# FourierSAT: A Fourier Expansion-Based Algebraic Framework for Solving Hybrid Boolean Constraints *

**Anastasios Kyrillidis, Anshumali Shrivastava, Moshe Y. Vardi, Zhiwei Zhang** †
Rice University, Houston, TX, USA
{anastasios, anshumali, vardi, zhiwei}@rice.edu

December 4, 2019

**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. In addition, previous work indicated that for specific classes of benchmarks, the running time of extant SAT solvers depends heavily on properties of the formula and details of encoding, instead of the scale of the benchmarks, which adds uncertainty to expectations of running time.

To address the issues above, we design FourierSAT [3], an incomplete SAT solver based on Fourier analysis 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. Empirical results demonstrate that FourierSAT is more robust than other solvers on certain classes of benchmarks.

***Keywords*** SAT solving · Multilinear optimization · Fourier analysis on Boolean functions

## 1 Introduction

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.

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. 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 the Conflict-Driven Clause Learning (CDCL) algorithm [1], an evolution of the backtracking Davis-Putnam-Logemann-Loveland (DPLL) algorithm [2, 3]. The main techniques used in CDCL solvers include clause-based learning of conflicts, random restarts, heuristic variable selection, and

*The author list has been sorted alphabetically by last name; this should not be used to determine the extent of authors' contributions.

†Corresponding author: Zhiwei Zhang.

[3]The tool is available at https://github.com/vardigroup/FourierSAT

effective constraint-propagation data structures [4]. Examples of highly efficient complete SAT solvers include MiniSat [5], BerkMin [6], PicoSAT [7], Lingeling [8], Glusose [9] and MapleSAT [10]. 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) [11] and random walk GSAT (WSAT) [12]. 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 [4]. On top of these basic algorithms, several heuristics for variable selection have been proposed, such as NSAT [13], Sparrow [14], and ProbSAT [15]. While practical local search SAT 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. Local search algorithms can also have nice theoretical properties [16].

Non-CNF constraints are playing important roles in theoretical computer science and other engineering areas, *e.g.*, XOR constraints in cryptography [17] as well as cardinality constraints (CARD) and Not-all-equal (NAE) constraints in discrete optimization [18, 19]. The combination of different types of constraints enhances the expressive power of Boolean formulas; e.g., CARD-XOR is necessary and sufficient for maximum likelihood decoding (MLD) [20], 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. However, different encodings differ from the size, the ability to detect inconsistencies by unit propagation (arc consistency) and solution density [21]. It is generally observed that the running time of SAT solvers relies heavily on the detail of encodings. E.g., CDCL solvers benefit from arc-consistency [22], while local search solvers prefer short chains of variable dependencies [23]. Finding a best encoding for a solver usually requires considerable testing and comparison [24].

Another way is to extend the existing SAT solvers to adapt to non-CNF clauses. Work on this line includes Cryptominisat [25] for CNF-XOR, MiniCARD [26] for CNF-CARD, Pueblo [27] and RoundingSAT [28] for CNF-Pseudo Boolean and MonoSAT [29] for handling CNF-graph properties formulas. Such specialized extensions, however, often require different techniques for different types of constraints. Meanwhile, general ideas for solving hybrid constraints uniformly are still lacking.

**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 method is the Walsh-Fourier transform (Fourier transform, for short) on Boolean functions [30]. By transforming Boolean functions into "nice" polynomials, numerous properties can be analyzed mathematically. Besides the success of Fourier transform in theoretical computer science [31, 32], recently, this tool has also found surprising uses in algorithm design [33, 34]. In [35], the authors used a polynomial representation to design a SAT solver. However, their solver was still DPLL-based and mathematical properties of polynomials were not fully exploited. More algorithmic uses of this technique are waiting to be discovered.

To our best knowledge, this paper will be the first algorithmic work to study Fourier expansions of Boolean functions in the real domain instead of only Boolean domain. After this relaxation, we find Fourier expansions well-behaved in the real domain. 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, 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 known, our solver can be extended 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 existing 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 [36]. Although the study of [37, 38] indicate 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 class of hybrid formulas, FourierSAT is more robust compared to existing tools.

