. Hence, the performance heavily relies on the speed of gradient computation.

FourierSAT computes the explicit gradient of WE (Eq. 2). Computing the partial derivatives for one literal needs $k-2$ *convolutions*, with complexity $O(k^2)$. Repeating for all literals leads to the following fact.

Fact 1 *(Kyrillidis et al. 2020) For a symmetric Boolean constraint with k literals, FourierSAT computes the gradient via the WE with a time complexity of $O(k^3)$.*

GradSAT encodes the Boolean constraint with binary decision diagram (BDD) (Bryant 1995). By performing the belief propagation on BDDs, the messages can be used to compute the gradient (Pearl 1988; Shafer and Shenoy 1990). The complexity depends on the BDD size (Thornton and Nair 1994). The BDD size of a symmetric Boolean constraint is $O(k^2)$ (Sasao, Fujita et al. 1996).

Fact 2 *(Kyrillidis, Vardi, and Zhang 2021) For a symmetric Boolean constraint with k literals, GradSAT computes the gradient via BDD with a time complexity of $O(k^2)$.*

Algorithm 2: Forward Evaluation with FFT method

Input: The current assignment x
Parameter: Conjugated Fourier coefficient $\tilde{f}_c$'s $\forall c \in C$
Output: Value F_f

1: Initialize $F_f = 0$
2: **for** $c \in C$, **parallel do**
3: $\left[\gamma_{c_1} \cdots \gamma_{c_{|c|}}\right] = \left[1 \cdots \omega^{|c|}\right]^T + x_c$ ▷step 1/3: `ADD`
4: $\gamma_c = \gamma_1 \circ \cdots \circ \gamma_{c_{|c|}}$ ▷step 2/3: `MUL`
5: $\mathrm{WE}_c = \tilde{f}_c \cdot \gamma_c$ ▷step 3/3: `MUL`
6: `atomicAdd(&`F_f, WE_c`)`
7: **return** F_f

Limited Parallelism of Previous Approaches Consider the ideal parallel execution time, denoted by $O^*(\cdot)$, as the minimum required time for executing an algorithm, given unlimited computational resources. FourierSAT needs to convolve the literal sequentially, while GradSAT needs to traverse the BDD layer by layer. Thereafter, the computation from the preceding literal needs to wait until the computation of the previous literal is completed. As a result, the ideal execution time with unlimited resources of the two approaches above is $O^*(k)$ for a constraint with k literals.

3.2 FastFourierSAT

In this subsection, we describe our approach, named FastFourierSAT. We first propose a Fourier-transform-based approach to convert the polynomial computation into a vectorized form, in which the evaluation trace enables efficient parallelization on GPUs. Subsequently, the gradient can be computed by traversing the evaluation trace backward. This approach is also known as *Autodiff*, a fundamental technique that plays a crucial role in ML (Baydin et al. 2018). We reveal that the complexity of the proposed approach is $O(k^2)$. Moreover, our approach runs in the best theoretical execution time of $O^*(\log k)$ with parallel computing.

Fourier Transform-based Evaluation The most expensive operations in FourierSAT are the ESPs. Given a constraint c with a literal set $\{x_1, \cdots, x_k\}$, we observe that the ESPs can be obtained by convolutions of k sequences.

$$esp(x) = [x_1,\ 1] * [x_2,\ 1] * \cdots * [x_k,\ 1] \tag{5}$$

By Corollary 1, the convolution operations in Eq. 5 can be computed by 1) pointwise multiplications of sequences in the frequency domain; followed by 2) inverse FFT of this result. In the following, we describe the details of the three main steps of FastFourierSAT as Lines 3-5 in Algorithm 2.

Step 1 of 3. Fourier Transform by `ADD`: Given a Boolean constraint c with k literals, $esp(x)$ is a one-dimension sequence with $k+1$ entries. Prior to performing batched Fourier transform on sequences $[x_i,\ 1]$ for all $1 \le i \le k$, it is necessary to extend them with 0's, resulting in a sequence with a length of $k+1$, i.e., $g_i = [x_i\ 1\ 0\ \cdots\ 0]$. By Definition 3, the sequences in the frequency domain can be obtained by:

$$\gamma_i^T = W \cdot g_i^T = \left[\ 1 + x_i \quad \omega + x_i \quad \cdots \quad \omega^k + x_i\ \right]^T.$$

Hence, by eliminating the trivial operations, the batched Fourier transform of k sequences can be described as an *outer addition* of a column vector and a row vector:

$$[\gamma_{c_1}\ \gamma_{c_2} \cdots \gamma_{c_k}] = \left[1\ \omega \cdots \omega^k\right]^T + [x_{c_1}\ x_{c_2} \cdots x_{c_k}]. \tag{6}$$

Step 2 of 3. `MUL` in the Frequency Domain: By performing row-wise reduce product for the left-hand-side of Eq. 6, the elements encoding different variables but with the same frequencies are accumulated. The resulting column vector γ_c is the ESPs in the frequency domain of due to Corollary 1.

