Artificial Intelligence 299 (2021) 103559

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Solving hybrid Boolean constraints in continuous space via multilinear Fourier expansions[☆]

Anastasios Kyrillidis, Anshumali Shrivastava, Moshe Y. Vardi, Zhiwei Zhang[*]

Rice University, Houston, TX, USA

ARTICLE INFO

Article history:
Received 8 June 2020
Received in revised form 17 July 2021
Accepted 19 July 2021
Available online 27 July 2021

Keywords:
SAT solving
Multilinear optimization
Fourier analysis on Boolean functions
Walsh-Hadamard transform

ABSTRACT

The Boolean SATisfiability problem (SAT) is of central importance in computer science. Although SAT is known to be NP-complete, progress on the engineering side—especially that of Conflict-Driven Clause Learning (CDCL) and Local Search SAT solvers—has been remarkable. Yet, while SAT solvers, aimed at solving industrial-scale benchmarks in Conjunctive Normal Form (CNF), have become quite mature, SAT solvers that are effective on other types of constraints (e.g., cardinality constraints and XORs) are less well-studied; a general approach to handling non-CNF constraints is still lacking.
To address the issue above, we design FourierSAT,[1] an incomplete SAT solver based on Fourier Analysis (also known as Walsh-Fourier Transform) of Boolean functions, a technique to represent Boolean functions by multilinear polynomials. By such a reduction to continuous optimization, we propose an algebraic framework for solving systems consisting of different types of constraints. The idea is to leverage gradient information to guide the search process in the direction of local improvements. We show this reduction enjoys interesting theoretical properties. Empirical results demonstrate that FourierSAT can be a useful complement to other solvers on certain classes of benchmarks.

1. Introduction

1.1. Background

Constraint satisfaction problems (CSPs) are fundamental in mathematics, physics, and computer science. The Boolean SATisfiability problem (SAT) is a special class of CSPs, where each variable takes value from the binary set {True, False}. Solving SAT efficiently is of utmost significance in computer science, both from a theoretical and a practical perspective. Applications of SAT solving are in combinatorial optimization [1], software verification [2], probabilistic inference [3], mathematical conjecture proving [4] and so on.

As a special case of SAT, conjunctive normal forms (CNFs) are a conjunction (and-ing) of disjunctions (or-ing) of literals. Despite the NP-completeness of CNF-SAT, there has been a lot of progress on the engineering side of CNF-SAT solvers.

[☆] The author list has been sorted alphabetically by last name; this should not be used to determine the extent of authors' contributions.
[*] Corresponding author.
E-mail address: zhiwei@rice.edu (Z. Zhang).
URL: https://www.cs.rice.edu/~zz59/ (Z. Zhang).
[1] The code and benchmarks are available at https://github.com/vardigroup/FourierSAT.

https://doi.org/10.1016/j.artint.2021.103559
0004-3702/© 2021 Elsevier B.V. All rights reserved.

Mainstream SAT solvers can be classified into complete and incomplete ones: A complete SAT solver will return a solution if there exists one or prove unsatisfiability if no solution exists, while an incomplete algorithm is not guaranteed to find a satisfying assignment. Clearly, an incomplete algorithm cannot prove unsatisfiability.

Most modern complete SAT solvers are based on the Conflict-Driven Clause Learning (CDCL) algorithm introduced in GRASP [5], an evolution of the backtracking Davis-Putnam-Logemann-Loveland (DPLL) algorithm [6,7]. Examples of highly efficient complete SAT solvers include Chaff [8], MiniSat [9], PicoSAT [10], Lingeling [11], Glucose [12], and machine-learning-enhanced MapleSAT [13]. Overall, CDCL-based SAT solvers constitute a huge success for SAT problems, and have been dominating in the research of SAT solving.

Local search techniques are mostly used in incomplete SAT solvers. The number of unsatisfied clauses is often regarded as the objective function. Local search algorithms mainly include greedy local search (GSAT) [14] and random walk GSAT (WSAT) [15]. During the main loop, GSAT repeatedly checks the current assignments neighbors and selects a new assignment to maximize the number of satisfied clauses. In contrast, WSAT randomly selects a variable in an unsatisfied clause and inverts its value [16]. On top of these basic algorithms, several efficient variants of GSAT and WSAT have been proposed, such as NSAT [17], Sparrow [18], Novelty+ [19], SAPS [20,21] and ProbSAT [22]. While practical local-search solvers could be slower than CDCL solvers, local search techniques are still useful for solving a certain class of benchmarks, such as hard random formulas and MaxSAT [14,23]. Local search algorithms can also have nice theoretical properties [24]. Though there exist projects regarding proving unsatisfiability by local search [25,26], they fail to scale as well as resolution-based complete solvers.

Non-CNF constraints are playing important roles in theoretical computer science and other engineering areas, *e.g.*, XOR constraints in cryptography [27] as well as cardinality constraints (CARD) and Not-All-Equal (NAE) constraints in discrete optimization [28,29]. The combination of different types of constraints enhances the practical expressive power of Boolean formulas; e.g., CARD-XOR is necessary and sufficient for maximum likelihood decoding (MLD) [30], one of the most crucial problems in coding theory. Nevertheless, compared to that of CNF-SAT solving, efficient SAT solvers that can handle non-CNF constraints are less well studied.

One way to deal with non-CNF constraints is to encode them in CNF [31,32]. Different encodings, however, differ in size, the ability to detect inconsistencies by unit propagation (arc consistency) and solution density [33]. It is generally observed that the running time of SAT solvers relies heavily on the details of encodings. E.g., CDCL solvers benefit from arc-consistency [34], while local search solvers prefer short chains of variable dependencies [35]. Finding a best encoding for a solver usually requires considerable testing and comparison [36].

Another way is to extend the existing SAT solvers to adapt to non-CNF constraints. Work on this line includes Cryptominisat [37] for CNF-XOR, MiniCARD [38] for CNF-CARD, Pueblo [39] and RoundingSAT [40] for CNF-Pseudo Boolean and MonoSAT [41] for handling CNF-graph-property. Such specialized extensions, however, often require different techniques for different types of constraints. Meanwhile, general ideas for solving hybrid constraints uniformly are still lacking.

1.2. Contributions

The primary contribution of this work is the design of a novel algebraic framework as well as a versatile, robust incomplete SAT solver—FourierSAT—for solving hybrid Boolean constraints.

The main technique we used in our approach is that of *Fourier Expansions* (also known as *Walsh Transform* of Boolean functions [42]). By transforming Boolean functions into "nice" polynomials, many properties can be analyzed mathematically. Besides the success of the Fourier Expansions in theoretical computer science [43,44], recently, this technique has also found surprising uses in algorithm design [45,46]. We believe that more algorithmic applications of this technique are still waiting to be discovered.

The workflow of our method is illustrated in Fig. 1. First, each constraint of the hybrid Boolean formula f is transformed to a multilinear polynomial. Second, the objective function F is constructed by summing up all multilinear polynomials. Then the gradients of the objective function are computed. Afterwards, we relax the domain and apply continuous optimization methods to get a local minimum of F. Finally, the local minimum is discretized and we verify if the Boolean assignment is a solution. If not, we restart from a random point and optimize F again.

To our best knowledge, this paper is the first algorithmic work to study Fourier Expansions of Boolean functions as polynomials in the real domain instead of only Boolean domain. After lifting the domain, we find Fourier Expansions to be well-behaved. Thus, we manage to reduce satisfiability of Boolean constraints to continuous optimization and apply gradient-based methods. One of the attractive properties of our method is, that different types of constraints are handled uniformly—we no longer design specific methods for each type of constraints. Moreover, as long as the Fourier Expansions of a new type of constraints are evaluable, our solver can be extended to adapt the new constraints trivially. In addition, we explain the intuition of why doing continuous optimization for SAT is better than doing discrete local search.

Furthermore, our study on the local landscape of Fourier Expansions reveals that the existence of saddle points is an obstacle for us to design algorithms with theoretical guarantees. Previous research shows that gradient-based algorithms are in particular susceptible to saddle point problems [47]. Although the study in [48,49] indicates that stochastic gradient descent with random noise is enough to escape saddle points, strict saddle property, which is not valid in our case, is assumed in the work above. Therefore, we design specialized algorithms for optimizing Fourier Expansions. Finally, we demonstrate, by experimental results, that for certain classes of hybrid formulas and MaxSAT instances, FourierSAT shows promising

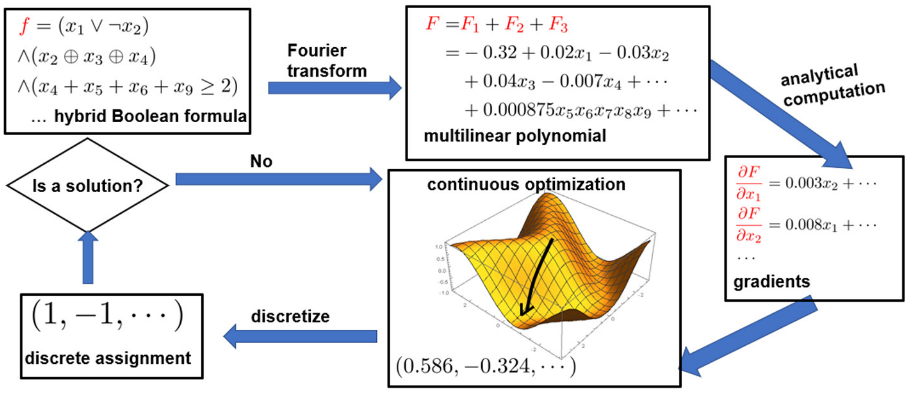

$f = (x_1 \vee \neg x_2)$
$\wedge(x_2 \oplus x_3 \oplus x_4)$
$\wedge(x_4 + x_5 + x_6 + x_9 \geq 2)$
... hybrid Boolean formula
Fourier transform
$F = F_1 + F_2 + F_3$
$= -0.32 + 0.02x_1 - 0.03x_2$
$+ 0.04x_3 - 0.007x_4 + \cdots$
$+ 0.000875x_5x_6x_7x_8x_9 + \cdots$
multilinear polynomial
analytical computation
$\frac{\partial F}{\partial x_1} = 0.003x_2 + \cdots$
$\frac{\partial F}{\partial x_2} = 0.008x_1 + \cdots$
gradients
Is a solution?
No
continuous optimization
$(0.586, -0.324, \cdots)$
discretize
$(1, -1, \cdots)$
discrete assignment

Fig. 1. Workflow of FourierSAT. First, each constraint of the hybrid Boolean formula f is transformed to a multilinear polynomial. Second, the objective function is constructed by summing all multilinear polynomials up. Then, the gradients of the objective function are computed. Afterwards, we relax the domain and apply continuous optimization methods to get a local minimum of F. Finally, the local minimum is discretized and we verify if the Boolean assignment is a solution. If not, we start from a random point and optimize F again.

potential to be a reliable solver. We believe that the natural reduction from SAT to continuous optimization, combined with state-of-the-art constrained-continuous optimization techniques, opens a new line of research on SAT solving. This paper, based on a Master's thesis [50], extends our AAAI 2020 conference paper [51] by providing the following novel contributions.

- An extension on the applicable types of constraints from {disjunctive clauses (`CNF` clauses), `XOR`, cardinality constraints (`CARD`), Not-All-Equal (`NAE`)} to all symmetric constraints.
- Self-contained, detailed proofs of all theoretical results.
- Algorithms for using second-order information to escape local optima.
- Extensive new experimental results (on MaxSAT instances), illustrations, examples, remarks, and related work.
- A historical note for Walsh-Hadamard-Fourier Transform.

1.3. *Related work*

Our work falls into the area of applying algebraic approaches for solving discrete logical reasoning problems, including SAT solving and Boolean refutation. As an attempt in this area, Discrete Lagrange Method (DLM) [52] considers the KKT condition in discrete setting and searches in the primal-dual space. DLM is, however, essentially equivalent to discrete local search algorithms with dynamic weight updating [53].

Our approach shares a similar core spirit with ZAP [54], in the sense of creating a uniform framework for handling various Boolean constraints. ZAP exploits the symmetry of clauses and represents Boolean instances compactly by group elements while we use polynomials.

Since our framework is based on polynomial-optimization, to our best knowledge, it finds connections with the following work in this more specific area. An important line of work in this field is to use a theoretically powerful proof system, Gröbner basis [55–57] for proving unsatisfiability. Besides giving refutations of satisfiability, Gröbner basis is also used for pre-processing the CNF formula to provide a good initial point [58]. As for other directions, the authors of [59] used "balanced polynomial", a polynomial representation similar to Fourier Expansion, to design an efficient algorithm for solving parity learning problem. The main framework of their solver, however, is still DPLL-based and the mathematical properties of polynomials are not sufficiently leveraged. Another direction that has been followed previously is the transformation between CNF and ANF [60,61]. While ANFs are polynomials in field $\mathbb{F}_2^n$, our polynomials are in $\mathbb{R}^n$. An alternative high-order polynomial representation of the objective function used in local search is proposed in [62]. Though those polynomials are also defined on $\mathbb{R}^n$, our multilinear Fourier representation enjoys more desirable theoretical properties, which we will elaborate at the end of Subsection 3.5. Moreover, optimizing a multilinear polynomial plays an important role in submodular optimization [63]. Nevertheless, in this work we do not assume submodularity and derive theoretical results independently.

1.4. *Paper organization*

Section 2 presents preliminaries about hybrid Boolean formula and Walsh-Fourier Transform of Boolean functions. Section 3 proposes a reduction from Boolean satisfiability to constrained continuous minimization and lists theoretical properties of this reduction. Section 4 proposes a group of specialized algorithms for minimizing our objective function. Furthermore, properties of those algorithms are presented and proved. Section 5 demonstrates the ability of our method on solving hybrid Boolean formula and MaxSAT instances by experimental results. Finally, Section 6 summarizes the main contributions of this paper and describes several future directions.

Table 1
Notations and their descriptions.

Notation	Description
a, y, z	A point in $[-1, 1]^n$
$\{a_{(i)}\}$	A sequence of points in optimization
$a^\star$	The convergent point in optimization
a_i	The i-th coordinate of a
a_S	The projected point of a on set S
b	A Boolean assignment in $\{-1, 1\}^n$
c	A Boolean constraint
d	The difference of sp
C_f	The set of constraints of f
$\mathbb{E}$	The expectation
$\emptyset$	The empty set
f	A Boolean function
$f\|_{x_i \leftarrow 1}$	The Boolean function obtained by fixing x_i to 1 in f
$\widehat{f}(S)$	Fourier coefficient of f at set S
F	A multilinear polynomial
$F\|_{x_i \leftarrow 1}$	The multilinear polynomial obtained by fixing x_i to 1 in F
F_f	The multilinear objective function corresponding to f
$\mathtt{FE}_c$	Fourier Expansion of constraint c
G	The projected Gradient
H	A matrix
g, h	A general function
I, J, S	A set of indices, i.e., a subset of $[n]$
$\mathcal{S}_a$	The probability space induced by rounding function $\mathcal{R}$ and point a
sp	The spectrum of symmetric Boolean function
T_ρ	The noise operator on Boolean functions
n	The number of Boolean variables in a formula
m	The number of Boolean constraints of a formula
$m_{\mathtt{SAT}}$	The number of satisfied Boolean constraints of a formula
$[n]$	The set $\{1, 2, \cdots, n\}$
v	A direction in $\mathbb{R}^n$
$\mathbb{P}$	The probability
$\mathbb{R}$	The real domain
$\mathcal{R}$	The randomized rounding function
t	The iteration number
$\mathcal{U}$	Uniform distribution
w_f	The constraint weight function of formula f
x	The set of Boolean variables $\{x_1, x_2, \cdots, x_n\}$
x_S	The set of Boolean variables $\{x_i \| i \in S\}$
α, β, γ	A real number
Δ	A (convex) set
η	The step size in gradient descent
ε	The accuracy in gradient descent
∇F	First-order derivative of F
$\nabla_i F$	First-order derivative of F at x_i, a.k.a., $\frac{\partial F}{\partial x_i}$ or $(\nabla F)_i$
$\nabla_v F$	Directional derivative of F in direction v
$\nabla^2 F$	Second-order derivative of F
$\nabla_v^2 F$	Second directional derivative of F in direction v
$\#$	The number of
$\#_{neg}(b)$	The number of negative elements in vector b
$\wedge$	The logical AND
$\vee$	The logical OR
$\oplus$	The logical XOR

1.5. Notations

For the readers' convenience, we list all notations used in this paper and their descriptions in Table 1, though those notations will still be introduced at their first appearance.

2. Boolean formulas and Walsh-Hadamard-Fourier transform

In this section we introduce basic concepts, notations and background. We also give a historical note for Walsh-Hadamard-Fourier Transform.

Table 2
Examples of Fourier Expansion on different constraints.

Constraint (f)	Fourier Expansion ($\texttt{FE}_f$)
$x_1 \vee x_2$	$-\frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_1x_2$
$\texttt{CARD}^{\geq 2}(x_1, x_2, x_3)$	$\frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_3 - \frac{1}{2}x_1x_2x_3$
$x_1 \oplus x_2 \oplus x_3$	$x_1x_2x_3$
$\texttt{NAE}(x_1, x_2, x_3)$	$-\frac{1}{2} + \frac{1}{2}x_1x_2 + \frac{1}{2}x_2x_3 + \frac{1}{2}x_1x_3$

2.1. Boolean formulas and constraints

Let $x = (x_1, ..., x_n)$ be a sequence of n Boolean variables. A Boolean function $f(x)$ is a mapping from a Boolean vector $\{\texttt{True}, \texttt{False}\}^n$ to $\{\texttt{True}, \texttt{False}\}$. A vector $b \in \{\texttt{True}, \texttt{False}\}^n$ is called an assignment and $f(b)$ denotes the value of f on b. A literal is either a variable x_i or its negation $\neg x_i$. In this paper, we assume $f = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ is the conjunction of m constraints. In the following we list four useful types of constraints:

- `Disjunctive Clauses (their conjunction forms a CNF formula):` A disjunctive clause is a disjunction of elements in a literal set, which is satisfied when at least one literal is true, e.g., $(x_1 \vee x_2 \vee \neg x_3)$.
- `CARD:` Given a set L of literals and an integer $k \in [n]$, a cardinality constraint $\texttt{CARD}^{\geq k}(L)$ (resp. $\texttt{CARD}^{\leq k}(L)$) requires the number of literals assigned `True` to be at least (resp. most) k.
- `XOR:` An XOR constraint indicates the parity of the number of literals assigned `True`, which can be computed by addition in $\mathbb{F}_2$, e.g., $x_1 \oplus x_2 \oplus x_3$.
- `NAE:` A Not-All-Equal (`NAE`) constraint is satisfied when not all the variables have the same value.[2] E.g., `NAE(True, True, True, True, False) = True`; `NAE(False, False, False, False, False) = False`.

Let the set of constraints of f be C_f, $m = |C_f|$ and n be the number of variables of f. A solution of f is an assignment that satisfies all the constraints in C_f. We aim to design a framework to find a solution of f.

2.2. Fourier expansion of a Boolean function

From now on, we define a Boolean function by $f : \{\pm 1\}^n \to \{\pm 1\}$. One point which might be counter-intuitive is that -1 is used to stand for `True` and $+1$ for `False`. An assignment now is a vector $b \in \{\pm 1\}^n$.

Walsh(-Hadamard-Fourier) Transform is a method for transforming a Boolean function into a multilinear polynomial. The polynomial after transformation is often referred as the Fourier Expansion. The following theorem shows that any function defined on a Boolean hyper-cube has an equivalent Fourier Expansion.