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.

| Clause | Fourier Expansion |
|---|---|
| $x_1 \vee x_2$ | $-\frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_1x_2$ |
| $D^{\geq 2}(x_1, x_2, x_3)$ | $\frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_3 - \frac{1}{2}x_1x_2x_3$ |
| $x_1 \oplus x_2 \oplus x_3$ | $x_1x_2x_3$ |
| $\texttt{NAE}(x_1, x_2, x_3)$ | $-\frac{1}{2} + \frac{1}{2}x_1x_2 + \frac{1}{2}x_2x_3 + \frac{1}{2}x_1x_3$ |

Table 1: Examples of Fourier Expansion on different clauses

## 2 Notations and Preliminaries

### Boolean Formulas and Clauses

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 $a \in \{\texttt{True}, \texttt{False}\}^n$ is called an assignment and $f(a)$ denotes the value of $f$ on $a$. A literal is either a variable $x_i$ or its negation $\neg x_i$. A formula $f = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ is the conjunction of $m$ Boolean functions, where each $c_i$ is called a clause and belongs to a type from the list below:

- `CNF`: A `CNF` clause is a disjunction of elements in a literal set, which is satisfied when at least one literal is true. E.g., $(x_1 \vee x_2 \vee x_3)$.
- `CARD`: Given a set $L$ of variables and an integer $k \in [n]$, a cardinality constraint $D^{\geq k}(L)$ (resp. $D^{\leq k}(L)$) requires the number of `True` variables to be at least (resp. most) $k$. E.g., $D^{\geq 2}(\{x_1, x_2, x_3\})$: $x_1 + x_2 + x_3 \geq 2$.
- `XOR`: An XOR clause indicates the parity of the number of variables assigned to be `True`, which can be computed by addition on $\texttt{GF}(2)$. 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. E.g., $\texttt{NAE}(1, 1, 1, 1, 0) = 1$; $\texttt{NAE}(0, 0, 0, 0, 0) = 0$

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

### Fourier Expansion of a Boolean Function

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 $a \in \{\pm 1\}^n$.

Fourier expansion is a method for transforming a Boolean function into a multilinear polynomial. The following theorem shows that any function defined on a Boolean hyper-cube has an equivalent Fourier representation.

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

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

*where $\widehat{f}(S)$'s$\in \mathbb{R}$ are called the Fourier coefficients, given $S$, and computed as:*

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

*where $x \sim \{\pm 1\}^n$ indicates $x$ is chosen uniformly from $\{\pm 1\}^n$. The polynomial is referred as the **Fourier expansion** of $f$.*

Table 1 shows some examples of Fourier expansions.

## 3 A Reduction from SAT to Continuous Optimization

In this section, we reduce finding a solution of a Boolean formula to finding a minimum of a continuous multivariate polynomial, based on Fourier expansion. Due to the space limit, we delay most of the proofs to the supplemental material.

### A Reduction to a Minimization Problem

Our basic idea is to do continuous optimization on Fourier expansions of formulas. However, in general, computing the Fourier coefficients of a Boolean function is nontrivial and often harder than SAT itself (for an example, 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 store and evaluate.

**Proposition 1.** *Computing the Fourier Expansion of a pseudo-Boolean constraint (a generalization of a cardinality constraint, e.g.,* $3x_1 + 2x_2 + 7(\neg x_3) \geq 0$*,* $x \in \{\pm 1\}^3$*) is #P-hard.*

Instead of computing Fourier coefficients of a monolithic logic formula, we take advantage of factoring, constructing a polynomial for a formula by the Fourier expansions of its clauses. Excitingly, Fourier expansions of many types of clauses with great interest can be computed easily. Details can be found in the proof of Proposition 2.

**Proposition 2.** *Fourier expansions of* `CNF`*,* `XOR`*,* `NAE` *clauses and cardinality constraints have closed form representations.*

For a formula $f$ with clause set $C(f)$, we define the *objective function* associated with $f$, denoted by $F_f$, by the sum of Fourier expansions of $f$'s clauses, i.e.,

$$F_f = \sum_{c \in C(f)} \texttt{FE}_c$$

where $\texttt{FE}_c$ denotes the Fourier expansion of clause $c$.

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

Let us provide an example to illustrate the above ideas.

**Example 1.** *Suppose* $f = (x_1 \vee x_2) \wedge (x_3) \wedge (x_2 \vee \neg x_4)$*. Then,* $C(f) = \{x_1 \vee x_2,\ x_3,\ x_2 \vee \neg x_4\}$ *and*

$$\begin{aligned} F_f &= \left(-\frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_1x_2\right) + x_3 + \left(-\frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_4 - \frac{1}{2}x_2x_4\right) \\ &= -1 + \tfrac{1}{2}x_1 + x_2 + x_3 - \tfrac{1}{2}x_4 + \tfrac{1}{2}x_1x_2 - \tfrac{1}{2}x_2x_4. \end{aligned}$$

Notice that the objective function is nothing special but a polynomial. Therefore we relax the domain from discrete to continuous. An assignment is now defined as a real vector $a \in [-1, 1]^n$. The reduction is formalized in Theorem 2.