$$\begin{aligned}\gamma_c &= \gamma_{c_1} \circ \gamma_{c_2} \circ \cdots \circ \gamma_{c_k} \\ &= \left[\prod_{i=1}^{k}(1 + x_{c_i})\ \prod_{i=1}^{k}(\omega + x_{c_i})\ \cdots\ \prod_{i=1}^{k}(\omega^k + x_{c_i})\right]^T\end{aligned} \tag{7}$$

Step 3 of 3. Inverse Transform as Vector `MUL`: By applying the inverse Fourier transform, γ_c can be converted back to the ESPs. The WE is then obtained by multiplying the Fourier coefficients with the ESPs. Thereafter, Eq. 2 can be written as:

$$WE_c(x) = \overbrace{\hat{f}_c \cdot \underbrace{W^{-1}}}^{\tilde{f}_c} \underbrace{\cdot\, \gamma_c(x)}_{esp(x)}. \tag{8}$$

However, given a constraint c, $\hat{f}_c$ and W^{-1} are fixed during the computation time. So $\tilde{f}_c \in \mathbb{C}^{k+1} = \hat{f}_c \cdot W^{-1}$ can be computed in preprocessing, and we denote the resulting row vector as conjugated Walsh coefficient.

Theorem 4 *(Evaluation) Lines 3-5 in Algorithm 2 compute the WE based on FFT. For a symmetric Boolean constraint with k literals, this algorithm runs in $O(k^2)$ time.*

Chain Rule-based Differentiation Given a function f, a computation graph can be constructed with the input and output nodes corresponding to the variables x and the value $WE(x)$. In the forward phase, the function is computed forward with the original operator and recording the intermediate variables γ_i. In the backward phase, derivatives are calculated by the differential operators, and local gradients γ_i' are propagated in reverse. The differential operators are derived based on:

Proposition 1 *(Chain rule) Given x and composite functions $WE_c(\gamma_c(x))$, which is differentiable, then we have:*

$$\left.\frac{\partial WE_c}{\partial x_{c_i}}\right|_x = \left.\frac{\partial WE_c}{\partial \gamma_c}\right|_{\gamma_c(x)} \cdot \left.\frac{\partial \gamma_c}{\partial x_{c_i}}\right|_x.$$

Theorem 5 *(Differentiation) The differential operators derived from Proposition 1 can compute the local gradients with the intermediate variables in Algorithm 2. It runs in $O(k^2)$ time for a symmetric Boolean constraint with k literals.*

In the following, we give an example of how FastFourierSAT evaluates and differentiates the WEs.

Example 2 *(Fourier Transform-based Implicit Gradient) Given a constraint $x_1+x_2+x_3+x_4 \geq 2$, and base frequency $\omega = \exp\left(-2\pi\sqrt{-1}/5\right)$, the conjugated Walsh coefficient is*

$$\tilde{f}_c^T = -\frac{1}{40}\begin{bmatrix} 3 \\ \omega^4-\omega^3-3\omega^2+3\omega+3 \\ -3\omega^4+\omega^3+3\omega^2-\omega+3 \\ -\omega^4+3\omega^3+\omega^2-3\omega+3 \\ 3\omega^4-3\omega^3-\omega^2+\omega+3 \end{bmatrix}.$$

i. Consider $x = (1/2, 1/2, -1/2, -1/2)$, we construct the corresponding column vector $[1/2, 1/2, -1/2, -1/2]^T$. By performing outer addition with $[1, \omega, \omega^2, \omega^3, \omega^4]$, the sequences in frequency domain are:

$$\gamma_1 = \gamma_2 = \left[\frac{3}{2}, \omega+\frac{1}{2}, \omega^2+\frac{1}{2}, \omega^3+\frac{1}{2}, \omega^4+\frac{1}{2}\right]^T,$$
$$\gamma_3 = \gamma_4 = \left[\frac{1}{2}, \omega-\frac{1}{2}, \omega^2-\frac{1}{2}, \omega^3-\frac{1}{2}, \omega^4-\frac{1}{2}\right]^T.$$

ii. By multiplying in the frequency domain along the binary tree, we can have:

$$\gamma_l = \gamma_1 \circ \gamma_2 = \left[\frac{3}{2}, \left(\omega+\frac{1}{2}\right)^2, \cdots, \left(\omega^4+\frac{1}{2}\right)^2\right]^T,$$
$$\gamma_r = \gamma_3 \circ \gamma_4 = \left[\frac{1}{2}, \left(\omega-\frac{1}{2}\right)^2, \cdots, \left(\omega^4-\frac{1}{2}\right)^2\right]^T,$$
$$\gamma_c = \gamma_l \circ \gamma_r = \left[\frac{3}{4}, \left(\omega^2-\frac{1}{4}\right)^2, \cdots, \left(\omega^8-\frac{1}{4}\right)^2\right]^T.$$

iii. As in Eq. 8, the WE of c can be evaluated as

$$WE_c(x) = -\frac{263}{640} + \frac{\omega+\omega^2+\omega^3+\omega^4}{20} = -\frac{59}{128}.$$