Theorem 1. (Walsh (-Hadamard-Fourier) Transform) *Given a function $f : \{\pm 1\}^n \to \mathbb{R}$, there is a unique way of expressing f as a multilinear polynomial, denoted by FE_f, which agrees with f on all points in $\{\pm 1\}^n$. The polynomial has at most 2^n terms and each term corresponds to one subset of $[n]$, according to:*

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

where each $\widehat{f}(S) \in \mathbb{R}$ is called the Fourier coefficient of f at S, and is computed as:

$$\widehat{f}(S) = \mathop{\mathbb{E}}_{b \sim \mathcal{U}[\{\pm 1\}^n]} \left[f(b) \cdot \prod_{i \in S} b_i \right] = \frac{1}{2^n} \sum_{b \in \{\pm 1\}^n} \left(f(b) \cdot \prod_{i \in S} b_i \right)$$

*where $b \sim \mathcal{U}[\{\pm 1\}^n]$ indicates b is uniformly sampled from $\{\pm 1\}^n$. The polynomial $\texttt{FE}_f$ is referred as the **Fourier Expansion** of f.*

Corollary 1. *For a Boolean function $f : \{\pm 1\}^n \to \mathbb{R}$, the constant term of the polynomial $\texttt{FE}_f$ equals to the expectation of f under uniform distribution, i.e., $\widehat{f}(\emptyset) = \mathop{\mathbb{E}}_{b \sim \mathcal{U}[\{\pm 1\}^n]} [f(b)]$.*

Table 2 shows some examples of Fourier Expansions.

[2] Note that an `NAE` constraint can be represented by two disjunctive clauses.

2.3. Walsh(-Hadamard-Fourier) transform: history and applications

The history of Fourier Expansion dates back to 1923 when Walsh introduced a complete orthonormal basis for ± 1 functions [64]. The values of Walsh function form a symmetric, involutive matrix $H \in \{-1,1\}^{2^n \times 2^n}$, known as the Hadamard Matrix [65]. The list of Fourier coefficients $\mathbf{s} \in \mathbb{R}^{2^n}$ can be computed by $\mathbf{s} = \frac{1}{2^n} H \cdot \mathbf{v}$ from the function value list $\mathbf{v} \in \{-1,1\}^{2^n}$. This linear transform is often called Walsh Transform or Walsh-Hadamard Transform and sometimes Fourier transform on Boolean functions.

The study of Walsh Transform splits into two directions: as a mathematical tool in theoretical computer science and as a practical transformation of vectors.

Towards theoretical direction, Walsh Transform provides a way of applying harmonic analysis on Boolean functions. In related literature, the transform is frequently referred as the "Fourier Transform". The use of "Fourier–Walsh analysis" in the study of Boolean functions quickly became well known in the early 1960s. In 1970, Bonami obtained the first hypercontractivity result [66], which later became a crucial tool for analyzing Boolean functions. However, the use of Boolean analysis in theoretical computer science seemed to wane until 1988, when the outstanding work of Kahn, Kalai, and Linial [67] ushered in a new area of sophistication [42]. The Walsh Transform on Boolean functions has over the past two decades become one of the most important and versatile tools for solving problems in theoretical computer science, such as proving circuit complexity [68], the threshold of phase-transition of graph property [69] and the impossibility of designing a social welfare function that satisfies certain conditions (Arrow's Theorem) [70]. Besides theory, there also exists some algorithmic work on applying Walsh-Fourier transform to solve more practical problems, such as finding new hashing functions for model counting [71] and approximating the partition function [46].

On the practical side, the transform is more often referred as the "Walsh-Hadamard Transform" instead of "Fourier Transform". As one of the most popular spectral transformations, Walsh Transform finds applications in Boolean function classification [72], Boolean function matching [73] and circuit synthesis [74]. Since one of the shortcomings of Walsh Transform-based methods is the expensive computational cost, extensive work has been done on making the transform more efficient. Walsh Transform is equivalent to multi-dimensional Discrete Fourier Transform (DFT). Therefore, the existence of Fast Walsh Transform with complexity $O(n2^n)$, is not surprising. In addition, there are works that use some compact data structures, e.g., Binary Decision Diagram (BDD) [75] as the representation of Boolean functions to be transformed. Nevertheless, the worst-case complexity of the Walsh-Fourier Transform is still exponential, even with respect to the size of the data structures for representing Boolean functions [76,77].

3. A reduction from SAT to continuous optimization

In this section, we reduce finding a solution of a Boolean formula to reaching a minimum of a continuous multilinear polynomial. The core idea is to evaluate the objective function composed of Fourier Expansions on the real domain. Then the value of the objective function essentially indicates the expectation of the number of satisfied constraints. Evaluating the Fourier Expansions is, in general, computationally prohibitive. Thus we focus on the Boolean formulas consisting of only symmetric constraints, which already covers many interesting cases and design methods to evaluate the objective function. We then explore promising theoretical properties of the multilinear representation.

3.1. Computing Fourier expansions is #P-hard in general

Our basic idea is to apply continuous optimization on Fourier Expansions of Boolean functions. However, in general, computing the Fourier coefficients of a Boolean function is nontrivial and often harder than SAT itself (see Proposition 1). Even if we are able to get the Fourier Expansion of a Boolean formula, the polynomial can have high degree—and as a result, exponentially many terms—making it hard to evaluate.

Proposition 1. *Computing the Fourier Expansion of a pseudo-Boolean constraint*[3] *is #P-hard.*

Proof. We reduce the counting of the number of solutions of a knapsack problem, a #P-hard problem, to the computation of the constant term of the Fourier Expansion of a pseudo-Boolean function.

For a knapsack problem with n items, weights of items $\{w_1, w_2, \dots, w_n\}$ and capacity $\mathtt{cap}$, we construct the pseudo-Boolean constraint c: $\sum_i w_i \cdot \frac{1-b_i}{2} \leq \mathtt{cap}$, or rewritten as $\sum_i w_i b_i \geq \sum_i w_i - 2 \cdot \mathtt{cap}$ where $b \in \{\pm 1\}^n$. Note that each $\frac{1-b_i}{2} \in \{0,1\}$ indicates whether the i-th item is selected in the knapsack as the traditional Boolean variable. It is easy to verify that there is a bijection between solutions of c and the solutions of the corresponding knapsack problem. Let the corresponding Boolean function of c be f_c ($f_c = -1$ if c is satisfied and $f_c = 1$ if c is unsatisfied). Let $\mathtt{SOL}_c = \{b | b \in \{\pm 1\}^n, f_c(b) = -1\}$ be the solution set of c. By Corollary 1, the constant term $\widehat{f_c}(\emptyset) = \mathop{\mathbb{E}}_{b \sim \mathcal{U}[\{\pm 1\}^n]}[f_c(b)] = \frac{(-1)\cdot|\mathtt{SOL}_c| + 1\cdot(2^n - |\mathtt{SOL}_c|)}{2^n}$, thus $|\mathtt{SOL}_c| =$

[3] a generalization of a cardinality constraint, e.g., $3x_1 + 2x_2 + 7(x_3) \geq 0$, $x \in \{\pm 1\}^3$.

$2^{n-1}(1-\widehat{f_c}(\emptyset))$. Therefore, there is a reduction from counting the solutions of a knapsack problem to computing Fourier coefficients (in this case, only the constant term) of a pseudo-Boolean function. □

Instead of computing Fourier coefficients of a Boolean function corresponding to a monolithic logical formula, we take advantage of factoring, constructing a polynomial from the Fourier Expansions of all constraints, as we will describe in the rest of this section.

3.2. Symmetric Boolean constraints have closed-form Fourier expansions

In this subsection we consider symmetric Boolean functions. Although the Fourier coefficients of symmetric Boolean functions on $\mathbb{F}_2$ has been studied by [78,79], to our best knowledge, we give the first explicit form of Fourier coefficients of symmetric Boolean functions defined on $\{-1,1\}^n$.

Definition 1. (Symmetric Boolean function) A Boolean function f is **symmetric** if $f(x_1,\cdots,x_n)\equiv f(x_{\pi(1)},\cdots,x_{\pi(n)})$ for every permutation π of $[n]$. Alternatively, symmetric Boolean functions can be regarded as functions whose value only depends on the number of coordinates assigned to be -1. Let the **spectrum** of a symmetric function f be $sp\in\{\pm1\}^{n+1}$, where for $0\le i\le n$, sp_i is equal to the value of f where i variables are set to -1 and the other $(n-i)$ variables are set to 1.

Example 1. The spectrum of NAE is $sp_{\texttt{NAE}}=(1,-1,\cdots,-1,1)$. The spectrum of OR is $sp_{\texttt{OR}}=(1,-1,\cdots,-1)$.

In fact, a symmetric Boolean function is uniquely determined by the value of its spectrum. In the rest of this subsection we derive the closed-form expressions of Fourier Expansions for symmetric functions in terms of their spectra.

Theorem 2. *Let f be a symmetric Boolean function and sp be its spectrum. Then for every $S\in[n]$,*

$$\widehat{f}(S)=\begin{cases}\frac{1}{2^n}\cdot\sum_{i=0}^{n}\binom{n}{i}\cdot sp_i, & |S|=0\\ \frac{g(\rho)_{[\rho^{|S|-1}]}}{\binom{n-1}{|S|-1}2^{n-1}}, & |S|\neq 0\end{cases}$$

where $g(\rho)_{[\rho^{|S|-1}]}$ is the coefficient of $\rho^{|S|-1}$ in the expansion of polynomial $g(\rho)$ and $g(\rho)=\sum_{i=0}^{n-1}d_i\cdot\binom{n-1}{i}(1+\rho)^{n-i-1}(1-\rho)^i$. $d\in\{-1,0,1\}^n$ is defined as $d_i=\frac{1}{2}(sp_i-sp_{i+1})$ for each $0\le i\le n-1$.

Proof. This proof is a generalization of the proof of Theorem 5.19 in [42]. Skipping this technical proof will not undermine understanding the rest of the paper.

For the case $|S|=0$. By Corollary 1, the constant term $\widehat{f}(\emptyset)$ equals to the expectation of the function:

$$\begin{aligned}\widehat{f}(\emptyset)&=\mathop{\mathbb{E}}_{b\sim\mathcal{U}[\{\pm1\}^n]}[f(b)]\\ &=\tfrac{1}{2^n}\left(\mathop{\#}_{b\in\{\pm1\}^n}f(b)=-1\right)\cdot(-1)+\tfrac{1}{2^n}\left(\mathop{\#}_{b\in\{\pm1\}^n}f(b)=1\right)\cdot 1\\ &=\frac{\sum_{i=0}^{n}\binom{n}{i}\cdot sp_i}{2^n}\end{aligned}$$

For the case of $|S|\neq 0$, we need to use more advanced arguments.

For a Boolean point $b\in\{-1,1\}^n$ and $\rho\in[0,1]$, we use $y\sim N_\rho(b)$ to denote the following distorted probability distribution of $y\in\{-1,1\}^n$: for each $i\in[n]$, y_i takes value b_i with prob. $(\frac{1}{2}+\frac{\rho}{2})$ and $-b_i$ with prob. $(\frac{1}{2}-\frac{\rho}{2})$. Based on this distribution, for a Boolean function f and a Boolean point b, let $T_\rho(f,b)=\mathop{\mathbb{E}}_{y\sim N_\rho(b)}[f(y)]$ and T_ρ is often called the **noise operator** with respect to ρ.

There are different ways of computing T_ρ, which leads to the derivation of the Fourier coefficients of symmetric functions.

Method 1 of computing T_ρ.

The derivative of f at variable x_n, denoted by $\nabla_n f:\{-1,1\}^{n-1}\to\{-1,0,1\}$, can be computed by

$$\nabla_n f(x)=\frac{1}{2}(f(x_1,x_2,\cdots,x_n=1)-f(x_1,x_2,\cdots,x_n=-1)).$$

For a vector $b \in \{-1,1\}^n$, let the number of -1s in b be denoted by $\#_{neg}(b)$. Given that f is symmetric, it can be verified that

$$\nabla_n f(b) = d_{\#_{neg}(b_{[n-1]})},$$

i.e., $\nabla_n f(b)$ equals the element of d with index $\#_{neg}(b_{[n-1]})$.

By the probabilistic definition of T_ρ (See Definition 2.46 in [42] for more details), we have

$$\begin{aligned}&T_\rho(\nabla_n f, (1,1,...,1))\\ &= \mathop{\mathbb{E}}_{y\sim N_\rho(1,1,...,1)}[\nabla_n f(y)]\\ &= \mathop{\mathbb{E}}_{y\sim N_\rho(1,1,...,1)}[d_{\#_{neg}(y_{[n-1]})|}]\\ &= \sum_{i=0}^{n-1} d_i \cdot \mathop{\mathbb{P}}_{y\sim N_\rho(1,1,...,1)}[d_{\#_{neg}(y_{[n-1]})} = i]\\ &= \sum_{i=0}^{n-1} d_i \cdot \binom{n-1}{i}(\frac{1}{2}+\frac{\rho}{2})^{n-i-1}(\frac{1}{2}-\frac{\rho}{2})^i.\end{aligned}$$

Method 2 of computing T_ρ.

Consider $\nabla_n f(x)$, the derivative of f at x_n. Since f is symmetric, we have

$$\widehat{f}(S) = \widehat{\nabla_n f}(T)$$

for any $T \subseteq [n-1]$ with $|T| = |S|-1$. This can be seen by observing that a term in $\nabla_n f$ with $|T|$ variables comes from a term with $|T|+1$ variables in f before differentiating on x_n.

By the Fourier Expansion representation of T_ρ (see Proposition 2.47 in [42]) and the fact that $\nabla_n f$ is symmetric, we have

$$T_\rho(\nabla_n f, (1,1,...,1)) = \sum_{U\subseteq[n-1]} \widehat{\nabla_n f}(U)\rho^{|U|} = \sum_{i=0}^{n-1}\binom{n-1}{i}\widehat{\nabla_n f}([i])\rho^i$$

The equation for computing Fourier coefficients can be obtained by equating coefficients of ρ in two representations of $T_\rho(\nabla_n f)(1,1,...,1)$ computed by Method 1 and Method 2. □

Remark 1. For symmetric functions, $\widehat{f}(S)$ can also be written more succinctly by Kravchuk polynomials $\mathcal{K}_k(i;n,q) = \sum_{j=0}^{k}(-1)^j(q-1)^{k-j}\binom{i}{j}\binom{n-i}{k-j}$.

$$\widehat{f}(S) = \begin{cases}\frac{1}{2^n}\cdot\sum\limits_{i=0}^{n}\binom{n}{i}\cdot sp_i, & |S|=0\\ \frac{1}{\binom{n-1}{|S|-1}2^{n-1}}\cdot\sum\limits_{i=0}^{n-1} d_i\cdot\mathcal{K}_{|S|}(i,n-1,2), & |S|\neq 0\end{cases}$$

Above we show how to obtain the closed-form expression of Fourier coefficients of symmetric functions with all positive literals. In the following we will see that allowing negative literals does no harm. The key observations is that the Fourier Expansion of $\neg f$ is simply $-\mathtt{FE}_f$. Let l_i be a literal (x_i or $\neg x_i$). Let $\mathtt{sgn}(l_i)=1$ if $l_i = x_i$ and $\mathtt{sgn}(l_i) = -1$ if $l_i = \neg x_i$. For a Boolean constraint c, suppose every variable appears at most once in c. Then c can be written as a functions of literals, i.e., $c(l_1, l_2, \cdots, l_n)$. Let f_p be the "positive version" of f, obtained by flipping all negative literals in f to positive. If we know $\widehat{f_p}(S)$, i.e., the Fourier coefficient of $\widehat{f_p}$ on S, then it is not hard to verify that $\widehat{f}(S) = \widehat{f_p}(S)\cdot\prod\limits_{i\in S}\mathtt{sgn}(l_i)$.

Example 2. (Negative literals) The Fourier Expansion of $(x_1 \vee x_2 \vee x_3)$ is $-\frac{3}{4}+\frac{1}{4}(x_1+x_2+x_3+x_1x_2+x_2x_3+x_1x_3+x_1x_2x_3)$. Thus the Fourier Expansion of $(x_1 \vee \neg x_2 \vee \neg x_3)$ is $-\frac{3}{4}+\frac{1}{4}(x_1+(-1)\cdot x_2+(-1)\cdot x_3+x_1\cdot(-1)\cdot x_2+(-1)\cdot x_2\cdot(-1)\cdot x_3+x_1\cdot(-1)\cdot x_3+x_1\cdot(-1)\cdot x_2\cdot(-1)\cdot x_3)$.

In Remark 2, we give closed-form expressions of Fourier coefficients of some types of constraints (in "positive" version) which are special cases of symmetric Boolean functions.

Remark 2. We give the closed-form representation of Fourier coefficients of the following classes of constraints. Without loss of generality, we assume all literals are positive. In order to demonstrate how Theorem 2 actually works, we show the detailed steps for deriving the Fourier coefficients of OR constraints.