**Theorem 2.** *(Reduction)* $f$ *is satisfiable if and only if*

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

Theorem 2 reduces SAT to a multivariate minimization problem over $[-1, 1]^n$. In the rest of this subsection, we will provide proof sketch of Theorem 2.

**Definition 1.** *(Constant). A clause is constant if it is equivalent to either* `True` *or* `False`*. The Fourier expansion of a clause is constant if it always equals to* $-1$ *or* $1$*.*

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 clause and* $a$ *be an assignment. Then:*

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

In order to formalize the procedure of converting a fractional solution to a Boolean assignment, we define the concept "feasibility".

**Definition 2.** *(Partially assigned function) Suppose* $[n]$ *is partitioned into two sets,* $J$ *and* $[n] - J$*. Let* $z \in [-1, 1]^{|J|}$ *be a real vector, we write* $F_{J \leftarrow z}$ *for the partially assigned function of* $F$ *given by fixing the coordinates in* $J$ *to be the values* $z$ *in* $F$*.*

**Definition 3.** *(Feasible Solution). For an objective function $F$ of a Boolean formula and an assignment $a \in [-1,1]^n$, let $I = \{i \mid a_i \in \{\pm 1\}\}$. $a$ is a **feasible solution** of $F$ if $F_{I \leftarrow a_I}$ is constant, where $a_I$ is the projected vector of $a$ at coordinates in $I$. We say $a$ is* feasible *if $F$ is clear in context.*

**Example 2.** *Let $f = x_1 \wedge x_2$; then*

$$F_f = \tfrac{1}{2} + \tfrac{1}{2}x_1 + \tfrac{1}{2}x_2 - \tfrac{1}{2}x_1x_2.$$

*$a = (1, -0.3)$ is a feasible solution because $I = \{1\}$ and $F_{I \leftarrow a_I} = F_{1 \leftarrow 1} = \frac{1}{2} + \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_2 = 1$ is constant not depending on $x_2$.*

By Definition 3 if we find a feasible solution $a^\star$ of the objective function, we can adjust it into a Boolean assignment $b^\star$ by modifying values of $a^\star$ in $(-1,1)$ to $\{\pm 1\}$ in $b^\star$ arbitrarily. By feasibility of $a^\star$, $F_f(a^\star) = F_f(b^\star)$.

**Lemma 2.** *Let $c$ be a non-constant clause and $a$ be an assignment. If $\texttt{FE}_c(a) = -1$, then $a$ is a feasible solution 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 $\texttt{FE}_{c\{I \leftarrow a_I\}}$,*

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

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

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

□

Now we are ready to prove Theorem 2.

**Proof of Theorem 2.** *Note that by the second argument in Lemma 1, we have $F_f(a) \geq -m$ for all $a \in [-1,1]^n$.*

- *"⇒": Suppose $f$ is satisfiable and $\phi \in \{\pm 1\}^n$ is one of its solution. $\phi$ is also a solution of every clause of $f$. Thus for every $c \in C(f)$, $\texttt{FE}_c(\phi) = -1$. Therefore $F_f(\phi) = -m$ and $\min_{x \in [-1,1]^n} F_f(x) = -m$.*
- *"⇐": Suppose $\min_{x \in [-1,1]^n} F_f(x) = -m$. Thus $\exists\, a^\star$ such that $F_f(a^\star) = -m$. By the second argument in Lemma 1, we have $\texttt{FE}_c(a^\star) = -1$ for every $c \in C(f)$. By Lemma 2 $a^\star$ is a feasible solution of $FE_c$ for every $c \in C(f)$. Thus $a^\star$ is also a feasible solution of $F_f$. Therefore, by feasibility, we can get a Boolean assignment which satisfies all the clauses by arbitrarily rounding all the fractional coordinates.* □

### Local Landscape of Multilinear Polynomials

In this subsection we characterize the local landscape of multilinear polynomials. We also show that, in our case, local minima are meaningful and useful.

First, we formally define local minimum for constrained problems in Definition 4.