From Proposition 1 we can derive:

$$\begin{aligned} x_1' &= \mathbb{1}\cdot\left(\left(\tilde{f}_c \circ \gamma_r\right) \circ \gamma_2\right), \\ &\vdots \\ x_4' &= \mathbb{1}\cdot\left(\left(\tilde{f}_c \circ \gamma_l\right) \circ \gamma_3\right). \end{aligned} \tag{9}$$

where $\mathbb{1}$ is a row vector with all elements as 1. By traversing the evaluation trace backward, i.e., solving Eq. 9 from inner parentheses to outside, we have

$$x' = \left(\frac{19}{64}, \frac{19}{64}, \frac{33}{64}, \frac{33}{64}\right).$$

In the supplemental material, we showed that FourierSAT and GradSAT can achieve the same result.

Acceleration with GPU The majority of the computations in Algorithm 2 are in Lines 3-5. With the multi-threaded computation scheme, the execution can be parallelized. The details are included in the proof of the following proposition.

Proposition 2 *(Parallelism) With unbounded computational resources, the ideal execution time with unlimited resources of Autodiff for Algorithm 2 scales at $O^*(\log k)$.*

Parallel SAT solving refers to the process of solving a SAT formula using multiple computational resources simultaneously. It is motivated by the desire to exploit the computational power of modern multi-core processors to speed up the solving process. We leverage massive parallelism to achieve the ideal execution time in Proposition 2.

i. Data-level Parallelism The majority of computations of FastFourierSAT are matrix operations, which are highly optimized for GPUs. The elementary operations are mainly additions and multiplications, which are simple instructions. Due to the *single instruction multiple data* (SIMD) scheme, concurrent execution of these simple instructions can be efficiently handled by a warp with a group of *threads*.

ii. Instruction-level Parallelism The differentiation of the objective function (Eq. 4) can be subdivided into the differentiation of WEs (Eq. 2) of all Boolean constraints. Since these computations are data independent, they can be instruction-level parallelized, and distributed to identical *warps* in a streaming multiprocessor.

iii. Thread-level Parallelism In CLS solvers, different initialization can be assigned to identical optimizers for parallel search. The search trajectory becomes diversified as optimizers might converge to different local optima. It increases the chance of finding a local optimum being globally optimal. The parallel search can be partitioned and executed by many *streaming multiprocessors*.

Implementation Techniques The essence of CLS is using continuous optimization to find the minima of a non-convex but smooth function. The results depend heavily on the initialization (Jain, Netrapalli, and Sanghavi 2013). To systematically search the continuous domain, we use the following two techniques.

i. Weight Adaptation for Hybrid SAT Solving To leverage the information along the search history, we propose to model the adaptive weight with *exponential recency weighted average*, a lightweight method for progressively approximating a moving average (Liang et al. 2016). Each constraint c maintains an integer $U_c \in [0, p_t]$, which is the number of tasks where c appears to be unsatisfied among the p_t parallel tasks. The unsatisfaction score is

$$r_c = \frac{U_c}{\max_{c \in C_f}(U_c)}.$$

If a constraint c is more on the SAT (resp. UNSAT) side, r_c is closer to 0 (resp. 1).

Consider the unsatisfaction scores along the search history $r_c[1], \cdots, r_c[t]$. We model the adaptive weight w_c using the exponential recency weighted average (Liang et al. 2016) of $r_c[i]$'s, i.e,

$$w_c[i] = \sum_{i=1}^{t} \beta_i r_c[I],$$

Algorithm 3: Integrating CLS for Hybrid MaxSAT Solving

Input: A hybrid constraint set $\phi = \phi_{hard} \cup \phi_{soft}$
Output: An discrete assignment $x \in \{-1, 1\}^n$

1: $\phi_{sol} = \emptyset$
2: `best_cost`=∞
3: **for** $j = 1, \cdots,$ `max_iter` **do**
4: `model=cdcl.solve(`$\phi_{hard} \cup \phi_{sol}$`)`
5: $\phi_{sol} = \phi_{sol} \cup \neg$`model`
6: $x =$`random_init(model)` ▷Polarities of the random initial assignment are fixed by `model`
7: Search for a local optimum x^* of $F_\phi(x)$
8: **if** $F_\phi(x^*) \leq$`best_cost` **then**
9: `cdcl.set_phase(`x^*`)`
10: `best_cost=`$F_\phi(x^*)$
11: $x^{**} = x^*$
12: **return** x^{**}

with decay rate $\beta_i = \alpha(1-\alpha)^{t-i}$ and $\alpha \in [0, 1]$. A constraint adapts to a higher weight if it is frequently unsatisfied along the search history. Therefore, the solver should focus more on the subspace that satisfies constraints with a higher w_c. In practice, the adaptive weight can be efficiently updated as follows. In practice, we use $\alpha = 0.1$ to adapt the weight as follows.