- Cardinality constraints ($\texttt{CARD}^{\geq k}$)/disjunctive clauses (OR).

$$\widehat{\texttt{CARD}^{\geq k}}(S) = \begin{cases} 1 - \frac{\sum_{i=k}^{n}\binom{n}{i}}{2^{n-1}}, |S| = 0 \\ \frac{\binom{n-1}{k-1}((1+\rho)^{n-k}(1-\rho)^{k-1})_{[\rho^{|S|-1}]}}{\binom{n-1}{|S|-1}2^{n-1}}, |S| \neq 0 \end{cases}$$

In particular, a disjunctive clause is $\texttt{CARD}^{\geq 1}$, a.k.a., OR.

$$\widehat{\texttt{OR}}(S) = \begin{cases} \frac{1}{2^{n-1}} - 1, |S| = 0 \\ \frac{1}{2^{n-1}}, |S| \neq 0 \end{cases}$$

Proof. The spectrum of OR is $sp_{\texttt{OR}} = (1, -1, \cdots, -1)$.
By Theorem 2,

$$\begin{aligned}\widehat{\texttt{OR}}(\emptyset) =& \frac{1}{2^n} \cdot \sum_{i=0}^{n} \binom{n}{i} \cdot (sp_{\texttt{OR}})_i \\ =& \frac{1}{2^n}(1 - \sum_{i=1}^{n}\binom{n}{i}) \\ =& \frac{1}{2^{n-1}} - 1\end{aligned}$$

Next we assume $S \neq \emptyset$. By the definition in Theorem 2,

$$d_{\texttt{OR}} = \frac{1}{2} \cdot (1-(-1), (-1)-(-1), \cdots, (-1)-(-1)) = (1, 0, 0, \cdots, 0)$$

According to Theorem 2,

$$\begin{aligned}g(\rho) &= \sum_{i=0}^{n-1}(d_{\texttt{OR}})_i \cdot \binom{n-1}{i}(1+\rho)^{n-i-1}(1-\rho)^i \\ &= \binom{n-1}{0}(1+\rho)^{n-0-1}(1-\rho)^0 = (1+\rho)^{n-1}\end{aligned}$$

Therefore by Theorem 2, for $S \neq \emptyset$ we have

$$\widehat{\texttt{OR}}(S) = \frac{g(\rho)_{[\rho^{|S|-1}]}}{\binom{n-1}{|S|-1}2^{n-1}} = \frac{\binom{n-1}{|S|-1}}{\binom{n-1}{|S|-1}2^{n-1}} = \frac{1}{2^{n-1}}. \quad \square$$

- Not-All-Equal (NAE).

$$\widehat{\texttt{NAE}}(S) = \begin{cases} 0, |S| \text{ is odd} \\ (\frac{1}{2})^{n-2} - 1, |S| = 0 \\ (\frac{1}{2})^{n-2}, |S| \text{ is even and non-zero} \end{cases}$$

- Exact-One (EO).

$$\widehat{\texttt{EO}}(S) = \begin{cases} 1 - \frac{n}{2^{n-1}}, |S| = 0 \\ \frac{\binom{n-1}{|S|-1} - (n-1)\left[\binom{n-2}{|S|-1} - \binom{n-2}{|S|-2}\right]}{\binom{n-1}{|S|-1}2^{n-1}}, |S| \neq 0 \end{cases}$$

- XOR on variable set X.

$$\widehat{\texttt{XOR}}(S) = \begin{cases} 1, S = X \\ 0, S \neq X \end{cases}$$

Example 3. In this example we will compute all Fourier coefficients of the cardinality constraint $c = \texttt{CARD}^{\geq 2}(x_1, x_2, x_3)$ in Table 2 by Remark 2. Note that c can be regarded as an odd function, i.e., $c(x) = -c(-x)$. Thus all coefficients with even degree will be zero.

$$\widehat{c}(\emptyset) = 1 - \frac{\sum_{i=2}^{3}\binom{3}{i}}{2^{3-1}} = 1 - \frac{3+1}{4} = 0$$

$$\widehat{c}(\{x_1\}) = \widehat{c}(\{x_2\}) = \widehat{c}(\{x_3\}) = \frac{\binom{3-1}{2-1}\left((1+\rho)^{3-2}(1-\rho)^{2-1}\right)_{[\rho^{1-1}]}}{\binom{3-1}{1-1}2^{3-1}} = \frac{2}{4} = \frac{1}{2}$$

$$\widehat{c}(\{x_1, x_2\}) = \widehat{c}(\{x_2, x_3\}) = \widehat{c}(\{x_1, x_3\}) = \frac{\binom{3-1}{2-1}\left((1+\rho)^{3-2}(1-\rho)^{2-1}\right)_{[\rho^{2-1}]}}{\binom{3-1}{2-1}2^{3-1}} = \frac{2\cdot\left(1-\rho^2\right)_{[\rho]}}{2\cdot 2^2} = 0$$

$$\widehat{c}(\{x_1, x_2, x_3\}) = \frac{\binom{3-1}{2-1}\left((1+\rho)^{3-2}(1-\rho)^{2-1}\right)_{[\rho^{3-1}]}}{\binom{3-1}{3-1}2^{3-1}} = \frac{2\cdot\left(1-\rho^2\right)_{[\rho^2]}}{1\cdot 2^2} = -\frac{1}{2}$$

Thus $\texttt{FE}_c = \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_3 - \frac{1}{2}x_1x_2x_3$.

3.3. A reduction from SAT to optimization

Definition 2. (Objective function) For a formula f with constraint set C_f, we define the *objective function* associated with f, denoted by F_f, by the sum of Fourier Expansions of f's constraints, i.e.,

$$F_f = \sum_{c \in C_f} \texttt{FE}_c$$

where $\texttt{FE}_c$ is the Fourier Expansion of constraint c.

The degree (maximum number of variables in all terms) of F_f equals to the maximum number of literals among all constraints. Note that, in general, F_f is not the Fourier Expansion of f. Instead, it can be regarded as a surrogate of f's Fourier Expansion which is relatively easy to compute.

Let us provide an example to demonstrate the above ideas.

Example 4. Suppose $f = (x_1 \vee x_2) \wedge (x_3 \oplus x_5) \wedge (x_2 \vee \neg x_4)$. Then, $C_f = \{x_1 \vee x_2,\ x_3 \oplus x_5,\ x_2 \vee \neg x_4\}$ and

$$\begin{aligned} F_f &= \left(-\frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_1x_2\right) + x_3x_5 + \left(-\frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_4 - \frac{1}{2}x_2x_4\right) \\ &= -1 + \tfrac{1}{2}x_1 + x_2 + x_3x_5 - \tfrac{1}{2}x_4 + \tfrac{1}{2}x_1x_2 - \tfrac{1}{2}x_2x_4. \end{aligned}$$

The objective function is a polynomial, and nothing prevents us from relaxing the domain from discrete to continuous. An assignment is now defined as a real vector in $[-1, 1]^n$. The reduction is formalized in Theorem 3. ·

Theorem 3. (Reduction) *f is satisfiable if and only if*

$$\min_{a \in [-1,1]^n} F_f(a) = -m.$$

Recall that m is the number of constraints. Theorem 3 reduces SAT to a multivariate minimization problem over $[-1, 1]^n$.

Example 5. Suppose $f = (x_1 \vee \neg x_2) \wedge (\neg x_1 \oplus x_2)$. Then

$$\begin{aligned} F_f &= \left(-\frac{1}{2} + \frac{1}{2}x_1 - \frac{1}{2}x_2 - \frac{1}{2}x_1x_2\right) - x_1x_2 \\ &= -\frac{1}{2} + \frac{1}{2}x_1 - \frac{1}{2}x_2 - \frac{3}{2}x_1x_2. \end{aligned}$$

f has two solutions: $x_1 = \texttt{True}$, $x_2 = \texttt{True}$ and $x_1 = \texttt{False}$, $x_2 = \texttt{False}$, which correspond to two global minima $(-1, -1)$ and $(1, 1)$ respectively of F_f in $[-1, 1]^n$ with function value -2. The plots for F_f and its gradient are shown in Fig. 2.

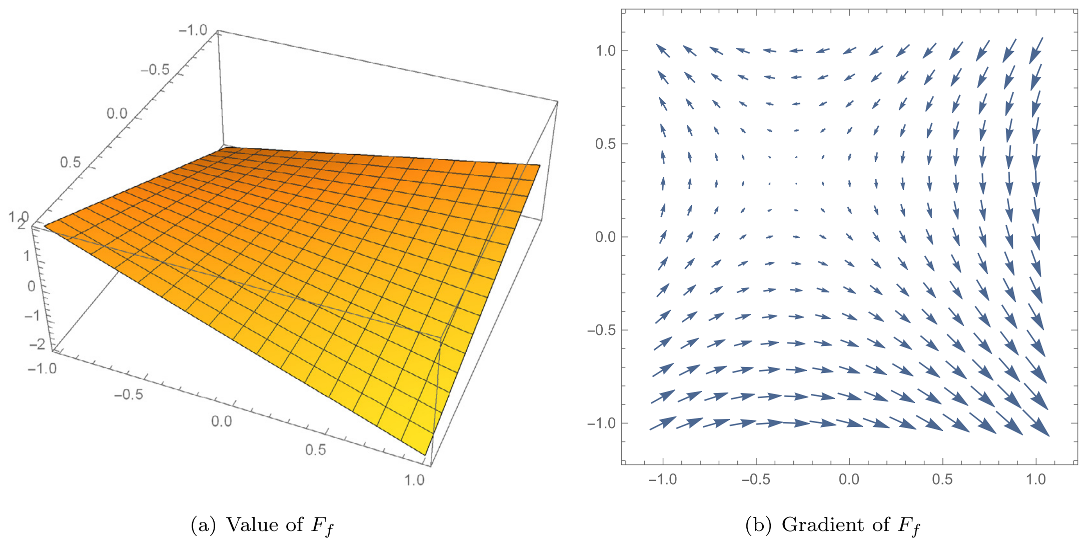


Fig. 2. Value and Gradient of $F_f = -\frac{1}{2} + \frac{1}{2}x_1 - \frac{1}{2}x_2 - \frac{3}{2}x_1x_2$ with $f = (x_1 \vee \neg x_2) \wedge (\neg x_1 \oplus x_2)$, the case in Example 5. Two global minima of F_f are $x_1 = x_2 = -1$ and $x_1 = x_2 = 1$, corresponding to two solutions of f.

In the rest of this subsection, we will prove Theorem 3.

Definition 3. (Constant) A constraint is constant if it is equivalent to either `True` or `False`. The Fourier Expansion of a constraint is constant if it always equals to -1 or 1.

Definition 4. (Partially assigned function) Let J be a subset of $\{1, \cdots, n\}$, $a \in [-1, 1]^{|J|}$ be a real vector and $F : [-1, 1]^n \to \mathbb{R}$, we use $F|_{x_J \leftarrow a}$ to denote the partially assigned function of F given by fixing the value of variables with indices in J to be the values in a.

Example 6. Let $F = \frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_1x_2$, $J = \{1\}$ and $a = (\frac{1}{2})$, then $F|_{x_J \leftarrow a} = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2} \cdot \frac{1}{2} \cdot x_2 = \frac{3}{4} + \frac{1}{4}x_2$.

Definition 5. (Projection of vectors) For a real vector $a \in [-1, 1]^n$ and $S \subseteq [n]$, we use a_S to denote the projected vector of a at coordinates in S. E.g., let $a = (0.1, 0.2, 0.3)$ and $S = \{1, 3\}$, then $a_S = (0.1, 0.3)$.

The following fact can be regarded as the Fourier Expansion version of Boole's Expansion Theorem. We will use this decomposition of multilinear polynomials several times in the proofs.

Fact 1. Let F be a multilinear polynomial. For all $a \in [-1, 1]^n$, we have

$$F(a) = \tfrac{1-a_n}{2} \cdot F|_{x_n \leftarrow (-1)}(a_{[n-1]}) + \tfrac{1+a_n}{2} \cdot F|_{x_n \leftarrow 1}(a_{[n-1]}).$$

Note that both $F|_{x_n \leftarrow 1}$ and $F|_{x_n \leftarrow (-1)}$ are independent with x_n, the variable we split on.

Lemma 1 indicates the value of multilinear polynomials is well-behaved in the cube $[-1, 1]^n$.

Lemma 1. *Let c be a non-constant constraint and a be an assignment. Then:*

1. *if $a_i \in \{-1, 1\}$ for all $i \in [n]$, then $\mathtt{FE}_c(a) \in \{-1, 1\}$.*
2. *if $a_i \in [-1, 1]$ for all $i \in [n]$, then $\mathtt{FE}_c(a) \in [-1, 1]$.*
3. *if $a_i \in (-1, 1)$ for all $i \in [n]$, then $\mathtt{FE}_c(a) \in (-1, 1)$.*

Proof Sketch. The first argument is a direct result from the definition of the Fourier Expansion of Boolean functions. The proofs of 2. and 3. are based on induction on the number of variables and the decomposition technique in Fact 1.

Proof. `Basis step`: Let $n = 1$. Since $\mathtt{FE}_c$ is non-constant and $\mathtt{FE}_c(1)$, $\mathtt{FE}_c(-1) \in \{\pm 1\}$, $\mathtt{FE}_c(x)$ is either x_1 or $-x_1$. Then, the two last arguments of the lemma hold trivially.

`Inductive step`: Suppose $n \geq 2$. Then, $\mathtt{FE}_c(a)$ can be expanded as:

$$\mathtt{FE}_c(a) = \tfrac{1-a_n}{2} \cdot \mathtt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]}) + \tfrac{1+a_n}{2} \cdot \mathtt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]})$$

i) Suppose $a_i \in [-1,1]$ for $\forall i \in [n]$.

- If $\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]}) \cdot \texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]}) < 0$. Then,

$$|\texttt{FE}_c(a)| \leq \max\left\{ \tfrac{1-a_n}{2} \cdot \left|\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]})\right|, \tfrac{1+a_n}{2} \cdot \left|\texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]})\right| \right\} \leq 1$$

- If $\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]}) \cdot \texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]}) \geq 0$, then we have

$$|\texttt{FE}_c(a)| \leq \left(\tfrac{1-a_n}{2} + \tfrac{1+a_n}{2}\right) \cdot \max\left\{ \left|\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]})\right|, \left|\texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]})\right| \right\} \leq 1$$

Note that the last steps in the two cases follow by inductive hypothesis.

ii) Suppose $a_i \in (-1,1)$ for $\forall i \in [n]$.

- If $\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]}) \cdot \texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]}) < 0$, then, since $|\frac{1+a_n}{2}| < 1, |\frac{1-a_n}{2}| < 1$, we have by inductive hypothesis that,

$$|\texttt{FE}_c(a)| \leq \max\left\{ \tfrac{1-a_n}{2} \cdot \left|\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]})\right|, \tfrac{1-a_n}{2} \cdot \left|\texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]})\right| \right\} < 1.$$

- If $\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]}) \cdot \texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]}) \geq 0$, since we already have proved that $|\texttt{FE}_c(a)| \leq 1$, what we need to show is $|\texttt{FE}_c(a)| \neq 1$. Note that for $|\texttt{FE}_c(a)| = 1$ to be true, both $|\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]})| = 1$ and $|\texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]})| = 1$ must hold. By inductive hypothesis, if one of $\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]})$ and $\texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]})$ is non-constant, then

$$|\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]})| < 1 \text{ or } |\texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]})| < 1.$$

Therefore both $\texttt{FE}_c|_{x_n \leftarrow (-1)}(a_{[n-1]})$ and $\texttt{FE}_c|_{x_n \leftarrow 1}(a_{[n-1]})$ are constant and $\texttt{FE}_c$ only relies on one variable x_n, which is covered by the basis. □

Most relaxing-based methods rely on specific rounding procedures to convert a real assignment to discrete. In this paper, we will show that our continuous method converges to a "discrete-like" point spontaneously. In order to prove this property, we first define the concept "feasibility".

Definition 6. (Feasible assignments) For the objective function F_f of a Boolean formula f and an assignment $a \in [-1,1]^n$, let $I = \{i \mid a_i \in \{\pm 1\}\}$ be the index set of the discrete part of a. a is a **feasible assignment** of F_f if $F_f|_{x_I \leftarrow a_I}$ is constant, where a_I is the projected vector of a at coordinates in I, defined in Definition 5. We say a is **feasible** if F_f is clear in context.

Example 7. Let $f = x_1 \wedge x_2$; then

$$F_f = \tfrac{1}{2} + \tfrac{1}{2}x_1 + \tfrac{1}{2}x_2 - \tfrac{1}{2}x_1 x_2.$$

$a = (1, -0.3)$ is a feasible assignment because $I = \{1\}$ and $F_f|_{x_I \leftarrow a_I} = F|_{x_1 \leftarrow 1} = \frac{1}{2} + \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_2 = 1$ is constant not depending on x_2. The intuition is, when we set x_1 to be `False`, then $x_1 \wedge x_2$ is always `False` despite of the value of x_2.

By Definition 6, if we find a feasible assignment a of the objective function, we can adjust it into a Boolean assignment b by rounding values of a in $(-1,1)$ to $\{\pm 1\}$ arbitrarily. By the feasibility of a, we have $F_f(a) = F_f(b)$.

Lemma 1 indicates that for a constraint c, the value of $\texttt{FE}_c(a) \geq -1$ for all $a \in [-1,1]^n$. Suppose we find a fractional point a such that $\texttt{FE}_c(a) = -1$. Can we extract a Boolean solution of c from a? Lemma 2 gives us the affirmative answer.

Lemma 2. *Let c be a non-constant constraint and $a \in [-1,1]^n$ be an assignment. If $\texttt{FE}_c(a) = -1$, then a is a feasible assignment of $\texttt{FE}_c$.*

Proof. Suppose $\texttt{FE}_c(a) = -1$. We partition $[n]$ into two sets, I and $[n] - I$, where $I = \{i \mid a_i \in \{\pm 1\}\}$ and $[n] - I = \{i \mid a_i \in (-1,1)\}$.

Consider the polynomial $\texttt{FE}_c|_{x_I \leftarrow a_I}$.

- If $\texttt{FE}_c|_{x_I \leftarrow a_I}$ is constant, then a is feasible by definition.
- Otherwise, $\texttt{FE}_c|_{x_I \leftarrow a_I}$ is non-constant. Thus $[n] - I \neq \emptyset$. Since every variable with index in I is fixed in $\{\pm 1\}$, $\texttt{FE}_c|_{x_I \leftarrow a_I}$ is the Fourier Expansion of a Boolean function on variables with indices in $[n] - I$. Since $a_i \in (-1,1)$ for every $i \in [n] - I$, by the third argument of Lemma 1 we have

$$\texttt{FE}_c(a) = \texttt{FE}_c|_{x_I \leftarrow a_I}(a_{[n]-I}) \in (-1,1),$$

which conflicts with the fact that $\texttt{FE}_c(a) = -1$. Thus $[n] - I = \emptyset$ and a is feasible by definition. □

Now we are ready to prove Theorem 3, which gives the reduction from SAT to continuous optimization.

Proof Sketch. The statement of Theorem 3 might look straightforward and natural, given that there is a one-to-one correspondence between models of f and $b \in \{\pm 1\}^n$ such that $F_f(b) = -m$. However, since the statement includes minimization in the real domain, there are still several points that need to be taken care of. For the "$\Rightarrow$" direction, when f is satisfiable, we need to guarantee that there is no inner point of the box $[-1, 1]^n$ that has an objective value even smaller than $-m$ (Lemma 1). For the "$\Leftarrow$" direction, if there is a real point $a \in [-1, 1]^n$ such that $F_f(a) = -m$, then we must guarantee that a binary assignment can be generated form a with the same objective value.

Proof of Theorem 3. Note that by the second argument in Lemma 1, we have $F_f(a) \geq -m$ for all $a \in [-1, 1]^n$.

- "$\Rightarrow$": Suppose f is satisfiable and $b \in \{\pm 1\}^n$ is one of its solutions. Then b is also a solution of every constraint of f. Thus for every $c \in C_f$, $\mathtt{FE}_c(b) = -1$. Therefore $F_f(b) = -m$ and $\min\limits_{a \in [-1,1]^n} F_f(a) = -m$.
- "$\Leftarrow$": Suppose $\min\limits_{a \in [-1,1]^n} F_f(a) = -m$. Thus there exists $a \in [-1, 1]^n$ such that $F_f(a) = -m$. By the second argument in Lemma 1, we have $\mathtt{FE}_c(a) = -1$ for every $c \in C_f$. By Lemma 2, a is a feasible assignment of $\mathtt{FE}_c$ for every $c \in C_f$. Thus a is also a feasible assignment of F_f. Therefore, by feasibility, we can get a Boolean assignment which satisfies all the constraints by arbitrarily rounding all the fractional coordinates of a. $\square$

3.4. *Where will we converge to? Local landscape of multilinear polynomials*

So far we have proved that we can obtain a solution to the original SAT problem if we find the global optimum of the objective function F_f in $[-1, 1]^n$. However, our method has not provided any guarantees regarding the solution quality of local minima, which are more often the destinations of convergence. In this subsection we characterize the local landscape of multilinear polynomials. Our main result is that local minima are meaningful and useful.

First, we formally define local minima in constrained problems.

Definition 7. (l^2-norm) For a vector $a \in \mathbb{R}^n$, the l^2-norm of a, denoted by $\|a\|$, is defined by $\|a\| = \sqrt{\sum_{i=1}^n a_i^2}$.

Definition 8. (Local minimum for constrained problem) For a vector $a \in \mathbb{R}^n$, we define $\mathcal{N}_\delta(a)$, the neighborhood of a with radius δ as $\mathcal{N}_\delta(a) = \{y \mid \|a - y\|^2 \leq \delta\}$. Given a set Δ, we say an assignment a is a **local minimum of a polynomial** F **in** Δ if

$$\exists \delta > 0,\ \forall a' \in \mathcal{N}_\delta(a) \cap \Delta,\ F(a) \leq F(a').$$

A local maximum is defined similarly. A point is called a local optimum if it is a local minimum or a local maximum.