**Definition 4.** *(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 $\delta$ as $\mathcal{N}_\delta(a) = \{x \mid \|a - x\|_2^2 \leq \delta\}$. Given a set $\Delta$, we say an assignment $a$ is a **local minimum of $F$ in $\Delta$** if*

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

For a multivariate function, 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. Therefore, a saddle point is not a local minimum in an unconstrained problem.

**Lemma 3.** *Every critical point (where all the first derivatives are zero) of a non-constant multilinear polynomial is a saddle point.*

Lemma 3 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. Lemma 4 uses feasibility again to formalize this intuition.

**Lemma 4.** *If $a^\star \in [-1,1]^n$ is a local minimum of $F$ in $[-1,1]^n$, then $a^\star$ is feasible.*

Lemma 4 indicates that every local minimum $a^\star$ in $[-1,1]^n$ is meaningful in the sense that it can be easily converted into a Boolean assignment $b^\star$. Moreover, since $b^\star$ is a Boolean assignment, it is easy to see that the number of clauses satisfied by $b^\star$ is $\frac{m-F(a^\star)}{2}$, which is a special case of Theorem 3 in the next section. Thus if a global minimum is too hard to get, our method is still useful for some problems, e.g., MaxSAT.

In most multilinear polynomials generated from Boolean formulas in practice, saddle points rarely exist. However, there exist "pathological" polynomials that have unaccountably infinite saddle points, as Example 3 shows.

**Example 3.** *Let $F(x) = x_1x_2x_3x_4$. Then every point which has 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.*

## 4 Why Continuous?

Before introducing our algorithm for minimizing multilinear polynomials, we would like to illustrate why doing continuous optimization might be better than discrete local search.

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

Theorem 3 formalizes the intuition. It indicates that when we decrease the polynomial, we are in fact increasing the expectation of the number of satisfied clauses, after a randomized rounding.

**Definition 5.** *(Randomized Rounding) The randomized rounding function, denoted by $R : [-1,1]^n \to \{\pm 1\}^n$ is defined by:*

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

where $i \in [n]$ and $a \in [-1,1]^n$. Note that the closer $a_i$ is to $-1$, the more likely $R(a)_i$ will be $-1$ and vise versa.

**Theorem 3.** *Let $f$ be a formula with $m$ clauses and $F$ be the multilinear polynomial associated with $f$. For $a \in [-1,1]^n$, let $R(a) \in \{\pm 1\}^n$ be the vector given by rounding $a$. Let $m_{\mathtt{SAT}}(R(a))$ be the number of satisfied clauses by $R(a)$, then*

$$\mathop{\mathbb{E}}_{R(a)}(m_{\mathtt{SAT}}(R(a)) = \frac{m-F(a)}{2}.$$

In particular, if $F(a) = -m$ then $\mathop{\mathbb{E}}_{R(a)}(m_{\mathtt{SAT}}(R(a)) = m$. Since $m_{\mathtt{SAT}}(b) \le m$ for any $b \in \{\pm 1\}^n$, $m_{\mathtt{SAT}}(R(a)) = m$ and $R(a)$ is a solution of $f$.

**Proof.** *We first prove that for the Fourier Expansion $p$ of a clause $c$, the probabilistic of $c$ is satisfied by $R(a)$, i.e., $\mathbb{P}[p(R(a)) = -1]$ is $\frac{1}{2} - \frac{p(a)}{2}$. Then by the linearity of expectation, Theorem 3 follows directly.*

*We prove by induction on the number of variables $n$.*

`Basis step`: Let $n = 1$. $p$ is either constant or $x_1$, or $-x_1$. It's easy to verify the statement holds.

`Inductive step`: Suppose $n \ge 2$. Then, $p(R(a))$ can be expanded as :

$$p(R(a)) = \frac{1-R(a)_n}{2} \cdot p_{n\leftarrow(-1)}(R(a)_{[n-1]}) + \frac{1+R(a)_n}{2} \cdot p_{n\leftarrow 1}(R(a)_{[n-1]})$$

Note that the value of $R(a)_n$ and $R(a)_{[n-1]}$ are independent, thus

$$\begin{aligned} &\mathbb{P}[p(R(a)) = -1] \\ =&\mathbb{P}[R(a)_n = -1] \cdot \mathbb{P}[p_{n\leftarrow(-1)}(R(a)_{[n-1]}) = -1] + \mathbb{P}[R(a)_n = 1] \cdot \mathbb{P}[p_{n\leftarrow 1}(R(a)_{[n-1]}) = -1] \\ =&\frac{1-a_n}{2} \cdot \frac{1-p_{n\leftarrow -1}(a_{[n-1]})}{2} + \frac{1+a_n}{2} \cdot \frac{1-p_{n\leftarrow 1}(a_{[n-1]})}{2} \text{ (by I.H.)} \\ =&\frac{1}{2} - \frac{p(a)}{2} \end{aligned}$$

□

Research showed that local search SAT solvers, such as GSAT, spend most of the time on the so-called "sideway" moves [11]. Sideway moves are moves that do not increase or decrease the total number of unsatisfied clauses. 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 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`'s, local search solvers need to flip at least half of all bits to make progress. In other words, local search solvers will encounter a neighbourhood with exponential size where their 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.