Proposition 3 *(Adaptive weight) The weight of* c *at step* $t+1$ *can be computed by*

$$w_c[t+1] = (1-\alpha)w_c[t] + \alpha r_c[t].$$

ii. Initialization for Hybrid MaxSAT Solving The above weighting heuristic applies to SAT solving with the reduction given by Theorem 3. For MaxSAT solving, we initialize the assignments by solving the hard constraints with CDCL. A valid solution found by CLS satisfying more soft constraints will be set as the CDCL phase. The integration of CDCL and CLS results in FastFourierMaxSAT in Algorithm 3. In our experiments, we use Cadical195 and Gluecard4 from PySAT (Ignatiev, Morgado, and Marques-Silva 2018) to enumerate multiple models, which are used to warm-start the CLS component in parallel.

4 Experimental Results

In this section, we compare our solver FastFourierSAT with CLS and other SOTA SAT solvers. We aim to conduct experiments to answer the following research questions:

RQ1. Can implementing FastFourierSAT on GPU lead to a substantial acceleration in gradient computations compared to prior CPU implementations of CLS?

RQ2. Can FastFourierSAT outperform SOTA parallel solvers in solving (Max)SAT problems that can be naturally encoded with hybrid Boolean constraints?

RQ3. Given industrial-scale problems as (Max)SAT formulas, how does FastFourierSAT perform compared to the CDCL solvers that have dominated over the past decades?

We implement FastFourierSAT in both CPU and GPU frameworks. We use JAX (Bradbury et al. 2018) for the framework and JAXopt (Blondel et al. 2022) for the optimizer. We compare our approach with the following solvers.

Solver Competitors For the hybrid SAT benchmark, we compare with 1) Parallel DLS solvers using CPUs, including PalSAT (Biere 2014) and WalkSAT; 2) Parallel CDCL solvers using CPUs, including PRS (Balyo et al. 2023) and CryptoMiniSAT (CMS) (Soos, Nohl, and Castelluccia 2009); 3) Parallel CDCL solvers using GPUs, including GpuShareSAT (Prevot, Soos, and Meel 2021) and ParaFrost (Osama, Wijs, and Biere 2021). GpuShareSAT has two versions, using Glucose-Syrup or Relaxed-LCMDCBDL-newTech as the base solver. ParaFrost has two versions, which are complied for CPU or GPU. The portfolio results in GSS-Glucose+Relax and ParaFrost-cpu+gpu.

For the hybrid MaxSAT benchmark, we compare with 4) LinPB (Yang and Meel 2021). 5) Solvers from *MSEval'23*, including NoSAT-MaxSAT (Schupp 2023), Loandra (Berg 2023), NuWLS-c (Chu, Cai, and Luo 2023).

Experimental Setup The CPU solvers run on a server with dual AMD Epyc 7773X CPUs and 2 TB of RAM. Each experiment for CPU solvers uses 32 CPU threads, and 128 GB of RAM. The GPU solvers run on a workstation with INTEL i9-12900F CPU, 32 GB of RAM, and NVIDIA RTX 3080 Ti GPU. Timeout is 300 s unless specifically stated.

Encodings CLS solvers accept hybrid constraints via WEs or BDDs. Solvers who do not accept the cardinality constraints use the encoding from (Ansótegui et al. 2021), which automatically chooses the best encoding. Except for CMS and LinPB, solving XOR constraints by Gaussian elimination, other non-CLS solvers encode XOR by (Li 2000).

Benchmark 1: Gradient Computation Time To evaluate the gradient computation time, we generate $m \in \{100, 200, 400\}$ cardinality constraints, with length $l \in \{8, 16, 32\}$. We evaluate the above constraints with 10000 points from $[-1, 1]^n$ for average gradient computation time.

Benchmark 2: Hybrid SAT formulas This benchmark includes two types of instances. 1) **Random Cardinality Constraints**. For each $N \in \{50, 100, 150, 200, 250\}$, we choose $l = N/5$ and $m = 3l$ to generate 100 instances with m random $(l/2)$-cardinality constraints. Each constraint sampled l variables as the literals with $1/2$ probability of all positive or all negative. 2) **Parity Learning with Error Problems** aim to find an assignment that can satisfy at least $(1-e)\cdot m$ out of m XOR constraints. For $e \in (0, 1/2)$, whether the problem is in P remains an open question and is known to be hard for local search solver (Crawford, Kearns, and Schapire 1994). For each $N \in \{20, 30, 40, 50, 60\}$, we choose $e = 1/4$ and $m = 2N$ to generate 100 hard instances. CLS solvers can solve them natively with at most $e \cdot m$ XOR constraints violated. Otherwise, we used the encoding due to (Hoos and Stützle 2000).