Definition 9. (Critical points and saddle points) A critical point of a function F is a point at which all the first derivatives are zero. A saddle point is a critical point which is **not** a local optimum.

Fig. 3 illustrates some examples of saddle points.

In Theorem 4, we show that in the unconstrained setting, multilinear polynomials do not have local optima and all critical points are saddle points.

Theorem 4. *In the unconstrained setting, every critical point of a non-constant multilinear polynomial is a saddle point.*

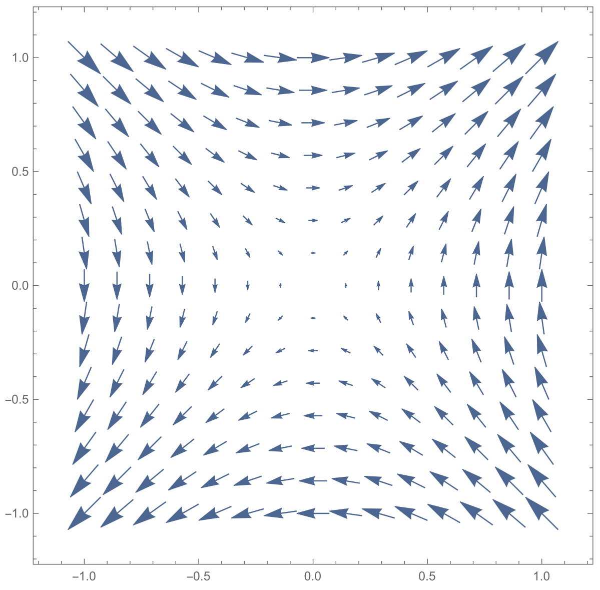
Instead of directly proving Theorem 4, we prove a stronger result in Lemma 3. Roughly speaking, for each point of a multilinear polynomial, there is always a local increasing direction and a local decreasing direction.

Remark 3. Theorem 4 can be proved directly if one leverages the fact that multilinear polynomials are harmonic functions and applies the Maximum Principle [80]. However, the proof of the Maximum Principle is non-constructive. In this article, we provide a self-contained constructive proof by proving Lemma 3, which is of algorithmic interest as shown later in Algorithm 3.

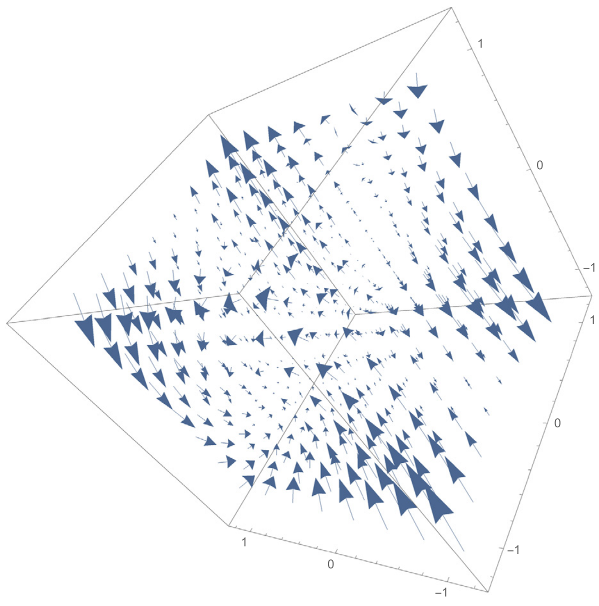
Lemma 3. *For every non-constant multilinear polynomial F and point $a \in \mathbb{R}^n$, there exist a region size $\varepsilon > 0$ and two directions $v^+, v^- \in \mathbb{R}^n$, such that for all δ s.t. $0 < \delta < \varepsilon$, the following holds:*

$$F(a + \delta v^+) > F(a) \text{ and } F(a + \delta v^-) < F(a).$$

Example 8. In the example from Fig. 2, $x_1 = -\frac{1}{3}$, $x_2 = \frac{1}{3}$ is a critical point but not a local optimum (thus a saddle point). There exist directions $v^+ = \pm(1, -1)$ towards which F_f can be increased and $v^- = \pm(1, 1)$ towards which F_f can be decreased.



(a) Gradient of parity function $x_1 \oplus x_2$



(b) Gradient of parity function $x_1 \oplus x_2 \oplus x_3$

Fig. 3. Gradient of two parity functions. $x_1 \oplus x_2$ has one saddle point: $(0, 0)$. $x_1 \oplus x_2 \oplus x_3$ has infinitely many saddle points: all points with at least two zero coordinates are saddle points. However, saddle points are not local minima, since there exist decreasing directions at each saddle point.

Proof. Though the statement considers every points in $a \in \mathbb{R}^n$, we point out that it is sufficient to only prove the case for the origin, a.k.a., $a = \mathbf{0}$. To see this, note that we can always shift the polynomial such that a is moved to $\mathbf{0}$. Indeed, shifting will not change the local geometry around the point we are considering. We can further assume that $F(\mathbf{0}) = 0$ without loss of generality since subtracting $F(a)$ does not change the geometry as well. Formally speaking, we can always consider the point $\mathbf{0}$ of the shifted polynomial, $F_{\mathtt{shift}}(x) = F(x + a) - F(a)$, instead of point a of $F(x)$.

After simplifying the scenario, we prove Lemma 3 by induction on the number of variables n of F.

`Basis step`: Assume $n = 1$. Since $F(0) = 0$ and F is non-constant, we have $F = \alpha x_1$ for some $\alpha \neq 0$. By assigning $v^+ = (\mathtt{sgn}(\alpha))$, $v^- = (-\mathtt{sgn}(\alpha))$ and ε be any positive real number, then, for all δ s.t. $0 < \delta < \varepsilon$, we have:

$$F(\mathbf{0} + \delta v^+) = |\alpha| \cdot \delta > 0 = F(\mathbf{0}),$$
$$F(\mathbf{0} + \delta v^-) = -|\alpha| \cdot \delta < 0 = F(\mathbf{0}).$$

`Inductive Step`: Since F is a multilinear function with $n \geq 2$ variables. By Fact 1, F can be decomposed as:

$$F(x_1, \ldots, x_n) = x_1 \cdot g(x_2, \ldots, x_n) + h(x_2, \ldots, x_n),$$

where both g and h are multilinear polynomials not depending on x_1. Since F is non-constant, without loss of generality, we assume $g(x_2, ..., x_n) \not\equiv 0$. Otherwise if $g(x_2, ..., x_n) \equiv 0$, then x_1 is an irrelevant variable and we can use the next variable to decompose F.

The rest of this proof is provided by case analysis. Note that $h(\mathbf{0}) = 0$ since $F(\mathbf{0}) = 0 \cdot g(\mathbf{0}) + h(\mathbf{0}) = 0$.

- If h is constant, i.e., $h \equiv 0 = h(\mathbf{0})$. Then $F = x_1 g(x_2, ..., x_n)$.
  - If $g(x_2, ..., x_n)$ is constant, say $g \equiv \alpha$. We are left with $F = \alpha x_1$, a case covered by the basis.
  - If $g(x_2, ..., x_n)$ is non-constant.
    - ☐ If $g(\mathbf{0}) = 0$, then by inductive hypothesis, there exist v_g^+, v_g^- and ε_g, such that $g(\delta v_g^+) > 0$ and $g(\delta v_g^-) < 0$ hold for every δ s.t. $0 < \delta < \varepsilon_g$. Set $v^+ = (1, v_g^+)$, $v^- = (1, v_g^-)$ and $\varepsilon = \varepsilon_g$. Now for all δ such that $0 < \delta < \varepsilon$,

$$F(\mathbf{0} + \delta v^+) = \delta \cdot g(\delta v_g^+) > 0 = F(\mathbf{0})$$
$$F(\mathbf{0} + \delta v^-) = \delta \cdot g(\delta v_g^-) < 0 = F(\mathbf{0})$$

      It is worth mentioning that we could also set $v^+ = (-1, v_g^-)$ and $v^- = (-1, v_g^+)$.
    - ☐ Else if $g(\mathbf{0}) \neq 0$, let

$$v^+ = (\mathtt{sgn}(g(\mathbf{0})), 0, \ldots, 0)$$

$$v^- = (-\mathtt{sgn}(g(\mathbf{0})),\ 0,\ \ldots,\ 0)$$

and ε be an arbitrary positive number. For all δ such that $0 < \delta < \varepsilon$, we have

$$F(\delta v^+) = \delta \cdot \mathtt{sgn}(g(\mathbf{0})) \cdot g(\mathbf{0}) > 0,$$
$$F(\delta v^-) = \delta \cdot (-\mathtt{sgn}(g(\mathbf{0}))) \cdot g(\mathbf{0}) < 0.$$

- If h is non-constant, by inductive hypothesis, there exist v_h^+, v_h^- and ε_h such that, $h(\delta v_h^+) > 0$, $h(\delta v_h^-) < 0$ hold for every $0 < \delta < \varepsilon_h$. We construct $v^+ = (0, v_h^+)$, $v^- = (0, v_h^-)$ and set $\varepsilon = \varepsilon_h$. For all δ such that $0 < \delta < \varepsilon$,

$$F(\mathbf{0} + \delta v^+) = h(\delta v_h^+) > 0 = F(\mathbf{0}),$$
$$F(\mathbf{0} + \delta v^-) = h(\delta v_h^-) < 0 = F(\mathbf{0}). \quad \square$$

Remark 4. The Hessian (the matrix of second-order derivatives) can provide intuition of why Theorem 4 holds. Note that for a multilinear polynomial F and a critical point $a \in [-1, 1]^n$, $\frac{\partial^2 F}{\partial x_i^2}(a)$ is zero for all $i \in [n]$ and a. Therefore, all elements on the diagonal of the Hessian matrix $H_F(a)$ of a multilinear polynomial F at point a is zero and $\mathtt{trace}(H_F(a)) = 0$. Since $\mathtt{trace}(H_F(a))$ equals the sum of all eigenvalues of $H_F(a)$, then either 1) there exist a positive and a negative eigenvalue of $H_F(a)$ or 2) all eigenvalues of $H_F(a)$ are zero. If 1) holds, then there exist a increasing direction and a decreasing direction, as Theorem 4 states. Otherwise, if 2) holds, then the critical point a is degenerate. Degenerate points are possible in theory (see Example 9) but rarely observed in real-life instances.

Theorem 4 indicates for unconstrained, non-constant multilinear polynomials, no local minimum exists. On the other hand, after adding the bound $[-1, 1]^n$, intuitively, a local minimum of F in $[-1, 1]^n$ can only appear on the boundary. Theorem 5 uses feasibility to formalize this intuition.

Theorem 5. *If $a \in [-1, 1]^n$ is a local minimum of a multilinear polynomial F in $[-1, 1]^n$, then a is feasible.*

Proof. Since a is a local minimum, then for every $i \in [n]$, either $a_i \in \{\pm 1\}$ or $\frac{\partial F}{\partial x_i}(a) = 0$ holds.[4] Otherwise the gradient would give us a direction to decrease the value of F. Let $I = \{i \mid a_i \in \{\pm 1\}\}$. Consider $F|_{x_I \leftarrow a_I}$, where every variable in I is assigned to its value in a. We have $a_{[n]-I}$ is a critical point of $F|_{x_I \leftarrow a_I}$ since each derivative of variable with index in $[n] - I$ is zero.

Suppose $F|_{x_I \leftarrow a_I}$ is non-constant. Since $a_{[n]-I}$ is a critical point inside the cube, by Theorem 4, $a_{[n]-I}$ is a saddle point of $F|_{x_I \leftarrow a_I}$, which means a is not a local minimum of F. Thus $F|_{x_I \leftarrow a_I}$ must be constant and it follows that a is feasible by Definition 6. $\square$

Theorem 5 indicates that every local minimum a in $[-1, 1]^n$ is meaningful in the sense that it can be easily converted into a Boolean assignment b. Moreover, since b is a Boolean assignment, it is not hard to see that the number of constraints satisfied by b is $\frac{m - F(a)}{2}$, which is a special case of Theorem 6 in the next section. Thus if a global minimum is too hard to reach, our method is still useful for some problems where quickly finding an assignment with good quality is enough, e.g., (weighted) MaxSAT.

In most multilinear polynomials generated from Boolean formulas in practice, saddle points rarely exist. However, there exist "pathological" polynomials that have uncountably infinite saddle points, as Example 9 shows.

Example 9. Let $f = x_1 \oplus x_2 \oplus x_3 \oplus x_4$, then $F_f = x_1 x_2 x_3 x_4$. Therefore every point which has the form $(x_1, 0, 0, 0)$, $(0, x_2, 0, 0)$, $(0, 0, x_3, 0)$ or $(0, 0, 0, x_4)$ is a degenerate saddle point where both the gradient and Hessian are zero.

To illustrate saddle points, in Fig. 3 we plot the gradient of two parity constraints: $x_1 \oplus x_2$ and $x_1 \oplus x_2 \oplus x_3$.

3.5. A randomized rounding and expectation perspective

Before introducing our projected-gradient-descent-based algorithm for minimizing multilinear polynomials, we would like to explain why doing continuous optimization might be better than discrete local search by viewing the objective function F_f from an expectation perspective.

Our explanation is, compared to local search SAT solvers which only make progress when the number of satisfied constraints increases, our tool makes progress as long as the value of the polynomial decreases, even by a small amount.

Theorem 6 formalizes the intuition. It indicates that when we decrease the value of the polynomial, we are in fact increasing the expectation of the number of satisfied constraints, after a randomized rounding.

[4] This can also be obtained by the KKT conditions.

Definition 10. (Randomized Rounding) The randomized rounding function, denoted by $\mathcal{R} : [-1, 1]^n \to \{\pm 1\}^n$ is defined by:

$$\begin{cases} \mathbb{P}[\mathcal{R}(a)_i = -1] = -\frac{1}{2}a_i + \frac{1}{2} \\ \mathbb{P}[\mathcal{R}(a)_i = +1] = +\frac{1}{2}a_i + \frac{1}{2} \end{cases}$$

for all $i \in [n]$ and $a \in [-1, 1]^n$.

Intuitively, the closer a_i is to -1, the more likely $\mathcal{R}(a)_i$ will be -1 and vice versa.

Definition 11. (Vector probability space) We define a probability space on Boolean vectors w.r.t. real point $a \in [-1, 1]^n$, denoted by $\mathcal{S}_a : \{-1, 1\}^n \to [0, 1]$ by:

$$\mathcal{S}_a(b) = \mathbb{P}[\mathcal{R}(a) = b] = \prod_{b_i=-1} \frac{1-a_i}{2} \prod_{b_i=1} \frac{1+a_i}{2},$$

for all $b \in \{-1, 1\}^n$, with respect to the randomized-rounding function $\mathcal{R}$. We use $b \sim S_a$ to denote that $b \in \{-1, 1\}^n$ is sampled from the probability space $\mathcal{S}_a$.

Theorem 6. *Given a multilinear polynomial F, for a real point $a \in [-1, 1]^n$, we have*

$$F(a) = \mathop{\mathbb{E}}_{b\sim\mathcal{S}_a}[F(b)],$$

where $\mathbb{E}$ denotes expectation.

Proof. We prove by induction on the number of variables n.

`Basis step`: Let $n = 1$. Then $F = \alpha \cdot x_1$ for some $\alpha \in \mathbb{R}$. We have

$$\mathop{\mathbb{E}}_{b\sim\mathcal{S}_a}[F(b)] = \alpha \cdot (\frac{a_1}{2} + \frac{1}{2}) + (-\alpha) \cdot (-\frac{a_1}{2} + \frac{1}{2}) = \alpha \cdot a_1 = F(a)$$

`Inductive step`: Suppose $n \geq 2$. Then, for $b \sim \mathcal{S}_a$, $F(b)$ can be expanded as:

$$F(b) = \tfrac{1-b_n}{2} \cdot F|_{x_n\leftarrow(-1)}(b_{[n-1]}) + \tfrac{1+b_n}{2} \cdot F|_{x_n\leftarrow 1}(b_{[n-1]})$$

Note that the value of b_n and $b_{[n-1]}$ are independent, thus we have

$$\begin{aligned} &\mathop{\mathbb{E}}_{b\sim\mathcal{S}_a}[F(b)] \\ =&\mathop{\mathbb{E}}_{b\sim\mathcal{S}_a}[\tfrac{1-b_n}{2} \cdot F|_{x_n\leftarrow(-1)}(b_{[n-1]}) + \tfrac{1+b_n}{2} \cdot F|_{x_n\leftarrow 1}(b_{[n-1]})] \\ =&\tfrac{1-a_n}{2} \cdot \mathop{\mathbb{E}}_{b\sim\mathcal{S}_a}[F|_{x_n\leftarrow(-1)}(b_{[n-1]})] + \tfrac{1+a_n}{2} \cdot \mathop{\mathbb{E}}_{b\sim\mathcal{S}_a}[F|_{x_n\leftarrow 1}(b_{[n-1]})] \end{aligned}$$

By I.H., the equation above can be simplified to:

$$\begin{aligned} &\mathop{\mathbb{E}}_{b\sim\mathcal{S}_a}[F(b)] \\ =&\tfrac{1-a_n}{2} \cdot F|_{x_n\leftarrow(-1)}(a_{[n-1]}) + \tfrac{1+a_n}{2} \cdot F|_{x_n\leftarrow 1}(a_{[n-1]}) \\ =&F(a) \quad \square \end{aligned}$$

Consider the multilinear polynomial generated by a formula according to Definition 2. We immediately derive the following corollary.

Corollary 2. *Let f be a formula with m constraints and F_f be the objective multilinear polynomial associated with f. Let $m_{SAT}(b)$ be the number of constraints satisfied by $b \in \{-1, 1\}^n$. For $a \in [-1, 1]^n$, we have*

$$\mathop{\mathbb{E}}_{b\sim\mathcal{S}_a}[m_{SAT}(b)] = \tfrac{m-F_f(a)}{2}.$$

In particular, if $F_f(a) = -m$, then $\mathop{\mathbb{E}}_{\mathcal{R}(a)}[m_{\mathtt{SAT}}(\mathcal{R}(a)] = m$. Since $m_{\mathtt{SAT}}(b) \leq m$ for all $b \in \{\pm 1\}^n$, we have $m_{\mathtt{SAT}}(\mathcal{R}(a)) = m$. Thus $\mathcal{R}(a)$ is a solution of f, which is consistent with Theorem 3.

Proof. For a discrete point $b \in \{-1, 1\}^n$,

$$F_f(b) = 1 \cdot m_{\text{SAT}}(b) + (-1) \cdot (m - m_{\text{SAT}}(b)) = 2m_{\text{SAT}}(b) - m$$

Thus by Theorem 6, we have

$$\underset{b \sim \mathcal{S}_a}{\mathbb{E}}[m_{\text{SAT}}(b)] = \frac{m}{2} - \frac{1}{2} \cdot \underset{b \sim \mathcal{S}_a}{\mathbb{E}}[F(b)] = \frac{m - F_f(a)}{2}. \quad \square$$

Remark 5. In fact, Theorem 3 can be proved independently from Theorem 6 without relying on multilinearity too much. However, we believe that the current order of presentation is more natural and informative.

Remark 6. The idea of evaluating Fourier Expansions and the result of Theorem 6 can find connections with circuit-output probability and spectral coefficients [81,82].