## 5 A Gradient Descend-Based Algorithm for Minimizing Multilinear Polynomials

In this section, we propose an algorithm for minimizing multilinear polynomials associated with Boolean formulas. We aim to show promising theoretical guarantees that gradient-based algorithms can enjoy. In practice, there are elaborate packages available and we used them in our experiments.

Since the objective function is constrained, continuous and differentiable, projected gradient descent (PGD) [39] is a candidate for solving our optimization problem. As a non-convex problem, the initialization plays a key role on the result. We use random initialization for each iteration and return the best result within a time limit.

**Efficient Evaluation of 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 $F(a)$ for $a \in [-1,1]^n$. Instead, we leverage the symmetry of clauses to compute the value of Fourier expansion of each clause $c$, denoted as $\texttt{FE}_c(a)$, separately. By this trick, we are able to evaluate the objective function in $O(\sum_{c\in C(f)} k_c^2)$ time (in worst case $O(n^2m)$), where $k_c$ is the length of clause $c$. By this trick, we bypass considering data structures that store the multilinear polynomials.

First note that for every negative literal, say $\neg x_i$, we can convert it to positive by flipping the sign of $a_i$, since applying a negation on formulas is equivalent to adding a minus sign to Fourier expansions (assume each variable appears at most once in a clause $c$).

Now suppose $c$ is a clause from $\{\texttt{CNF}, \texttt{CARD}, \texttt{XOR}, \texttt{NAE}\}$ with no negative literals. Then $c$ is symmetric, which means its Fourier coefficient at $S$ only depends on $|S|$. Thus the set of Fourier coefficients can be denoted as $\texttt{Coef}(\texttt{FE}_c) = (\kappa(\emptyset), \kappa([1]), \kappa([2]), \ldots, \kappa([k_c]))$. Let $s(a) = \left(1, \sum_{i=1}^{k_c} a_i, \sum_{i<j} a_i a_j, \ldots, \prod_{i=1}^{k_c} a_i\right)$. One can easily verify that $\texttt{FE}_c(a) = \texttt{Coef}(\texttt{FE}_c) \cdot s(a)$, where "$\cdot$" represents the inner product of two vectors.

$s(a)$ can be obtained by computing the coefficients of $t^0, t^1, \cdots, t^{k_c}$ in the expansion of the following polynomial.

$$\prod_{i=1}^{k_c}(a_i + t) = t^{k_c} + \left(\sum_{i=1}^{k_c} a_i\right) \cdot t^{k_c-1} + \cdots + \prod_{i=1}^{k_c} a_i \cdot t^0$$

The coefficients of the polynomial above can be computed in $O({k_c}^2)$ time, given $a \in [-1,1]^{k_c}$. Thus we can evaluate the objective function in $O(\sum_{c\in C(f)} k_c^2)$.

Since the gradient and Hessian of a multilinear polynomial are still multilinear [30], we are able to calculate them analytically by the method above. In experiments, we observed feeding gradients to the optimizer significantly accelerated the minimization process.

**Our PGD-Based Multilinear Optimization Algorithm.** 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:

$$x'_{t+1} = x_t - \eta \cdot \nabla F(x_t), \quad x_{t+1} = x'_{t+1},$$

where $\eta > 0$ is a step size.

For a constrained domain other than $\mathbb{R}^n$, $x'_{t+1}$ may be outside of the domain. In PGD, we choose the point nearest to $x'_{t+1}$ in $[-1,1]^n$ as $x_{t+1}$ [40], i.e., the Euclidean projection of $x'_{t+1}$ onto the set $[-1,1]^n$, denoted as $\Pi_{[-1,1]^n}(x'_{t+1})$.