Benchmark 3: Hybrid MaxSAT for graph problems Soft XOR constraints are added to the formulas in this benchmark. We harden the soft XOR constraints by (Soos and Meel 2021) for non-CLS solvers. Finding the maximal number of satisfied XOR constraints can be useful in

CLS Solvers	Average Gradient Computation Time (ms)								
	8_100	8_200	8_400	16_100	16_200	16_400	32_100	32_200	32_400
FastFourierSAT-gpu	1.75	1.77	1.81	**1.81**	**1.79**	**1.80**	**1.82**	**1.88**	**1.91**
FastFourierSAT-cpu	**1.58**	**1.70**	**1.76**	1.84	2.13	3.06	3.09	4.44	7.57
GradSAT	3.93	8.80	19.60	18.40	41.06	94.15	90.76	208.08	472.31
FourierSAT	25.95	53.08	106.47	87.37	179.56	356.29	359.66	743.84	1280.89

Table 1: The average gradient computation time of different CLS solvers. l_m means m cardinality constraints with length l.

hashing-based approximate model counting (Chakraborty, Meel, and Vardi 2021). This benchmark includes two types of graph problems encoded by hybrid formulas. Let $G(V, E)$ be an undirected graph, 1) **Hamiltonian Cycle Problem** is to find a path from edges E that visits each vertex in V exactly once. Problems are encoded in $N = |V|^2$ variables $\{x_{v,i}\}$, where v refers to the vertex and i is the order in a prospective cycle. It uses hard *exact-one* constraints for one-hot encoding, i.e., $\sum_v x_{v,i} = 1, \forall i$ and $\sum_i x_{v,i} = 1, \forall v$. Hard binary clauses $(\bar{x}_{u,i} \vee \bar{x}_{v,i+1}) \wedge (\bar{x}_{v,i} \vee \bar{x}_{u,i+1}), \forall (u,v) \notin E$ enforce the path can only be constructed by a subset of E. For each $|V| \in \{10, 20, 30, 40, 50\}$, we choose degree $|V|/2$ to generate 10 regular graphs. $|V|^2$ soft XOR constraints are added, each with a length of $N/2$. 2) **Graph Coloring Problem** is to assign colors to vertices such that no two adjacent vertices have the same color. Given $|C|$ colors, we encode the problem in $N = |V| \cdot |C|$ variables $\{x_{v,i}\}$, where v and i refer to the vertex and i represent the color. It uses hard *exact-one* constraints for assigning exactly one color to every vertices, i.e., $\sum_i x_{v,i} = 1, \forall v$. Every edge $(u, v) \in E$ use hard binary clauses $\bar{x}_{u,i} \vee \bar{x}_{v,i}$ to avoid adjacent vertices having the same color. For each $|V| \in \{100, 200, 300, 400, 500\}$, we choose degree $|C| = |V|/10$ to generate 10 regular graphs. $|V|$ soft XOR constraints are added, each with a length of $N/2$.

Benchmark 4: SAT Competition (SC) and MaxSAT Evaluation (MSEval) 2023 Since CLS is incomplete, we select the instances proven to be satisfiable from *SC'23* and all unweighted instances from anytime track in *MSEval'23*.

Scoring Metrics We use the PAR-2 score for evaluating the satisfaction problems. The PAR-2 score is defined to be its average runtime over all instances, whereby unsolved instances contribute twice the time limit. We use the anytime score for evaluating the optimization problems. The anytime score of a solver s on instance i is the ratio between 1 plus the best solution found by s and 1 plus the best-known solution for i.

RQ1. Table 1 showcases the results of Benchmark 1. GradSAT and FourierSAT use for-loops to compute the gradients for various constraints. FastFourierSAT, as the vectorized version of FourierSAT, leverages the highly optimized BLAS (Basic Linear Algebra Subprograms) for matrix multiplication. There has been a great improvement in using the CPU implementation of FastFourierSAT, especially for small instances. The GPU implementation offers scalability, despite suffering from the data communication between the