Related research showed that local search SAT solvers, such as GSAT, spend most of the time on the so-called “sideway” moves [14]. Sideway moves are moves that do not increase or decrease the total number of unsatisfied constraints. Although heuristics and adding noise for sideway moves lead the design of efficient solvers, e.g., WalkSAT, local search SAT solvers fail to provide any theoretical guarantee when making sideway moves.

It is illustrative to think of a cardinality constraint, e.g., the majority constraint which requires at least half of all the variables to be `True`. If we start from the assignment of all `False`, local search solvers need to flip at least half of all bits to make progress. In other words, local search solvers will encounter a neighborhood with exponential size where no progress can be made and any movements will be “blind”. In contrast, guided by the gradient, our method will behave like “flipping” all the variables by a small amount towards the solution zone.

We end this section by explaining why we choose the multilinear representation, since any arbitrary polynomial that maps $[-1, 1]^n$ to $[-1, 1]$, probably with higher order, could be a candidate in our method. For example, higher order polynomials have been used as the objective function for SAT solving [62]. We list the advantages of the multilinear representation over an arbitrary polynomial in the following.

1. The multilinear representation can handle all types of symmetric constraints while the polynomial representation in [62] only works for disjunctive clauses (CNF).
2. The value of the multilinear objective has a nice interpretation: the value of the polynomial is directly relevant with the expectation of the number of satisfied constraints. Moreover, all local minima can be rounded to binary assignments without any loss of objective value. Intuitively, all local optima are “almost” binary. In contrast, there can be “inner local optima” in higher order polynomial representations.
3. With properties of multilinear polynomials, the number of iterations needed for convergence (convergence speed) is bounded by a polynomial of the problem size, which will be shown in the next section.
4. The problems tested in the experiments of [62] are of relatively small size and easy ($n < 100$), while the instances in our experiments are considerably larger, as we will see in Section 5. This may be an evidence that the theoretical properties of multilinear polynomials are also beneficial in practice.

4. A gradient-descent-based algorithm for minimizing multilinear polynomials

In this section, we propose a set of algorithms for minimizing multilinear polynomials associated with Boolean formulas. We show promising theoretical guarantees that gradient-based algorithms can enjoy: a projected-gradient-descent-based algorithm converges to a critical point in polynomial time. We also explicitly consider the existence of saddle points and design algorithms to escape them though they seldom show up in practice.[5]

4.1. Non-convex optimization and saddle points

As mentioned in Definition 9, a saddle point is a point at which all the first derivatives are zero but there exist at least one positive and one negative direction. Again, saddle points are not local optima. In many cases, especially in those related to deep neural networks [83,84], the main bottleneck in optimization is not due to local minima, but the existence of many saddle points. Gradient based algorithms are in particular susceptible to the saddle point issue as they only rely on the gradient information. The saddle point problem is alleviated for second-order methods that also rely on the Hessian information [83]. Previous work such as [48,49] proposed that stochastic gradient descent with random noise is enough to escape saddle points. However, the strict saddle property, described in Definition 12, is either assumed or valid in the problems they studied.

[5] In the experimental result session, we take advantage of elaborate optimization packages instead of directly implementing our algorithms. The code and benchmarks are available at https://github.com/vardigroup/FourierSAT.

Definition 12. [48] (Strict saddle) Given a function F, a saddle point a is **strict** if all the eigenvalues of the Hessian at a are strictly negative, i.e., $\lambda_{max}(H_F(a)) < 0$. A function F has **strict saddle property** if all of F's saddle points are strict.

In contrast, strict saddle property is not valid for multilinear polynomials since there can be degenerate saddle points where all eigenvalues of the Hessian are zero (see Example 9 and Remark 4). Therefore, specific optimizing techniques for multilinear polynomials need to be designed.

4.2. Efficient evaluation of the objective function

Although theoretically the objective function can have up to 2^n terms, we do not actually store all the coefficients and do a naive evaluation to get $F_f(a)$ for $a \in [-1,1]^n$. Instead, we leverage the symmetry of constraints to compute the value of Fourier Expansion of each constraint c, denoted as $\texttt{FE}_c(a)$, separately. By this trick, we are able to evaluate the objective function in $O(\sum\limits_{c \in C_f} \texttt{length}(c)^2)$ time (in the worst case $O(n^2 m)$). Thus we can bypass considering data structures that store all coefficients of the multilinear polynomial.

First note that for every negative literal in a constraint, say $\neg x_i$, we can convert it to positive by flipping the sign of a_i, since applying a negation on a formula is equivalent to adding a minus sign to the Fourier Expansion in the Fourier domain.[6]

Now suppose c is a symmetric constraint with no negative literals. Thus c's Fourier coefficients can be denoted as

$$\texttt{Coef}(\texttt{FE}_c) = (\widehat{c}(\emptyset), \widehat{c}([1]), \widehat{c}([2]), \ldots, \widehat{c}([\texttt{length}(c)])) .$$

Let $\texttt{esp}(a) = \left(1, \sum_{i=1}^{\texttt{length}(c)} a_i, \sum_{i<j} a_i a_j, \ldots, \prod_{i=1}^{\texttt{length}(c)} a_i\right)$, known as the list of elementary symmetric polynomials. One can easily verify that $\texttt{FE}_c(a) = \texttt{Coef}(\texttt{FE}_c) \cdot \texttt{esp}(a)$, where "$\cdot$" denotes the inner product of two vectors.

Computing $\texttt{esp}(a)$ is equivalent to computing the coefficients of $\gamma^0, \gamma^1, \cdots, \gamma^{\texttt{length}(c)}$ in the expansion of the following polynomial of γ, i.e.,

$$\prod_{i=1}^{\texttt{length}(c)} (a_i + \gamma) = \gamma^{\texttt{length}(c)} + \left(\sum_{i=1}^{\texttt{length}(c)} a_i \right) \cdot \gamma^{\texttt{length}(c)-1} + \cdots + \prod_{i=1}^{\texttt{length}(c)} a_i \cdot \gamma^0$$

By a naive approach, the value of the list of elementary symmetric polynomials of size n can be computed in $O(n^2)$ time, which is efficient enough for small instance size. Thus we can evaluate the objective function in $O(\sum\limits_{c \in C_f} \texttt{length}(c)^2 \cdot m)$ or $O(n^2 m)$ in the worst case.[7]

Since the gradient and Hessian of a multilinear polynomial are still multilinear [42], we are able to calculate them analytically by the method above instead of relying on automatic differentiation. In experiments, we observed that feeding gradients to the optimizer significantly accelerated the minimization process.

4.3. Our PGD-based multilinear optimization algorithm

Since the objective function is constrained, continuous and differentiable, projected gradient descent (PGD) [85] is a candidate for solving our optimization problem. When solving a non-convex problem, the initialization plays a key role in the optimization algorithm. We use random initialization for each iteration and return the best result within a time limit.

In Algorithm 1, we propose a PGD-based algorithm for multilinear optimization problem. In gradient descent, the iteration for minimizing an objective function F is:

$$a'_{(t+1)} = a_{(t)} - \eta \cdot \nabla F(a_{(t)}), \;\; a_{(t+1)} = a'_{(t+1)},$$

where $\eta > 0$ is a step size and $a_{(t)}, a_{(t+1)}, a'_{(t+1)} \in \mathbb{R}^n$.

When the domain is bounded, $a'_{(t+1)}$ may be outside of the domain. In PGD, we choose the point nearest to $a'_{(t+1)}$ in $[-1,1]^n$ as $a_{(t+1)}$ [86], i.e., the Euclidean projection of $a'_{(t+1)}$ onto the set $[-1,1]^n$, denoted as $\prod_{[-1,1]^n}(a'_{(t+1)})$, as illustrated by Fig. 4.

Definition 13. The Euclidean projection of a point a, onto a set Δ, denoted by $\Pi_\Delta(a)$, is defined as

$$\Pi_\Delta(a) = \arg\min_{a' \in \Delta} \left\| a - a' \right\|^2 .$$

[6] Here we assume each variable appears at most once in a constraint c.

[7] In fact, there exists an FFT-based algorithm for evaluating the list of elementary symmetric polynomials of size n with complexity $O(n \log^2 n)$. Moreover, such an algorithm has the potential of exploiting the power of GPU. We leave leveraging GPU-based FFT implementation for future work. Another work of ours that uses BDDs to significantly accelerate the evaluation and gradient computation is described in [82].

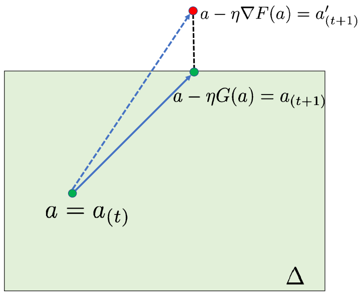

$a - \eta\nabla F(a) = a'_{(t+1)}$
$a - \eta G(a) = a_{(t+1)}$
$a = a_{(t)}$
Δ

Fig. 4. Projected gradient descent.

Algorithm 1: PGD for multilinear F.

```
Input  : A multilinear polynomial $F$, the accuracy $\varepsilon$, the step size $\eta$.
Output: A point $a^\star$ with low objective value.
1  for $j = 1, \dots, J$ do
2      $a_{(0)} \sim \mathcal{U}[-1,1]^n$
3      for $t = 0, \dots, T$ do
4          $G(a_{(t)}) = \frac{1}{\eta}\left(a_{(t)} - \Pi_{[-1,1]^n}\left(a_{(t)} - \eta\nabla F(a_{(t)})\right)\right)$
5          if $\|G(a_{(t)})\| > \varepsilon$ then
6              // the projected gradient is informative.
7              $a_{(t+1)} = a_{(t)} - \eta \cdot G(a_{(t)})$
8          else
9              // we meet a critical point or a local minimum.
10             if $a_{(t)}$ is not feasible then
11                 $a_{(t+1)} =$ DecInnerSaddle($F$, $a_{(t)}$, $\eta$)        //$a_{(t)}$ is an inner saddle point
12             else
13                 $I = \{i \mid (a_{(t)})_i \in \{-1,1\}\}$
14                 if $\nabla_i F(a_{(t)}) \neq 0,\ \forall i \in I$ then
15                     Break        // Local minimum identified by Proposition 7.
16                 else
17                     // try some second-order methods.
18                     (LocalMinFlag, $a_{(t+1)}$) = useHessian($F$, $a_{(t)}$)
19                     if LocalMinFlag=True or Unknown then
20                         Break
21 return $a_{(j)}$ with the lowest $F(a_{(j)})$ after $J$ iterations as $a^\star$
```

In our case $\Delta \equiv [-1,1]^n$; computing such a projection is almost free, as shown in Proposition 2.

Proposition 2.

$$\left(\Pi_{[-1,1]^n}(a)\right)_i = \begin{cases} a_i, & \textit{if } a_i \in [-1,1], \\ sgn(a_i), & \textit{otherwise}. \end{cases}$$

Proof.
- If $a_i \in [-1,1]$, suppose $\left(\Pi_{[-1,1]^n}(a)\right)_i \neq a_i$. Then we construct a vector $z \in [-1,1]^n$ as follows. Let $z_i = a_i$ and $z_j = \left(\Pi_{[-1,1]^n}(a)\right)_j$ for $j \neq i$. We have $z \in [-1,1]^n$ and $\|z-a\|^2 \le \left\|\Pi_{[-1,1]^n}(a) - a\right\|^2$, a contradiction with the definition of the projection.
- If $|a_i| > 1$, without loss of generality, suppose $a_i > 1$ and $\left(\Pi_{[-1,1]^n}(a)\right)_i < 1 = \text{sgn}(a_i)$. Let $z_i = 1$ and $z_j = \left(\Pi_{[-1,1]^n}(a)\right)_j$ for $j \neq i$. Similarly, $z \in [-1,1]^n$ and $\|z-a\|^2 \le \left\|\Pi_{[-1,1]^n}(a) - a\right\|^2$ still hold. □

Combining the above, the main iteration of PGD can be rewritten as

$$a_{(t+1)} = \Pi_{[-1,1]^n}\left(a_{(t)} - \eta \cdot \nabla F(a_{(t)})\right) = a_{(t)} - \eta G(a_{(t)}),$$

where

$$G(a) = \frac{1}{\eta}\left(a - \Pi_{[-1,1]^n}\left(a - \eta\nabla F(a)\right)\right)$$

is regarded as the *gradient mapping*, as marked in Fig. 4.

In Algorithm 1, we start at a uniformly random point in $[-1,1]^n$ (line 2). When the norm of the gradient mapping $G(\cdot)$ is large enough, we follow the gradient mapping to decrease the function value effectively, as formalized in Lemma 4.

Otherwise, it means that the algorithm either reaches a local minimum in $[-1,1]^n$, or falls into a saddle point where the original gradient $\nabla F(\cdot)$ is negligible. If the first case happens we are done for this trial. Otherwise, we still try to escape the saddle point by additional methods. Note that the feasibility check in line 10 can be implemented by Definition 6 combined with constant testing (Algorithm 4).

4.4. Convergence speed of PGD

In Theorem 7, we show that Algorithm 1 is guaranteed to converge to a point where the projected mapping is small, i.e., $\|G(a)\| < \varepsilon$, for a small accuracy level $\varepsilon > 0$, and depends polynomially on n, m and the tolerance ε.

Theorem 7. (Convergence speed) *With the step size $\eta = \frac{1}{nm}$, Algorithm 1 converges to a ε-projected-critical point $a^\star$ (where $\|G(a^\star)\| < \varepsilon$) in $O(\frac{nm^2}{\varepsilon^2})$ iterations in the worst case.*

The proof of Theorem 7 relies on several propositions and lemmas that we will introduce and prove below. Some ideas of this part are inspired by techniques from [48] and [86].

The first observation, summarized in Proposition 3, indicates that multilinear polynomials on $[-1,1]^n$ are relatively smooth.

Proposition 3. (Lipschitz) *Let $F : [-1,1]^n \to [-\alpha, \alpha]$ be a multilinear polynomial where $\alpha > 0$. Then, for every pair of points $y, z \in [-1,1]^n$, we have*

1. *$|F(y) - F(z)| \le \alpha\sqrt{n} \cdot \|y - z\|$. I.e., F is $(\alpha\sqrt{n})$-Lipschitz continuous.*
2. *$\|\nabla F(y) - \nabla F(z)\| \le \alpha n \cdot \|y - z\|$. I.e., F has (αn)-Lipschitz continuous gradients.*
3. *$\|\nabla^2 F(y) - \nabla^2 F(z)\| \le \left(\alpha n^{\frac{3}{2}}\right) \cdot \|y - z\|$. I.e., F has $\left(\alpha n^{\frac{3}{2}}\right)$-Lipschitz Hessians. Here $\|\cdot\|$ on the left-hand-side refers to the spectral norm of a matrix, defined by the square root of the maximum eigenvalue of $H^\top \cdot H$ for a matrix H.*

All the bounds are tight if we consider the parity function $F(x) = \prod_{i\in[n]} x_i$.

Proof. 1. By the triangle inequality,

$$\begin{aligned}
|F(y) - F(z)| &\le |F(y_1, \cdots, y_n) - F(z_1, y_2, \cdots, y_n)| \\
&\quad + |F(z_1, y_2, \cdots, y_n) - F(z_1, z_2, y_3, \cdots, y_n)| \\
&\quad + \cdots + |F(z_1, \cdots, z_{n-1}, y_n) - F(z_1, \cdots, z_n)| \\
&= |(y_1 - z_1)\nabla_1 F(y_1, ..., y_n)| + \cdots + |(y_n - z_n)\nabla_n F(z_1, ..., z_{n-1}, y_n)|
\end{aligned}$$

Since $\nabla_i F(x) = \frac{\partial F(x)}{\partial x_i} = \frac{1}{2}(F|_{x_i\leftarrow 1}(x) - F|_{x_i\leftarrow(-1)}(x)) \in [-\alpha, \alpha]$ for all $i \in [n]$, we have:

$$\begin{aligned}
&|(y_1 - z_1)\nabla_1 F(y_1, \cdots, y_n)| + \cdots + |(y_n - z_n)\nabla_n F(z_1, \cdots, z_{n-1}, y_n)| \\
\le &\alpha(\sum_{i=1}^n |y_i - z_i|) \le \alpha n^{\frac{1}{2}} \|y - z\|
\end{aligned}$$

The last inequality is an application of Cauchy-Schwartz Inequality.

2. Note that we can view $\nabla F(x)$ as a vector of n multilinear polynomials. In addition, $\nabla_i F(x) = \frac{\partial F(x)}{\partial x_i} \in [-\alpha, \alpha]$ for all $i \in [n]$. Thus we can apply 1. of Proposition 3 to all of these n multilinear polynomials:

$$\begin{aligned}
\|\nabla F(y) - \nabla F(z)\| &= (\sum_{i=1}^n (\nabla_i F(y) - \nabla_i F(z))^2)^{\frac{1}{2}} \le (n \cdot (\alpha n^{\frac{1}{2}} \|y - z\|)^2)^{\frac{1}{2}} \\
&= \alpha n \|y - z\|
\end{aligned}$$

3. Recall that $\nabla^2_{i,j} F(x)$ is the element at coordinate (i,j) of the second-order derivative matrix $\nabla^2 F(x)$. Similarly, we have $\nabla^2_{i,j} F(x) \in [-\alpha, \alpha]$ for all $i, j \in [n]$. Therefore, we can view $\nabla^2 F(x)$ as n^2 multilinear polynomials and apply 1 of Proposition 3 again.
In the following we will use the Frobenius norm of a matrix, i.e., $\|\cdot\|_F$, defined by $\|H\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^m H_{ij}^2}$, where $H \in \mathbb{R}^{n\times m}$ is a matrix. By $\|H\| \le \|H\|_F$, i.e., the spectral norm is no larger than Frobenius norm, we get:

$$\begin{aligned}
&\left\|\nabla^2 F(y) - \nabla^2 F(z)\right\| \\
\leq &\left\|\nabla^2 F(y) - \nabla^2 F(z)\right\|_F \\
= &\sqrt{\sum_{i,j\in[n]} (\nabla^2_{i,j} F(y) - \nabla^2_{i,j} F(z))^2} \\
\leq &\sqrt{n^2(\alpha\sqrt{n}\,\|y-z\|)^2} \quad \text{(by 1 of Proposition 3)} \\
= &\alpha n^{\frac{3}{2}}\,\|y-z\| \quad \square
\end{aligned}$$

Now that we know that multilinear polynomials are smooth from Proposition 3, the next thing to do is to bound the progress made by each iteration, as shown in Lemma 4. In order to prove Lemma 4, we first present several useful intermediate results, namely Proposition 4, 5 and 6. As well-known results in optimization, the proofs of Proposition 4 and 6 are taken from [87,88] respectively with some modifications, so that this paper is self-contained.

Proposition 4. *[87] Let h be a convex and differentiable function defined on a convex set Δ, let $y^\star = \arg\min_{y\in\Delta} h(y)$, then for all $y \in \Delta$ we have*

$$\langle \nabla h(y^\star), y - y^\star\rangle \geq 0$$

Proof. Assume that there exists $y \in \Delta$ such that

$$\langle \nabla h(y^\star), y - y^\star\rangle < 0$$

Consider the function $g(\gamma) = h(y^\star + \gamma(y - y^\star))$, $\gamma \in [0, 1]$. Note that

$$g(0) = h(y^\star)$$

$$\nabla g(0) = \lim_{\epsilon\to 0} \frac{h(y^\star + \epsilon(y - y^\star)) - h(y^\star)}{\epsilon} = \langle \nabla h(y^\star), y - y^*\rangle < 0$$

Therefore for small enough γ we have:

$$h(y^\star + \gamma(y - y^\star)) = g(\gamma) < g(0) = h(y^\star)$$

This is a contradiction. $\square$

Proposition 4 is useful because given a point $a \notin \Delta$, the square of the l^2-norm used in the projection (see Definition 13), a.k.a., $\left\|a - a'\right\|^2$, is a convex function of $a' \in \Delta$. We leverage this fact to prove Proposition 5, which indicates that the gradient mapping G is "not too far" from the direction of the original gradient ∇F. Therefore, the gradient mapping G should be an effective descending direction, given that the search is restricted within the box constraints.