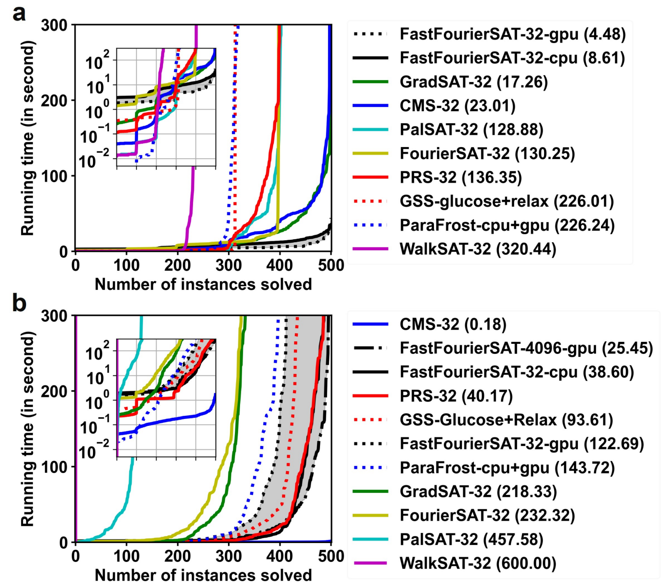

Figure 1: Results on **a**) cardinality constraints and **b**) parity learning problems from benchmark 2. Dashed lines are referred to as GPU solvers. Legends are ranked by PAR-2 scores, with the numbers in parentheses indicating each entry's specific PAR-2 score.

CPU and GPU. For the largest instance, it achieves 472× and 1280× speedup w.r.t. GradSAT and FourierSAT.

RQ2. The results of benchmark 2 are shown in Fig. 1. GPU-enabled CDCL solvers, *i.e.*, GSS-glucose+relax (312), ParaFrost-cpu+gpu (318), perform worse than the CPU CDCL solvers, *i.e.*, CMS (497), PRS (399)[3]. It may be due to CDCL solvers being memory-intensive for clause learning while GPUs have limited RAM. The cardinality constraints can be natively accepted by CLS solvers where GradSAT-32 and FastFourierSAT can solve all instances. FourierSAT-32 solves only 397 instances due to the slow gradient computation with non-parallelized convolution operations. FastFourierSAT-cpu-32 achieves a 15.13× speedup in terms of PAR-2 score while using GPU achieves another 1.92× speedup. From the inset of Fig. 1a, we observe that as the size of the instance increases, the running time

[3]The number in parentheses represents the number of instances that the solver solved within the time limit.

Solvers	Anytime Scores	
	Hamiltonian Cycle	Graph Coloring
FastFourierMaxSAT-32-gpu	0.989	0.938
FastFourierMaxSAT-32-cpu	0.981	0.920
LinPB	0.983	0.914
Loandra	0.976	0.893
NuWLS-c	0.962	0.881
FastFourierSAT-32-gpu	0.948	0.861
FastFourierSAT-32-cpu	0.884	0.621
NoSAT-MaxSAT	0.359	0.858
GradSAT-32	0.352	0.135
FourierSAT-32	0.205	0.132

Table 2: Results on graph problems from benchmark 3. The anytime score refers to the scoring metric in *MSEval'23*.

for most solvers increases significantly. Nevertheless, FastFourierSAT seems to be more scalable for larger formulas.

Parity learning problems were easily solved by CMS (500), which handles XORs by Gaussian elimination. The inset of Fig. 1b shows that CMS can solve most instances within 1 s. In contrast, DLS perform poorly, *e.g.*, PalSAT (129), and WalkSAT (0). FastFourierSAT-32-cpu (486) solving more instances than -gpu (412) seems counter-intuitive. FastFourierSAT-4096-gpu increases the GPU utilization rate and solves 495 instances.

RQ3. Benchmark 4 tests CLS solvers on satisfiable instances from *SC'23* with a 1000 s timelimit. FourierSAT-32 fails to solve any instance and GradSAT-32 solves 1 instance. FastFourierSAT-32-cpu (resp. -gpu) can solve 5 and (resp. 13) instances. However, these instances are easy for the solvers from *SC'23*, which take minimally 0.04 to 1.07 s.

We also test CLS solvers on the unweighted instances from anytime track in *MSEval'23*. FourierSAT-32 fails to give valid solutions. GradSAT-32 uses a portfolio optimizer and finds solutions with a score of 0.102. FastFourierMaxSAT-32-cpu (resp. -gpu) achieves scores of 0.321 (resp. 0.327). The improvement of using GPU is not significant, where the performance depends mostly on the CDCL-based initialization. For the ablation study, we test FastFourierSAT-32-cpu (resp. -gpu) on these instances, achieving scores of 0.093 (resp. 0.116). In contrast, the VBS solver from *MSEval'23* achieves an anytime score of 0.954.

The above results can conclude that given industrial problems, FastFourier(Max)SAT might not be competitive with mature CDCL solvers submitted to the competitions. The instances from competitions are in (W)CNF format and are encoded to be arc-consistent, where the CDCL solvers can capture the semantics and do efficient propagation. The competition solvers are crafted with well-engineered heuristics to achieve good performance. Usually, an arc-consistent encoding yields a larger formula than that of a non-arc-consistent encoding. While CLS solvers cannot capture the arc consistency, they need to solve a larger formula.

To see the performance loss, we try FastFourierSAT on the (W)CNF formulas from *RQ2*. Recall that FastFourierSAT is competitive with SOTA solvers when solving hybrid formulas. However, for *cardinality constraints* encoded in CNF, FastFourierSAT-32-cpu (resp. -gpu) can solve 120 (resp. 195) instances with PAR-2 score of 466.961 (resp. 373.579). *Parity learning* is challenging for local search, where FastFourierSAT-32-cpu (resp. -gpu) can solve 1 (resp. 27) instance with PAR-2 score of 598.995 (resp. 578.938). And FastFourierSAT fails to solve any instance from benchmark 3 encoded by WCNF. This is because a compact hybrid formula is encoded to an extremely large-size SAT formula. For example, a *graph coloring* instance originally has 2.5 K variables, 60 K clauses, 100 exact-one constraints, and 2.5 K XOR constraints. However, the encoded WCNF formula has more than 6 M variables and 26 M clauses.

5 Conclusion

The present study introduces a novel FFT-inspired approach for accelerating the evaluation of ESPs, which significantly enhances the CLS approach for hybrid SAT solving. The acceleration is achieved by leveraging the thread-, instruction-, and data-level parallelism on the GPU, which enables efficient gradient computation. Our results demonstrate that FastFourierSAT is competitive with the SOTA solvers in a few benchmarks. They also highlight that, with the massively parallel scheme, CLS approaches can complement the existing CDCL solvers for solving non-CNF constraints. We list the *limitations* and *future directions* in the following.

Rounding errors Rounding errors from Walsh coefficients and the computations in the frequency domain reduce the numerical stability for CLS. This work mitigates the errors by using higher bit precision, which comes with a notable computation and memory overhead. We are interested in computationally efficient methods for improving the numerical stability.

Global Optimization and Completeness CLS, as an incomplete framework, relies on local optimization to search the satisfiable assignments of the Boolean formulas. While CLS cannot prove unsatisfiability through techniques like deletion resolution asymmetric tautology (DRAT) employed by CDCL solvers (Wetzler, Heule, and Hunt Jr 2014), we can borrow ideas from continuous optimization methods that provide guarantees of a global minimum. We aim to develop a global optimizer based on the gradient of the Moreau envelope of the objectives (Osher, Heaton, and Wu Fung 2023). We are interested in studying the completeness of global optimization with the Moreau envelope.

Acknowledgments

We thank Moshe Vardi for the helpful and insightful comments and discussions that contributed to this work. This work was supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Competitive Research Program (NRF-CRP24-2020-0002 and NRF-CRP24-2020-0003), the Ministry of Education (Singapore) Tier 2 Academic Research Fund (MOE-T2EP50220-0012 and MOE-T2EP50221-0008). Zhiwei Zhang is sup-

ported in part by NSF grants (IIS-1527668, CCF1704883, IIS1830549), DoD MURI grant (N00014-20-1-2787), Andrew Ladd Graduate Fellowship of Rice Ken Kennedy Institute, and an award from the Maryland Procurement Office.

References

Ansótegui, C.; Ojeda, J.; Pacheco, A.; Pon, J.; Salvia, J. M.; and Torres, E. 2021. Optilog: A framework for sat-based systems. In *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*, 1–10. Springer.

Balyo, T.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M., eds. 2023. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Finland: Department of Computer Science, University of Helsinki.

Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; and Siskind, J. M. 2018. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18: 1–43.

Berg, J. 2023. Loandra in the 2022 (and 2023) MaxSAT Evaluation. *MaxSAT Evaluation 2023*, 21.

Biere, A. 2014. Yet another Local Search Solver and Lingeling and Friends Entering the SAT Competition 2014. In Balint, A.; Belov, A.; Heule, M.; and Järvisalo, M., eds., *Proc. of SAT Competition 2014 – Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, 39–40. University of Helsinki.

Blondel, M.; Berthet, Q.; Cuturi, M.; Frostig, R.; Hoyer, S.; Llinares-López, F.; Pedregosa, F.; and Vert, J.-P. 2022. Efficient and modular implicit differentiation. *Advances in neural information processing systems*, 35: 5230–5242.

Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; and Zhang, Q. 2018. JAX: composable transformations of Python+NumPy programs.

Bryant, R. E. 1995. Binary decision diagrams and beyond: Enabling technologies for formal verification. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, 236–243. IEEE.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2021. Approximate model counting. In *Handbook of Satisfiability*, 1015–1045. IOS Press.

Chu, Y.; Cai, S.; and Luo, C. 2023. NuWLS-c-2023: Solver description. *MaxSAT Evaluation 2023*, 23.

Crawford, J. M.; Kearns, M. J.; and Schapire, R. E. 1994. The minimal disagreement parity problem as a hard satisfiability problem. *Computational Intell. Research Lab and AT&T Bell Labs TR*.

Dal Palù, A.; Dovier, A.; Formisano, A.; and Pontelli, E. 2015. Cud@ sat: Sat solving on gpus. *Journal of Experimental & Theoretical Artificial Intelligence*, 27(3): 293–316.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *Communications of the ACM*, 5(7): 394–397.