Proposition 5. (Inequality of gradient mapping) *Given a function F defined on a convex set Δ, for every $a \in \Delta$, we have*

$$\langle \nabla F(a),\ G(a)\rangle \geq \|G(a)\|^2$$

where $G(a) = \frac{1}{\eta}\left(a - \Pi_\Delta\left(a - \eta\nabla F(a)\right)\right)$ is the gradient mapping.

Proof. The readers are encouraged to refer to Fig. 4 for an intuitive illustration. Let $h(y) = \|y - (a - \eta\nabla F(a))\|^2$ for $y \in \Delta$. Then $h(y)$ is a convex function. Since $\arg\min_{y\in\Delta} h(y)$ is exactly the projected point, i.e., $a - \eta G(a)$, by Proposition 4, for all $y \in \Delta$ we have

$$\langle \nabla h(a - \eta G(a)), y - (a - \eta G(a))\rangle \geq 0$$

Note that $\nabla h(a - \eta G(a)) = 2((a - \eta G(a)) - (a - \eta\nabla F(a))) = 2\eta(\nabla F(a) - G(a))$. Let $y = a$, the above inequality becomes:

$$\langle 2\eta(\nabla F(a) - G(a)), \eta G(a)\rangle \geq 0$$

and the statement follows directly. $\square$

Proposition 6 is a consequence of Lipschitz continuity which is widely used in optimization.

Proposition 6. *[87] Let F be a β-Lipschitz continuous function on $\mathbb{R}^n$. Then for every $y, z \in \mathbb{R}^n$, we have*

$$|F(y) - F(z) - \langle \nabla F(z), (y-z)\rangle| \leq \frac{\beta}{2}\|y-z\|^2$$

Proof. We represent $F(y) - F(z)$ as an integral, apply Cauchy-Schwarz and then β-Lipschitz continuous.

$$\begin{aligned}
&|F(y) - F(z) - \langle \nabla F(z), (y-z)\rangle| \\
=&\left|\int_0^1 \langle \nabla F(z+t(y-z)), (y-z)\rangle dt - \langle \nabla F(z), (y-z)\rangle\right| \\
\leq&\int_0^1 \|\nabla F(z+t(y-z)) - \nabla F(z)\| \cdot \|y-z\|\, dt \quad \text{(Cauchy-Schwarz)} \\
\leq&\int_0^1 \beta t\,\|y-z\|^2\, dt \quad (\beta\text{-Lipschitz continuous}) \\
=&\frac{\beta}{2}\|y-z\|^2 \quad \square
\end{aligned}$$

With the help of Proposition 6, Lemma 4 quantitatively measures the progress made by one iteration guided by the gradient mapping G.

Lemma 4. (Descent Lemma) *For a multilinear polynomial under box constraints $F : [-1,1]^n \to [-\alpha, \alpha]$ and its projected gradient mapping $G(\cdot)$, if the step size satisfies $\eta \leq \frac{1}{\alpha n}$, then the projected gradient descent (PGD) sequence $\{a_{(t)}\}$ satisfies:*

$$F(a_{(t+1)}) - F(a_{(t)}) \leq -\tfrac{\eta}{2}\|G(a_{(t)})\|^2.$$

Proof. According to the (αn)-gradient Lipschitz continuity property in Proposition 3, applying Proposition 6, we have:

$$\begin{aligned}
F(a_{(t+1)}) &\leq F(a_{(t)}) + \langle \nabla F(a_{(t)}),\ a_{(t+1)} - a_{(t)}\rangle + \tfrac{\alpha n}{2}\cdot \|a_{(t+1)} - a_{(t)}\|^2 \\
&= F(a_{(t)}) - \eta\langle \nabla F(a_{(t)}),\ G(a_{(t)})\rangle + \tfrac{\eta^2 \alpha n}{2}\left\|G(a_{(t)})\right\|^2
\end{aligned}$$

Applying Proposition 5 and $\eta \leq \frac{1}{\alpha n}$, we have

$$F(a_{(t+1)}) \leq F(a_{(t)}) - \tfrac{\eta}{2}\left\|G(a_{(t)})\right\|^2 \quad \square$$

Now we are ready to prove Theorem 7, the result regarding convergence speed.

Proof of Theorem 7. Assume that $\eta = \frac{1}{nm}$. Then, the recursion in Lemma 4 satisfies:

$$F(a_{(t+1)}) \leq F(a_{(t)}) - \tfrac{1}{2nm}\|G(a_{(t)})\|^2.$$

Combining all the iterations together, for T iterations, we have:

$$\begin{aligned}
F(a_{(T+1)}) &\leq F(a_{(T)}) - \tfrac{1}{2nm}\|G(a_T)\|^2 \\
F(a_{(T)}) &\leq F(a_{(T-1)}) - \tfrac{1}{2nm}\|G(a_{(T-1)})\|^2 \\
&\vdots \\
F(a_{(1)}) &\leq F(a_{(0)}) - \tfrac{1}{2nm}\|G(a_{(0)})\|^2
\end{aligned}$$

Summing all these inequalities, and under the observation that $F(a^\star) \leq F(a_{(T+1)})$, we get the following:

$$\tfrac{1}{2nm}\sum_{t=0}^{T}\|G(a_{(t)})\|^2 \leq F(a_{(0)}) - F(a^\star).$$

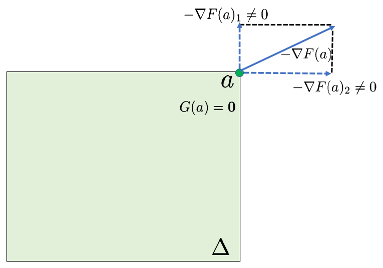


Fig. 5. Identifying local optima by Proposition 7. When all gradients at a are "pushing" a towards the boundary "tightly", a is a local optimum.

This implies that, even if we continue running gradient descent for many iterations, the sum of gradient norms is always bounded by something constant; this indicates that the gradient norms that we add at the very end of the execution has to be small, which further implies convergence to a stationary point.

Further, we know:

$$(T+1)\cdot \min_t \|G(a_{(t)})\|^2 \le \sum_{t=0}^{T} \|G(a_{(t)})\|^2.$$

Then,

$$\begin{aligned}
&\tfrac{T+1}{2nm}\cdot \min_t \|G(a_{(t)})\|^2 \le \tfrac{1}{2nm}\sum_{t=0}^{T} \|G(a_{(t)})\|^2 \le F(a_{(0)}) - F(a^\star)\\
&\Rightarrow \min_t \|G(a_{(t)})\|^2 \le \tfrac{2nm}{T+1}\cdot \big(F(a_{(0)}) - F(a^\star)\big)\\
&\min_t \|G(a_{(t)})\| \le \sqrt{\tfrac{2nm}{T+1}}\cdot \big(F(a_{(0)}) - F(a^\star)\big)^{\frac{1}{2}} = O\left(\tfrac{1}{\sqrt{T}}\right).
\end{aligned}$$

Note that $|F(a_{(0)}) - F(a^*)| \le m$. Thus, to achieve a point such that $\|G(a_{(T)})\| \le \varepsilon$, we require:

$$\min_t \|G(a_{(t)})\| \le \varepsilon \Rightarrow \sqrt{\tfrac{2nm}{T+1}}\cdot \big(F(a_{(0)}) - F(a^\star)\big)^{\frac{1}{2}} \le \varepsilon \Rightarrow O\left(\tfrac{nm^2}{\varepsilon^2}\right) \text{ iterations.} \quad \square$$

4.5. *Identifying local optima and escaping saddle points*

As non-convex functions, constrained multilinear polynomials are inherently difficult to minimize. In this subsection, we show our efforts towards identifying local optima and escaping saddle points by approaches from different levels, including using first-order gradient, second-order Hessian and geometry of multilinear polynomials.

4.5.1. *First-order methods*

Although Algorithm 1 could not eliminate the possibility of converging to a saddle point under some pathological cases (see Example 9), it is able to detect local minima by some sufficient conditions. Proposition 7 gives a sufficient condition based on gradient information. See Fig. 5 for an illustration.

Proposition 7. *When Algorithm 1 reaches line 15, a is a local minimum. I.e., if a is feasible,* $G(a) = \mathbf{0}$ *and* $\nabla_i F(a) \neq 0$ *for all* $i \in I$ *(i.e.,* $a_i \in \{-1, 1\}$*), then a is a local minimum in* $[-1, 1]^n$ *of F.*

Remark 7. In the context of discrete local search algorithms for SAT, a binary assignment that satisfies the condition in Proposition 7 is a point where flipping one arbitrary bit will strictly decrease the number of satisfied constraints. Thus this point is also a local optimum in discrete context. Note that in discrete context there are local optima where flipping one arbitrary bit does not change the objective. However, such a point is not necessarily a local minimum in our continuous setting. In other words, our approach has the potential to find decreasing directions for local minima in the discrete domain, as demonstrated in Example 10.

Example 10. Let $f = x_1 \wedge x_2$, then $F_f = \frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_1x_2$. As Fig. 6 shows, $(1, 1)$ is a local minimum in the discrete setting. However, $(1, 1)$ is not a local minimum in the continuous space.

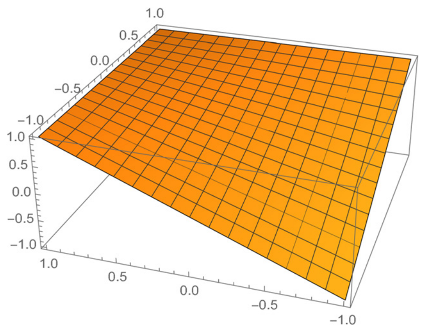


Fig. 6. The 3D-surface of the polynomial $\frac{1}{2}+\frac{1}{2}x_1+\frac{1}{2}x_2-\frac{1}{2}x_1x_2$, which is the Fourier Expansion of $x_1 \wedge x_2$.

Proof of Proposition 7. First, we consider the state of each coordinate of a. For every $i \in [n]$, by the definition of gradient mapping and $(G(a))_i = 0$ we have

$$a_i = \Pi_{[-1,1]}(a_i - \eta \nabla_i F(a)).$$

- If $a_i \in (-1, 1)$, i.e., $a \notin I$, since a is feasible, then $\nabla_i F(a) = 0$.
- Else we have $i \in I$. If $a_i = -1$, by the definition of projection and a is a fixed point, we have $\nabla_i F(a) > 0$. Similarly, if $a_i = 1$, then $\nabla_i F(a) < 0$.

Next, we need to prove that a has the lowest function value among all the points in the intersection of a's neighborhood $\mathcal{N}_\delta(a)$ and cube $[-1, 1]^n$. Consider all the directions v ($\|v\| \neq 0$) where a can move a small step to and $a + \eta \cdot v$ still remains in $\mathcal{N}_\delta(a) \cup [-1, 1]^n$. We have the following constraints for v:

$$\begin{cases} v_i \geq 0, \text{ if } a_i = -1, \\ v_i \leq 0, \text{ if } a_i = 1. \end{cases}$$

If $a_i \in (-1, 1)$, there is no restriction for v_i.

- If for all $i \in I$, $v_i = 0$. Since a is feasible, changing the value of variables with indices only from $[n] - I$ does not change the function value. Thus $F(a) = F(a + \eta \cdot v)$ and a is a trivial local minimum.
- Else, there exists $i' \in I$ s.t. $v_{i'} \neq 0$. The directional derivative at a in direction v, denoted as $\nabla_v F(a)$, can be computed by:

$$\nabla_v F(a) = \langle v, \nabla F(a) \rangle = \sum_{i \in [n]} v_i \nabla_i F(a).$$

First, note that $v_i \nabla_i F(a) \geq 0$ for all $i \in [n]$, thus $\nabla_v F(a) \geq 0$. Since $\nabla_{i'} F(a) \neq 0$ and $v_{i'} \neq 0$, we have $\nabla_v F(a) > 0$. Therefore, moving from a by the direction v will increase the objective value. In other words, $F(a) < F(a + \eta \cdot v)$ and a is a local minimum. □

4.5.2. Leveraging the geometry of multilinear polynomials

From Theorem 4 we know that for a multilinear polynomial in the unconstrained setting, there is always a decreasing direction despite of the position of the point. Thus for every interior point, a.k.a., every point in $(-1, 1)^n$, we can always decrease the function value. `DecInnerSaddle` (Algorithm 2) implements this idea.

Note that $F_{\texttt{shift}}$ in line 2 of `DecInnerSaddle` is the shifted polynomial function where a is shifted to the origin with $F_{\texttt{shift}}(\mathbf{0}) = 0$. After shifting the objective function, Algorithm 2 invokes Algorithm 3, which is a recursive implementation of Lemma 3 (see Remark 3). Note that `isZero` is used for checking if a point is feasible by testing whether a multilinear polynomial is equivalent to 0. We use Example 11 to demonstrate how our algorithm can escape from a saddle point.

Proposition 8. *Suppose the procedure* `isZero` *is always correct. When a is a non-feasible critical point, Algorithm 2 always gives a decreasing direction.*

Proof. Algorithm 2 is a recursive implementation of Lemma 3. □

Example 11. Consider the case in Example 5 and Fig. 2, where $f = (x_1 \vee \neg x_2) \wedge (\neg x_1 \oplus x_2)$ and $F_f = -\frac{1}{2} + \frac{1}{2}x_1 - \frac{1}{2}x_2 - \frac{3}{2}x_1x_2$. Suppose Algorithm 2 is at the saddle point $x_1 = -\frac{1}{3}$, $x_2 = \frac{1}{3}$. Then $F_{\texttt{shift}}(x) = F_f(x_1 - \frac{1}{3}, x_2 + \frac{1}{3}) - F_f(-\frac{1}{3}, \frac{1}{3}) = -\frac{3}{2}x_1x_2 =$

$x_1 \cdot (-\frac{3}{2}x_2) + 0$. By the notations in Lemma 3, we have $h(x_2) = 0$ is constant and $g(x_2) = -\frac{3}{2}x_2$ is non-constant but with $g(0) = 0$. Thus by recursively invoking Algorithm 3 on $F = -\frac{3}{2}x_2$ we get a decreasing direction $v^- = (1, -\mathrm{sgn}(-\frac{3}{2})) = (1, 1)$.

Algorithm 2: `DecInnerSaddle`(F, a, η).

```
Input : A multilinear polynomial $F$, a non-feasible critical point $a$, the step size $\eta > 0$.
Output: The new point after moving towards a negative direction.

1  $I \leftarrow \{i \mid a_i \in \{-1, 1\}\}$
2  $F_{\mathtt{shift}}(x) := F|_{x_I \leftarrow a_I}(x + a_{[n]-I}) - F(a)$
3  $v = \mathtt{NegDirectionSaddle}(F_{\mathtt{shift}})$
4  return $a + \eta \cdot v$
```

Algorithm 3: `NegDirectionSaddle`(F).

```
Input : A multilinear polynomial $F$ that satisfies $F(\mathbf{0}) = 0$.
Output: A negative direction $v^-$ of $F$ at point $\mathbf{0}$

1  /*                              See the proof of Lemma 3                              */
2  /*                                 $F = x_1 \cdot g + h$                                */
3  if $isZero(F|_{x_1 \leftarrow 0}) = True$ then
4      // $h$ is constant
5      if $isZero(F|_{x_1 \leftarrow 1} - F(1, 0, ..., 0)) = True$ then
6          // $g$ is constant
7          $v^- = (-\mathrm{sgn}(F|_{x_1 \leftarrow 1}), 0, ..., 0)$
8      else
9          // $g$ is non-constant
10         if $F(1, 0, ..., 0) = 0$ then
11             // $g(\mathbf{0}) = 0$
12             $v^- = (1, \mathtt{NegDirectionSaddle}(F|_{x_1 \leftarrow 1}))$
13         else
14             // $g(\mathbf{0}) \neq 0$
15             $v^- = (-\mathrm{sgn}(F(1, 0, ..., 0), 0, ..., 0))$
16 else
17     // $h$ is non-constant
18     $v^- = (0, \mathtt{NegDirectionSaddle}(F|_{x_1 \leftarrow 0}))$
19 return $v^-$
```

Algorithm 4: `isZero`(F).

```
Input : A polynomial $F$
Output: Correct answer of True/False on $[F \equiv 0]$ with probability at least $1 - \frac{n}{|S|}$ //See Proposition 11

1  Let $S \subset \mathbb{R}$ be a finite large set.
2  $a \sim \mathcal{U}[S^n]$
3  return True if $F(a) = 0$, otherwise False
```

Algorithm 5: `useHessian`(F, x).

```
Input : A polynomial $F$, a feasible critical point $a$.
Output: $a \in [-1, 1]^n$;
        LocalMinFlag $\in$ {True, False, Unknown} which indicates if $a$ is a local minimum of $F$ in $[-1, 1]^n$.

1  Let $J = \{j \mid \nabla F(a)_j = 0\}$  and  $I = \{i \mid a_i \in \{\pm 1\} \wedge i \in J\}$
2  $H = \nabla^2 F|_{(x_{[n]-J}) \leftarrow (a_{[n]-J})}(a_J)$
3  if there exist $i \in I,\ j \in J - I$ such that $H_{ij} \neq 0$ then
4      $v = (0, 0, ..., v_i = -\mathrm{sgn}(a_i), 0, ..., 0, v_j = \mathrm{sgn}(a_i \cdot H_{ij}), 0, ..., 0)$      // see Proposition 9
5      return $(a + \eta \cdot v$, False$)$
6  else if there exist $i_1, i_2 \in I$ such that $H_{ij} \cdot a_{i_1} a_{i_2} < 0$ then
7      $v = (0, 0, ..., v_i = -\mathrm{sgn}(a_{i_1}), 0, ..., 0, v_j = -\mathrm{sgn}(a_{i_2}), 0, ..., 0)$      // see Proposition 9
8      return $(a + \eta \cdot v$, False$)$
9  else if $\forall i, j,\ i \neq j$, at least one of $i$ and $j$ is in $I$ such that $H_{ij} \neq 0$ then
10     return $(a$, True$)$                                                              // See Proposition 10
11 else
12     return $(a$, Unknown$)$ //We meet a very rare degenerate saddle point on the boundary. Do random restart
```

4.5.3. Second-order methods

Algorithm 2 and 3 only work if the saddle point is an inner point of the cube. If Algorithm 1 falls into a point which is on the boundary of the cube but not a local minimum, it invokes `useHessian` to use second-order derivatives, attempting to give a negative direction, or to identify a local minimum. Proposition 9 and 10 provide sufficient conditions for finding negative directions and identifying local minima by second-order Hessian information, respectively.

Remark 8. The Hessian-based approaches in the continuous setting can be interpreted as the counterpart of flipping two variables simultaneously in the discrete local search setting.