Davis, M.; and Putnam, H. 1960. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3): 201–215.

Golia, P.; Juba, B.; and Meel, K. S. 2022. A Scalable Shannon Entropy Estimator. In *International Conference on Computer Aided Verification*, 363–384. Springer.

Gu, J. 1994. Global optimization for satisfiability (SAT) problem. *IEEE Transactions on Knowledge and Data Engineering*, 6(3): 361–381.

Hamadi, Y.; Jabbour, S.; and Sais, L. 2010. ManySAT: a parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6(4): 245–262.

Hamadi, Y.; and Wintersteiger, C. 2013. Seven challenges in parallel SAT solving. *AI Magazine*, 34(2): 99–99.

Hoos, H. H.; and Stützle, T. 2000. SATLIB: An online resource for research on SAT. *Sat*, 2000: 283–292.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, 428–437.

Jain, P.; Kar, P.; et al. 2017. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4): 142–363.

Jain, P.; Netrapalli, P.; and Sanghavi, S. 2013. Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 665–674.

Jouppi, N. P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, 1–12.

Kyrillidis, A.; Shrivastava, A.; Vardi, M.; and Zhang, Z. 2020. FourierSAT: A Fourier expansion-based algebraic framework for solving hybrid boolean constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 1552–1560.

Kyrillidis, A.; Vardi, M.; and Zhang, Z. 2021. On continuous local BDD-based search for hybrid SAT solving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 3841–3850.

Le Frioux, L.; Baarir, S.; Sopena, J.; and Kordon, F. 2017. PaInleSS: a framework for parallel SAT solving. In *Theory and Applications of Satisfiability Testing–SAT 2017: 20th International Conference, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 20*, 233–250. Springer.

Li, C. M. 2000. Integrating equivalency reasoning into Davis-Putnam procedure. *AAAI/IAAI*, 2000: 291–296.

Liang, J.; Ganesh, V.; Poupart, P.; and Czarnecki, K. 2016. Exponential recency weighted average branching heuristic for SAT solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Marques-Silva, J. P.; and Sakallah, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5): 506–521.

Martins, R.; Manquinho, V.; and Lynce, I. 2012. An overview of parallel SAT solving. *Constraints*, 17: 304–347.

Mitchell, D.; Selman, B.; and Leveque, H. 1992. A new method for solving hard satisfiability problems. In *Proceedings of the tenth national conference on artificial intelligence (AAAI-92)*, 440–446.

O'Donnell, R. 2014. *Analysis of boolean functions*. Cambridge University Press.

Osama, M.; Wijs, A.; and Biere, A. 2021. SAT solving with GPU accelerated inprocessing. In *Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27–April 1, 2021, Proceedings, Part I 27*, 133–151. Springer.

Osher, S.; Heaton, H.; and Wu Fung, S. 2023. A Hamilton–Jacobi-based proximal operator. *Proceedings of the National Academy of Sciences*, 120(14): e2220469120.

Pearl, J. 1988. Belief updating by network propagation. In *Probabilistic Reasoning in Intelligent Systems*, 143–237. Morgan Kaufmann San Francisco (CA).

Prevot, N.; Soos, M.; and Meel, K. S. 2021. Leveraging GPUs for Effective Clause Sharing in Parallel SAT Solving. In *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, 471–487. Springer.

Sasao, T.; Fujita, M.; et al. 1996. *Representations of discrete functions*. Springer.

Schupp, O. L. S. 2023. noSAT-MaxSATv2. *MaxSAT Evaluation 2023*, 27.

Shafer, G. R.; and Shenoy, P. P. 1990. Probability propagation. *Annals of mathematics and Artificial Intelligence*, 2: 327–351.

Soos, M.; and Meel, K. S. 2021. Gaussian elimination meets maximum satisfiability. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 18, 581–587.

Soos, M.; Nohl, K.; and Castelluccia, C. 2009. Extending SAT solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, 244–257. Springer.

Thornton, M.; and Nair, V. 1994. Efficient spectral coefficient calculation using circuit output probabilities. *Digital Signal Processing*, 4(4): 245–254.

Vardi, M. Y.; and Zhang, Z. 2023. Solving Quantum-Inspired Perfect Matching Problems via Tutte's Theorem-Based Hybrid Boolean Constraints. *arXiv preprint arXiv:2301.09833*.

Wang, F.; Liu, J.; and Young, E. F. 2023. FastPass: Fast Pin Access Analysis with Incremental SAT Solving. In *Proceedings of the 2023 International Symposium on Physical Design*, 9–16.

Wetzler, N.; Heule, M. J.; and Hunt Jr, W. A. 2014. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, 422–429. Springer.

Yang, J.; and Meel, K. S. 2021. Engineering an efficient PB-XOR solver. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

Zhang, H.; Bonacina, M. P.; and Hsiang, J. 1996. PSATO: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21(4-6): 543–560.