Proposition 9. *When Algorithm 5 reaches line 4 or 7, v is a negative direction of F, i.e., there exist step size $\eta > 0$ such that $F(a+\eta v) < F(a)$ and $a + \eta v$ is within the cube $[-1, 1]^n$.*

Proof. Consider partially assigned function $F' = F|_{x_{[n]-J} \leftarrow a_{[n]-J}}$. By definition of J, $\nabla F'(a_J) = \mathbf{0}$. In the following cases, we introduce the *second directional derivative* of F' at a_J in direction v, denoted by $\nabla_v^2 F'(a_J)$, which can be computed by:

$$\nabla_v^2 F'(a_J) = \sum_{i,j\in J} v_i v_j H_{ij} = 2 \cdot \sum_{i,j\in J, i\neq j} v_i v_j H_{ij},$$

where H is the Hessian of F' at a_J and the last step follows by multilinearity of F' ($H_{ii} = 0$ for every $i \in [n]$).

- Suppose Algorithm 5 reaches line 4. First note that given η being small enough, $a_J + \eta \cdot v \in [-1, 1]^{|J|}$ because $a_i \in \{\pm 1\}$ and $a_i - \eta \cdot \mathtt{sgn}(a_i) \in [-1, 1]$ for all $i \in I$; $a_j \in (-1, 1)$ and $a_j + \mathtt{sgn}(a_i \cdot H_{ij}) \in [-1, 1]$ for all $j \in I - J$. Moreover, the following holds:

  $$\nabla_v^2 F'(a_J) = 2 \cdot (-\mathtt{sgn}(a_i))\mathtt{sgn}(a_i \cdot H_{ij}) \cdot H_{ij} < 0.$$

  Thus, moving towards direction v from a will decrease $\nabla_v F'$. Since $\nabla F'(a_J) = \mathbf{0}$ we have $\nabla_v F'(a_J) = 0$. Therefore $\nabla_v F'(a_J + \eta v) < 0$ and $F'(a_J + v) < F'(a_J)$, which means that v is a negative direction of F' at a. v is also a negative direction of F at a since v keeps coordinates of a in $[n] - J$ unchanged.
- Suppose Algorithm 5 reaches line 7. It is easy to verify that $a_J + \eta \cdot v \in [-1, 1]^{|J|}$. Meanwhile,

  $$\nabla_v^2 F'(a_J) = 2 \cdot (-\mathtt{sgn}(a_{i_1}))(-\mathtt{sgn}(a_{i_2})) \cdot H_{ij} < 0$$

  Note that the inequality follows by the condition in line 6 of Algorithm 5. Similar analysis with the case above guarantees v is a negative direction. □

Proposition 10. *When Algorithm 5 reaches line 10, a is a local minimum.*

Proof. Again, let $F' = F|_{x_{[n]-J} \leftarrow a_{[n]-J}}$. We have $\nabla F'(a_J) = \mathbf{0}$. Recall that

$$\nabla_v^2 F'(a_J) = 2 \sum_{i,j\in J, i\neq j} v_i v_j H_{ij}$$

Consider all the directions v ($\|v\| \neq 0$) where $a + \eta \cdot v \in [-1, 1]^{|J|}$. When the algorithm executes line 10, neither of condition in line 3 nor 6 holds. In addition, $H_{ij} = 0$ holds for every $i, j \in J - I$ because a_J is a feasible assignment of F'. Thus $F'|_{x_I \leftarrow a_I}$ is constant. Therefore $v_i v_j H_{ij} \geq 0$ for all $i, j \in J$ and $\nabla_v^2 F'(a_J) = 2 \sum_{i,j\in J, i\neq j} v_i v_j H_{ij} \geq 0$.

- If v has two or more non-zero elements, then $\nabla_v^2 F'(a_J) > 0$. Since $\nabla_v F'(a_J) = 0$, $F'(a_J) < F'(a_J + \eta v)$.
- Else if v has only one non-zero element, say $v_1 \neq 0$, then $\nabla_v^2 F'(a_J + \eta \cdot v) = H_{11} = 0$. Thus moving towards direction v from a does not change $\nabla_v F'$. Combining this fact and $\nabla_1 F'(a_J) = 0$, we know that moving towards v does not change F', i.e., $F'(a_J) = F'(a_J + \eta v)$.

Therefore, a_J is a local minimum of F'.

Next, we will show a is a local minimum of F as well. Suppose a is not a local minimum of F and there exists v s.t. $a + \eta \cdot v \in [-1, 1]^n$ and $F(a + \eta \cdot v) < F(a)$. Then there exists $i \in [n] - J$ s.t. $v_i \neq 0$, otherwise a_J is not a local minimum of F'. However, recall that $G(a) = \mathbf{0}$ and $\nabla_i F(a) \neq 0$ since $i \in [n] - J$. The directional derivative $\nabla_v F(a) > 0$ holds by similar analysis in the proof of Proposition 7, which conflicts with $F(a + \eta \cdot v) < F(a)$. □

In Proposition 8 we assume that `isZero` is always correct. While this can be done by using symbolic representations of multilinear polynomials, in practice it is not efficient. Instead, we can use a light approach to check whether a polynomial is constant probabilistically. As a result of the Schwartz-Zippel Lemma, Proposition 11 shows how we can test whether a multilinear polynomial is constant with high probability when we sample from a large finite subset of the universe.

Proposition 11. *(Schwartz-Zippel Lemma) Let $F : \mathbb{R}^n \to \mathbb{R}$ ($F \not\equiv 0$) be a multilinear polynomial and $S \subset \mathbb{R}$ be a finite set with $|S| > n$. If we pick a point $a = (a_1, \cdots, a_n)$, where each element is sampled independently and uniformly at random from S, then*

$$\mathbb{P}_{a\sim\mathcal{U}[S^n]}[F(a) = 0] \leq \frac{n}{|S|}.$$

Therefore, if a multilinear polynomial is not constant, we are able to find a witness by $\lceil \frac{\ln \delta}{\ln \frac{n}{|S|}} \rceil$ samples with probability at least $1 - \delta$.

4.6. Weighted case

The weighted version of the objective function is defined as

$$F_{f,w} = \sum_{c\in C_f} w(c) \cdot \mathtt{FE}_c$$

where $w : C_f \to \mathbb{R}$ is the weight function. It is easy to verify that the weighted version of Theorem 3 and Theorem 5 still hold. The weighted case can be useful in the following cases:

- **Vanishing Gradient.** When a constraint contains a large number of variables (e.g., a global cardinality constraint), we observe that the gradient given by this constraint on a single variable becomes negligible. By assigning large weights to long constraints, the varnishing gradient problem can be alleviated.
- **Adaptive Weighting.** By increasing the weights of unsatisfied constraints, the landscape of the objective function can be refined and the algorithm might be able to escape local optima. This technique has been applied in discrete local search SAT/MaxSAT solvers [53,23].
- **Weighted MaxSAT.** Our tool can be easily adapted to accepting weighted MaxSAT problems.

5. Experimental results

In this section we compare our tool, `FourierSAT` with state-of-the-art SAT and MaxSAT solvers.

5.1. Experiment setup

We choose `SLSQP` [89], implemented in `scipy` [90] as our optimization oracle, after comparing available algorithms for non-convex, constrained optimization.

Since random restart is used, different iterations are independent. Thus, we parallelized `FourierSAT` to take advantage of the multicore computation resource. Each experiment was run on an exclusive node in a homogeneous Linux cluster. These nodes contain 16-processor cores at 2.63 GHz each with 1 GB of RAM per node. We implement two versions of `FourierSAT`, namely the single-core version `FourierSAT-1core` and the multi-processor version `FourierSAT-16core`. The time cap (wall-clock time) for each job is 1 minute.

We compare our method with the following SAT/pseudo-Boolean solvers:

- Cryptominisat [37] (CMS), an advanced SAT solver supporting `CNF+XOR` constraints by enabling Gauss Elimination.
- WalkSATlm [91], an efficient C implementation of the local search SAT algorithm WalkSAT.
- MiniCARD [38], a MiniSAT-based `CARD-CNF` solver. It has been implemented in `pysat` [92].
- MonoSAT [41], a SAT Modulo Theory solver which supports a large set of graph predicates.
- RoundingSAT [40], a conflict-driven-pseudo-Boolean-Search-based (CDPB) solver.
- NAPS [93] and Open-WBO [94], two pseudo-Boolean solvers based on CDCL SAT solver cores,

and the following MaxSAT solvers:

- Loandra [95], the best partial MaxSAT solver in MaxSAT Competition 2019.
- WalkSAT (v56) [15], an incomplete MaxSAT solver based on WalkSAT algorithm.
- Discrete local search. We implemented a naive version of local search SAT solver. Per iteration, we flip the bit which can maximize the number of satisfied constraints until no improvements can be made. Ties are broken randomly.

MiniCARD can handle cardinality constraints natively. For MonoSAT, we reduced cardinality constraints to Max-Flow. For CMS and WalkSAT, we applied a set of cardinality encodings, which includes Sequential Counter [96], Sorting Network [97], Totalizer [98] and Adder [99]. For the sake of clarity, we only show the results of the best and worst encoding for each solver in the plots. For pseudo-Boolean constraints, all disjunctive clauses are treated as cardinality constraints. For solvers that do not support `XOR`s, we used a linear encoding due to [100] to decompose them into 3-`CNF`. To alleviate the vanishing gradient issue, in `FourierSAT`, the weight of each constraint equals to its length.

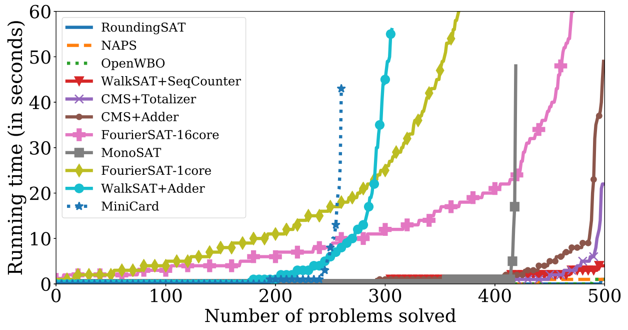


Fig. 7. Results on 500 vertex covering problems. The legends are sorted according to the performance of solvers.

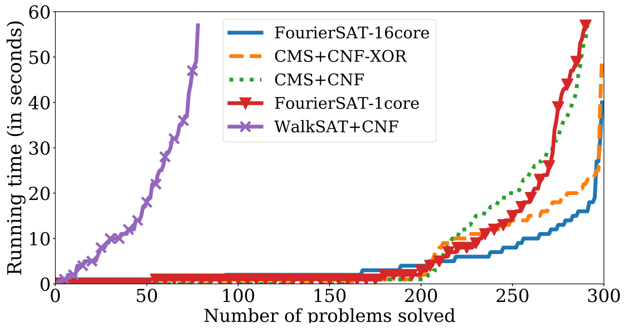


Fig. 8. Results on 300 parity learning with error problems. The legends are sorted according to the performance of solvers.

5.2. Benchmarks

We generated instances with hybrid Boolean constraints from the following benchmark problems.[8]

Benchmark 1: Approximate minimum vertex covering Minimum vertex covering is a well-known NP-optimization problem (NPO). For each $n \in \{50, 100, 150, 200, 250\}$, we generated 100 random cubic graphs. For each graph, `Gurobi` [101] was used to compute the minimum vertex cover, denoted by `Opt`. Then we added one cardinality constraint, $\text{CARD}^{\leq k}$ where $k = \lceil 1.1 \cdot |\texttt{Opt}| \rceil$ to the CNF encoding of vertex covering.

Benchmark 2: Parity learning with error Parity Learning is to identify an unknown parity function given I/O samples. In other words, the task is to learn the subset of variables which appear in the unknown parity function. Suppose there are n candidate variables and m I/O samples. A solution to this problem is allowed to be incorrect on at most $(e \cdot m)$ I/O samples. For $e = 0$ the problem is in P (Gaussian Elimination); for $0 < e < \frac{1}{2}$, whether the problem is in P still remains open. Parity learning is a well-known hard problem for SAT solvers, especially for local search solvers [102].

We chose $N \in \{8, 16, 32\}$, $e = \frac{1}{4}$ and $m = 2n$ to generate hard cases. For `FourierSAT`, this problem can be encoded into solving M XOR constraints where we allow at most eM constraints to be violated by using the `-tolerance` option. For WalkSAT and CMS, we used the encoding due to [103].

Benchmark 3: Random CNF-XOR-CARD formulas For each $n \in \{50, 100, 150\}$, $r \in \{1, 1.5\}$, $s \in \{0.2, 0.3\}$, $l \in \{0.1, 0.2\}$ and $k \in \{0.4n, 0.5n\}$, we generated random benchmarks with rn 3-CNF constraints, sn XOR constraints with length ln and 1 global cardinality constraint $\text{CARD}^{\leq k}$. Those parameters are chosen to guarantee that all problems are satisfiable [104] [105].

Benchmark 4: Small-size MaxSAT benchmarks We gathered 575 instances from the `crafted` and `random` track of MaxSAT competition 2016. We do not consider partial MaxSAT cases with both hard and soft constraints. The reason is that, in those cases, a feasible assignment should first satisfy all the hard constraints, which is non-trivial for `FourierSAT`.

5.3. Results

The experimental results of four benchmarks above are shown in Fig. 7, 8, 9 and Table 3 respectively. The legends in figures and entries in the table are sorted according to the ranking of solvers.

[8] All benchmarks are available at https://github.com/vardigroup/FourierSAT.

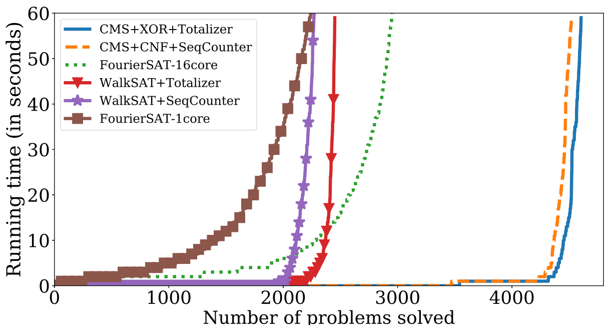


Fig. 9. Results on 4800 random `CNF-XOR-CARD` problems. The legends are sorted according to the performance of solvers.

Table 3
Results on MaxSAT instances.

Methods	Avg. Score	#Win
WalkSAT	0.926	168
FourierSAT-16core	0.917	223
Mixing Method	0.901	285
Loandra	0.883	47
FourierSAT-1core	0.865	35
Discrete Local Search	0.843	56

Fig. 7 shows the results on the vertex covering benchmarks. First of all, all three pseudo-Boolean solvers (RoundingSAT, NAPS and OpenWBO) quickly solved all instances, followed by CMS with different encodings. Compared with solvers that naturally handle cardinality constraints, the single core version `FourierSAT-1core` solved more problems (367) than MiniCard (261) but less than MonoSAT (420). The performance of two versions of `FourierSAT` is between that of WalkSATlm with the worst (Adder, 307) and the best (SeqCounter, 500) encoding. With the help of 16 processors, `FourierSAT-16cores` was able to solve 472 instances.

In Fig. 8 for parity learning problem, CMS+`CNF-XOR` encoding was able to solve all instances. The competition of `FourierSAT-1core` (290) and CMS with pure `CNF` formula (292) is roughly a tie. WalkSATlm with `CNF` encoding only solved 79 problems. By exploiting 16 cores, `FourierSAT-16core` solved all 300 problems.

According to the results on random hybrid constraints benchmarks shown in Fig. 9, CMS with different encodings wins with a large margin (4605 instances solved by Totalizer and 4520 by SeqCounter). `FourierSAT-1core` (2245) solved almost the same number of problems with WalkSATlm equipped with the worst encoding (SeqCounter, 2272). With more cores `FourierSAT-16core` (2957) is able to beat WalkSATlm with the best encoding (Totalizer, 2452) in the “unfair” competition.

Table 3 lists the results on MaxSAT benchmarks. We adapted `FourierSAT` to an incomplete MaxSAT solver and computed the `score`, which is often used as a merit in the evaluation of MaxSAT Competition, for each solver on each instance. For a given instance, let the number of violated constraints given by i-th solver be $\texttt{cost}_i$. Let the least number of violated constraints from results of all solvers be $\texttt{cost}_{best}$. Then the `score` for solver i on the case is:

$$\texttt{score}_i = \frac{\texttt{cost}_{best} + 1}{\texttt{cost}_i + 1}$$

For each solver, we also count the number of instances where this solver provides the solution with the best cost among all solvers (#Win). While WalkSAT (v56) achieves the best average `score`, `FourierSAT-16core` provides the second best performance on both the average `score` and #Win. With a single core, `FourierSAT-1core` still exhibits a better average `score` than the naive discrete local search.

Summary. We would like to emphasize the fact that `FourierSAT` is a versatile solver, which is especially valuable when we observe that different encodings do make a difference for CNF solvers. Thus it is not surprising that `FourierSAT` can be outperformed by specialized solvers. For example, when it comes to handling cardinality constraints, it is still better to choose pseudo-Boolean solvers over both `FourierSAT` and CNF solvers with encodings. The general performance of our proof-of-concept implementation of `FourierSAT` on a single core is generally less competitive than that of mature SAT/MaxSAT solvers. However, when given more cores, `FourierSAT` is comparable with many existing, well-developed solvers. This gives us the hope that with a refined implementation, our framework might be able to exhibit more satisfactory performance as a versatile solver. We believe that algebraic methods are worth exploring for the Boolean SAT/MaxSAT community in the future.

6. Conclusion and future directions

In this paper, we propose a novel algebraic framework for solving Boolean formulas consisting of hybrid constraints. Fourier Expansions are used to reduce Boolean satisfiability to multilinear optimization. Our study on the landscape and properties of multilinear polynomials leads to the discovery of attractive features of this reduction. Furthermore, to show the algorithmic benefits of this reduction, we design algorithms with certain theoretical guarantees. Finally, we implement our method as `FourierSAT`. The experimental results indicate that in practice, the proof-of-concept implementation of `FourierSAT` is able to handle hybrid Boolean constraints and can be a useful complement to state-of-the-art solvers on certain benchmarks by leveraging parallelization. A more mature and efficient implementation of our framework is still desirable.

We also list a few future directions in the following.

- **Complete algebraic SAT solvers.** Besides giving solutions to satisfiable formulas, we also aim to prove unsatisfiability algebraically. In other words, we hope to design a complete SAT solver based on algebraic approaches. Several algebraic proof systems, such as Hilbert's Nullstellensatz and Gröbner basis (e.g., [106–108]) can be used for giving an algebraic proof of unsatisfiability. We are interested to see if Fourier Expansions can be used in any proof systems in practice to make `FourierSAT` a complete solver.
- **Escaping local minima and saddle points.** As a local-information-based approach, `FourierSAT` suffers from getting stuck in local minima and relies heavily on random restart. We believe the idea of "sideway moves" from Local Search SAT solvers [14] is a good direction to address these issues. For example, one can use the solution of WSAT (`FourierSAT`) as the initial point of `FourierSAT` (WSAT). We also would like to consider gradient descent approaches with random noise [48], and how they perform on escaping saddle points both in theory and practice. Furthermore, how to take advantage of the searching history to search the continuous space more systematically, e.g., designing smarter restart protocols, is another interesting problem.
- **Alternative objective functions.** The Fourier Expansions of constraints with large number of variables and low solution density can be much less smooth, which is one of the main disadvantages of `FourierSAT` in practice. To refine the objective function, one possible way is to develop better weight functions, which can be static or dynamic. Due to the techniques for learning Fourier coefficients [109,43], it is also promising to use the low-degree approximation of Fourier Expansions as the objective function.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work is supported in part by NSF grants IIS-1527668, CCF-1704883, IIS-1830549, CCF-1907936, CNS-2003137, DoD MURI grant N00014-20-1-2787, and an award from the Maryland Procurement Office. We thank the anonymous reviewers for many substantial and useful comments which significantly improved this paper.

References

[1] A. Ignatiev, F. Pereira, N. Narodytska, J. Marques-Silva, A sat-based approach to learn explainable decision sets, in: D. Galmiche, S. Schulz, R. Sebastiani (Eds.), Automated Reasoning, Springer International Publishing, Cham, 2018, pp. 627–645.
[2] M.N. Velev, Efficient translation of boolean formulas to cnf in formal verification of microprocessors, in: ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No. 04EX753), 2004, pp. 310–315.
[3] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (6) (2008) 772–799, https://doi.org/10.1016/j.artint.2007.11.002, http://www.sciencedirect.com/science/article/pii/S0004370207001889.
[4] E. Zulkoski, V. Ganesh, K. Czarnecki, Mathcheck: a math assistant via a combination of computer algebra systems and sat solvers, in: A.P. Felty, A. Middeldorp (Eds.), Automated Deduction - CADE-25, Springer International Publishing, Cham, 2015, pp. 607–622.
[5] J.P. Marques-Silva, K.A. Sakallah, GRASP: a search algorithm for propositional satisfiability, IEEE Trans. Comput. 48 (5) (1999) 506–521.
[6] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (3) (1960) 201–215.
[7] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Commun. ACM 5 (7) (1962) 394–397.
[8] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: engineering an efficient sat solver, in: DAC '01, 2001, pp. 530–535, http://doi.acm.org/10.1145/378239.379017.
[9] N. Eén, N. Sörensson, An extensible SAT-solver, in: SAT, 2003, pp. 502–518.
[10] A. Biere, PicoSAT essentials, JSAT 4 (2008) 75–97.
[11] A. Biere, Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010, 2010.
[12] G. Audemard, L. Simon, Lazy clause exchange policy for parallel SAT solvers, in: SAT 2014, Cham, 2014, pp. 197–205.
[13] J.H. Liang, Machine learning for sat solvers, Ph.D. thesis, University of Waterloo, Dec. 2018.
[14] B. Selman, H. Levesque, D. Mitchell, A new method for solving hard satisfiability problems, http://dl.acm.org/citation.cfm?id=1867135.1867203, 1992.
[15] B. Selman, H. Kautz, B. Cohen, Local Search Strategies for Satisfiability Testing, Second DIMACS Implementation, Challenge 26.
[16] W. Gong, X. Zhou, A survey of SAT solver, AIP Conf. Proc. 1836 (1) (2017) 020059, https://doi.org/10.1063/1.4981999, https://aip.scitation.org/doi/pdf/10.1063/1.4981999, https://aip.scitation.org/doi/abs/10.1063/1.4981999.

[17] D. McAllester, B. Selman, H. Kautz, in: Evidence for Invariants in Local Search, 1997.
[18] A. Balint, A. Fröhlich, Improving stochastic local search for SAT with a new probability distribution, in: SAT 2010, 2010, pp. 10–15.
[19] H. Hoos, T. Stützle, Local search algorithms for sat: an empirical evaluation, J. Autom. Reason. 24 (2000) 421–481, https://doi.org/10.1023/A:1006350622830.
[20] F. Hutter, D.A.D. Tompkins, H. Hoos, Scaling and probabilistic smoothing: efficient dynamic local search for sat, in: CP, 2002.
[21] D.A.D. Tompkins, H.H. Hoos, Ubcsat: an implementation and experimentation environment for sls algorithms for sat and max-sat, in: H.H. Hoos, D.G. Mitchell (Eds.), Theory and Applications of Satisfiability Testing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 306–320.
[22] A. Balint, U. Schöning, Choosing probability distributions for stochastic local search and the role of make versus break, in: SAT 2012, 2012, pp. 16–29.
[23] Z. Lei, S. Cai, Solving (weighted) partial maxsat by dynamic local search for sat, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 1346–1352.
[24] T. Brueggemann, W. Kern, An improved deterministic local search algorithm for 3-SAT, Theor. Comput. Sci. 329 (1) (2004) 303–313, https://doi.org/10.1016/j.tcs.2004.08.002, http://www.sciencedirect.com/science/article/pii/S0304397504005146.
[25] H. Fang, W. Ruml, Complete local search for propositional satisfiability, in: AAAI, 2004.
[26] G. Audemard, L. Simon, Gunsat: A Greedy Local Search Algorithm for Unsatisfiability, 2007, pp. 2256–2261.
[27] A. Bogdanov, D. Khovratovich, C. Rechberger, Biclique cryptanalysis of the full AES, in: ASIACRYPT 2011, 2011.
[28] M.-C. Costa, D. de Werra, C. Picouleau, B. Ries, Graph coloring with cardinality constraints on the neighborhoods, Discrete Optim. 6 (4) (2009) 362–369, https://doi.org/10.1016/j.disopt.2009.04.005, http://www.sciencedirect.com/science/article/pii/S1572528609000231.
[29] I. Dinur, O. Regev, C. Smyth, The hardness of 3-uniform hypergraph coloring, Combinatorica 25 (5) (2005) 519–535, https://doi.org/10.1007/s00493-005-0032-4.
[30] J. Feldman, M.J. Wainwright, D.R. Karger, Using linear programming to decode binary linear codes, IEEE Trans. Inf. Theory 51 (3) (2005) 954–972, https://doi.org/10.1109/TIT.2004.842696.
[31] M. Gwynne, O. Kullmann, On sat representations of xor constraints, in: A.-H. Dediu, C. Martín-Vide, J.-L. Sierra-Rodríguez, B. Truthe (Eds.), Language and Automata Theory and Applications, Springer International Publishing, Cham, 2014, pp. 409–420.
[32] T. Philipp, P. Steinke, Pblib – a library for encoding pseudo-boolean constraints into cnf, in: M. Heule, S. Weaver (Eds.), Theory and Applications of Satisfiability Testing – SAT 2015, Springer International Publishing, Cham, 2015, pp. 9–16.
[33] S. Prestwich, CNF encodings, in: Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications, 2009.
[34] C.-G. Quimper, T. Walsh, Decomposing global grammar constraints, in: CP 2007, 2007, pp. 590–604.
[35] H. Kautz, D. Mcallester, B. Selman, Exploiting variable dependency in local search, in: Abstracts of the Poster Sessions of IJCAI-97, 1997.
[36] R. Martins, V. Manquinho, I. Lynce, Exploiting cardinality encodings in parallel maximum satisfiability, in: ICTAI, 2011, pp. 313–320.
[37] M. Soos, K. Nohl, C. Castelluccia, Extending SAT solvers to cryptographic problems, in: SAT, 2009, pp. 244–257.
[38] M.H. Liffiton, J.C. Maglalang, A cardinality solver: more expressive constraints for free, in: SAT 2012, 2012.
[39] H.M. Sheini, K.A. Sakallah, Pueblo: a hybrid pseudo-boolean sat solver, JSAT 2 (2006) 165–189.
[40] J. Elffers, J. Nordström, Divide and Conquer: towards faster pseudo-Boolean solving, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 1291–1299, https://doi.org/10.24963/ijcai.2018/180.
[41] S. Bayless, N. Bayless, H.H. Hoos, A.J. Hu, SAT modulo monotonic theories, in: AAAI, 2015.
[42] R. O'Donnell, Analysis of Boolean Functions, Cambridge University Press, New York, NY, USA, 2014.
[43] N. Linial, Y. Mansour, N. Nisan, Constant depth circuits, Fourier transform, and learnability, in: ASFCS, 1989.
[44] C. Wei, S. Ermon, General bounds on satisfiability thresholds for random CSPs via Fourier analysis, http://dl.acm.org/citation.cfm?id=3298023.3298143, 2017.
[45] T. Achim, A. Sabharwal, S. Ermon, Beyond parity constraints: Fourier analysis of hash functions for inference, in: ICML, 2016, pp. 2254–2262, http://proceedings.mlr.press/v48/achim16.html.
[46] Y. Xue, S. Ermon, R. Le Bras, C.P. Gomes, B. Selman, Variable elimination in the Fourier domain, arXiv:1508.04032, 2015.
[47] R. Ge, F. Huang, C. Jin, Y. Yuan, Escaping from saddle points — online stochastic gradient for tensor decomposition, arXiv:1503.02101, 2015.
[48] C. Jin, P. Netrapalli, R. Ge, S.M. Kakade, M.I. Jordan, Stochastic gradient descent escapes saddle points efficiently, arXiv:1902.04811, 2019.
[49] J.D. Lee, M. Simchowitz, M.I. Jordan, B. Recht, Gradient descent converges to minimizers, arXiv:1602.04915, 2016.
[50] Z. Zhang, Solving Hybrid Boolean Constraints by Fourier Expansions and Continuous Optimization, Jan. 2020.
[51] A. Kyrillidis, A. Shrivastava, M.Y. Vardi, Z. Zhang, FourierSAT: a Fourier expansion-based algebraic framework for solving hybrid Boolean constraints, in: AAAI, 2020.
[52] B. Wah, Y. Shang, A discrete lagrangian-based global-search method for solving satisfiability problems, J. Glob. Optim. 12 (1) (1998) 61–99, https://doi.org/10.1023/A:1008287028851.
[53] P. Morris, The breakout method for escaping from local minima, in: AAAI, 1993.
[54] H.E. Dixon, M.L. Ginsberg, E.M. Luks, A.J. Parkes, Generalizing boolean satisfiability ii: theory, J. Artif. Intell. Res. 22 (2004) 481–534, https://doi.org/10.1613/jair.1555.
[55] Q.-N. Tran, M. Vardi, Groebner bases computation in boolean rings for symbolic model checking, in: Proceedings of the IASTED International Conference on Modelling and Simulation, 2007, pp. 440–445.
[56] M. Clegg, J. Edmonds, R. Impagliazzo, Using the groebner basis algorithm to find proofs of unsatisfiability, in: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC '96, Association for Computing Machinery, New York, NY, USA, 1996, pp. 174–183.
[57] D. Ritirc, A. Biere, M. Kauers, Improving and extending the algebraic approach for verifying gate-level multipliers, in: 2018 Design, Automation Test in Europe Conference, Exhibition (DATE), 2018, pp. 1556–1561.
[58] C. Condrat, P. Kalla, A gröbner basis approach to cnf-formulae preprocessing, in: O. Grumberg, M. Huth (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 618–631.
[59] J.P. Warners, H. van Maaren, A two-phase algorithm for solving a class of hard satisfiability problems, Oper. Res. Lett. 23 (3) (1998) 81–88, https://doi.org/10.1016/S0167-6377(98)00052-2, http://www.sciencedirect.com/science/article/pii/S0167637798000522.
[60] J. Horáček, M. Kreuzer, On conversions from cnf to anf, in: Symbolic Computation and Satisfiability Checking, J. Symb. Comput. 100 (2020) 164–186, https://doi.org/10.1016/j.jsc.2019.07.023, http://www.sciencedirect.com/science/article/pii/S0747717119300896.
[61] D. Choo, M. Soos, K.M.A. Chai, K.S. Meel, Bosphorus: bridging anf and cnf solvers, arXiv:1812.04580, 2018.
[62] Jun Gu, Global optimization for satisfiability (sat) problem, IEEE Trans. Knowl. Data Eng. 6 (3) (1994) 361–381.
[63] G. Calinescu, C. Chekuri, M. Pál, J. Vondrák, Maximizing a monotone submodular function subject to a matroid constraint, SIAM J. Comput. 40 (6) (2011) 1740–1766, https://doi.org/10.1137/080733991.
[64] J.L. Walsh, A closed set of normal orthogonal functions, Am. J. Math. 45 (1) (1923) 5–24, http://www.jstor.org/stable/2387224.
[65] J. Hadamard, Resolution d'une question relative aux determinants, Bull. Sci. Math. 2 (1893) 240–246, https://ci.nii.ac.jp/naid/20000814080/en/.
[66] A. Bonami, Étude des coefficients de fourier des fonctions de $l^p(g)$, Ann. Inst. Fourier 20 (2) (1970) 335–402, http://eudml.org/doc/74019.
[67] J. Kahn, G. Kalai, N. Linial, The influence of variables on boolean functions, in: [Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science, 1988, pp. 68–80.

[68] A. Tal, Tight bounds on the fourier spectrum of ac0, in: Proceedings of the 32nd Computational Complexity Conference, CCC '17, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, DEU, 2017.
[69] G. Kalai, S. Safra, Threshold phenomena and influence, with some perspectives from mathematics, computer science, and economics, Discussion paper series, The Federmann Center for the Study of Rationality, the Hebrew University, Jerusalem, 2005, https://EconPapers.repec.org/RePEc:huj:dispap:dp398.
[70] G. Kalai, A fourier-theoretic perspective on the condorcet paradox and arrow's theorem, Adv. Appl. Math. 29 (2002) 412–426, https://doi.org/10.1016/S0196-8858(02)00023-4.
[71] T. Achim, A. Sabharwal, S. Ermon, Beyond parity constraints: Fourier analysis of hash functions for inference, in: M.F. Balcan, K.Q. Weinberger (Eds.), Proceedings of the 33rd International Conference on Machine Learning, in: Proceedings of Machine, vol. 48, Learning Research, PMLR, New York, New York, USA, 2016, pp. 2254–2262, http://proceedings.mlr.press/v48/achim16.html.
[72] C.R. Edwards, The application of the rademacher–walsh transform to boolean function classification and threshold logic synthesis, IEEE Trans. Comput. C-24 (1) (1975) 48–62.
[73] J. Moore, K. Fazel, M.A. Thornton, D.M. Miller, Boolean function matching using walsh spectral decision diagrams, in: 2006 IEEE Dallas/CAS Workshop on Design, Applications, Integration and Software, 2006, pp. 127–130.
[74] M.A. Thornton, V.S.S. Nair, Efficient calculation of spectral coefficients and their applications, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 14 (11) (1995) 1328–1341.
[75] R.E. Bryant, Binary decision diagrams and beyond: enabling technologies for formal verification, in: Proceedings of IEEE International Conference on Computer Aided Design (ICCAD), 1995, pp. 236–243.
[76] J. Moore, K. Fazel, M.A. Thornton, D.M. Miller, Boolean function matching using walsh spectral decision diagrams, in: 2006 IEEE Dallas/CAS Workshop on Design, Applications, Integration and Software, 2006, pp. 127–130.
[77] M. Thornton, R. Drechsler, D. Miller, Spectral techniques in vlsi cad, https://doi.org/10.1007/978-1-4615-1425-1, 2001.
[78] P. Savický, On the bent boolean functions that are symmetric, Eur. J. Comb. 15 (4) (1994) 407–410, https://doi.org/10.1006/eujc.1994.1044.
[79] E. Dawson, C.-K. Wu, On the linear structure of symmetric boolean functions, Australas. J. Comb. 16 (1997) 239–243.
[80] MA317-UBC, Lecture 1, harmonic functions, http://www.math.ubc.ca/~rolfsen/ma317/317notes.pdf.
[81] M. Thornton, V. Nair, Efficient spectral coefficient calculation using circuit output probabilities, Digit. Signal Process. 4 (4) (1994) 245–254, https://doi.org/10.1006/dspr.1994.1024, http://www.sciencedirect.com/science/article/pii/S1051200484710244.
[82] A. Kyrillidis, M.Y. Vardi, Z. Zhang, On continuous local BDD-based search for hybrid SAT solving, arXiv:2012.07983, 2020.
[83] Y.N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, in: Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 2933–2941.
[84] A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, Y. LeCun, The loss surfaces of multilayer networks, arXiv:1412.0233, 2014.
[85] Y. Nesterov, Introductory Lectures on Convex Optimization: A Basic Course, 2014.
[86] N. He, Lecture Notes in IE, vol. 598, Big Data Optimization, 2016.
[87] Y. Nesterov, Introductory Lectures on Convex Optimization: A Basic Course, 1st edition, Springer Publishing Company, Incorporated, 2014.
[88] S. Bubeck, Convex optimization: algorithms and complexity, arXiv:1405.4980, 2014.
[89] D. Kraft, A Software Package for Sequential Quadratic Programming, Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt.
[90] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open Source Scientific Tools for Python, 2001.
[91] S. Cai, K. Su, C. Luo, Improving walksat for random k-satisfiability problem with $k > 3$, in: AAAI, 2013.
[92] A. Ignatiev, A. Morgado, J. Marques-Silva, PySAT: a python toolkit for prototyping with SAT oracles, in: SAT, 2018, pp. 428–437.
[93] M. Sakai, H. Nabeshima, Construction of an robdd for a pb-constraint in band form and related techniques for pb-solvers, IEICE Trans. Inf. Syst. E98.D (6) (2015) 1121–1127.
[94] R. Martins, V. Manquinho, I. Lynce Open-wbo, A modular maxsat solver, in: C. Sinz, U. Egly (Eds.), Theory and Applications of Satisfiability Testing – SAT 2014, Springer International Publishing, Cham, 2014, pp. 438–445.
[95] J. Berg, E. Demirović, P.J. Stuckey, Core-boosted linear search for incomplete maxsat, in: L.-M. Rousseau, K. Stergiou (Eds.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer International Publishing, Cham, 2019, pp. 39–56.
[96] C. Sinz, Towards an optimal CNF encoding of Boolean cardinality constraints, in: CP 2005, 2005, pp. 827–831.
[97] K.E. Batcher, Sorting networks and their applications, in: Spring Joint Computer Conference, AFIPS '68, Spring, 1968, http://doi.acm.org/10.1145/1468075.1468121.
[98] O. Bailleux, Y. Boufkhad, Efficient CNF encoding of Boolean cardinality constraints, in: CP, 2003.
[99] N. Eén, N. Sörensson, Translating pseudo-Boolean constraints into SAT, JSAT 2 (2006) 1–26.
[100] C.M. Li, Integrating equivalency reasoning into Davis-Putnam procedure, in: AAAI, AAAI Press, 2000, pp. 291–296, http://dl.acm.org/citation.cfm?id=647288.760210.
[101] L. Gurobi Optimization, Gurobi Optimizer Reference Manual, http://www.gurobi.com, 2019.
[102] J.M. Crawford, M.J. Kearns, The Minimal Disagreement Parity Problem as a Hard Satisfiability Problem, 1994.
[103] H. Hoos, T. Stützle, in: SATLIB: an Online Resource for Research on SAT, 04 2000.
[104] J.M. Dudek, K.S. Meel, M.Y. Vardi, Combining the k-CNF and XOR phase-transitions, in: IJCAI, 2016.
[105] Y. Pote, S. Joshi, K.S. Meel, Phase transition behavior of cardinality and XOR constraints, in: IJCAI-19, 2019.
[106] D.A. Cox, J. Little, D. O'shea, Ideals, Varieties and Algorithms, 1994.
[107] J.A.D. Loera, J. Lee, P.N. Malkin, S. Margulies, Computing infeasibility certificates for combinatorial problems through Hilbert's Nullstellensatz, J. Symb. Comput. 46 (2011) 1260–1283.
[108] Romero Barbosa Julian, in: Applied Hilbert's Nullstellensatz for Combinatorial Problems, 2016, http://hdl.handle.net/10012/10897.
[109] J. Bruck, R. Smolensky, Polynomial threshold functions, AC0 functions, and spectral norms, SIAM J. Comput. 21 (1) (1992) 33–42, https://doi.org/10.1137/0221003.