**Algorithm 1:** PGD for multilinear $F$

**Input** : Polynomial $F$, $\eta > 0$, $\varepsilon > 0$.
**Output** : Approximately minimizer $\widehat{x}$.

**for** $j = 1, \ldots, J$ **do**
  $x_0 \sim \mathcal{U}[-1,1]^n$
  **for** $t = 0, \ldots$ **do**
    $G(x_t) = \frac{1}{\eta}\left(x_t - \Pi_{[-1,1]^n}\left(x_t - \eta \nabla F(x_t)\right)\right)$
    **if** $\|G(x_t)\|_2 > 0$ **then**
      $x_{t+1} = x_t - \eta \cdot G(x_t)$
    **else**
      **if** $x_t$ *not feasible* **then**
        $x_{t+1} = \texttt{DecInnerSaddle}(F,\ x_t,\ \eta)$
      **else**
        $I = \{i \mid (x_t)_i \in \{-1, 1\}\}$
        **if** $\nabla F(x_t)_i \neq 0,\ \forall i \in I$ **then**
          `Break` // `Prop. 5`
        **else**
          $(\texttt{LocalMinFlag},\ x_{t+1}) = \texttt{useHessian}(F, x_t)$
          **if** *LocalMinFlag* $=$ *True or Unknown* **then**
            `Break`
  **Until convergence**
**return** $x_j$ with the lowest $F(x)$ after $J$ iterators

**Definition 6.** *The Euclidean projection of a point $y$, onto a set $\Delta$, denoted by $\Pi_\Delta(y)$, is defined as*

$$\Pi_\Delta(y) = \arg\min_{x \in \Delta} \tfrac{1}{2}\|x - y\|_2^2.$$

In our case $\Delta \equiv [-1,1]^n$; computing such a projection is almost free, as shown in Proposition 3.

**Proposition 3.**

$$\Pi_{[-1,1]^n}(y)_i = \begin{cases} y_i, & \text{if } y_i \in [-1,1], \\ \textit{sgn}(y_i), & \text{otherwise.} \end{cases}$$

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

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

where $G(x) = \frac{1}{\eta}\left(x_t - \Pi_{[-1,1]^n}\left(x_t - \eta \nabla F(x_t)\right)\right)$ is regarded as the *gradient mapping*.

In Algorithm 1, we start at a uniformly random point in $[-1,1]^n$. When gradient mapping $G(\cdot)$ is large, we follow it to decrease the function value. Otherwise, it means 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 iterator. Else, we still try to escape the saddle point by additional methods.

In Theorem 4, we show that Algorithm 1 is guaranteed to converge to a point where the projected mapping is small, $\|G(x_t)\|_2 = 0$,[4] and depends polynomially on $n$, $m$ and tolerance $\varepsilon$. In the proof, we used techniques from [37] and [40].

**Theorem 4.** *(Convergence speed) With the step size $\eta = \frac{1}{nm}$, Algorithm 1 converges to a $\epsilon$-projected-critical point (where $\|G(x)\|_2 < \epsilon$) in $O(\frac{nm^2}{\epsilon^2})$ iterators in the worst case.*

The proof of Theorem 4 relies on the fact that multilinear polynomials on $[-1,1]^n$ are relatively smooth as Proposition 4 indicates.

**Proposition 4.** *(Lipschitz) Let $F : [-1,1]^n \to [-m,m]$ be a multilinear polynomial. Then, for every $x, y \in [-1,1]^n$,*

1. $|F(x) - F(y)| \leq m\sqrt{n} \cdot \|x - y\|_2$. *I.e., $F$ is $(m\sqrt{n})$-Lipschitz continuous.*

[4] In practice, a stopping criterion to use is $\|G(x_t)\|_2 < \varepsilon$ for a small accuracy level $\varepsilon > 0$.

2. $\|\nabla F(x) - \nabla F(y)\|_2 \leq mn \cdot \|x - y\|_2$. *I.e., $F$ has $(mn)$-Lipschitz continuous gradients.*

3. $\|\nabla^2 F(x) - \nabla^2 F(y)\|_2 \leq \left(mn^{\frac{3}{2}}\right) \cdot \|x - y\|_2$. *I.e., $F$ has $\left(mn^{\frac{3}{2}}\right)$-Lipschitz Hessians.*

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

Proposition 5 shows that in Algorithm 1 we can identify local minima by gradient information.

**Proposition 5.** *When algorithm 1 reaches line 13, $x$ is a local minimum. i.e, if $x$ is feasible, $G(x) = 0$ and $\nabla F(x)_i \neq 0$ for all $i \in I$, then $x$ is a local minimum in $[-1, 1]^n$ of $F$.*

We design `DecInnerSaddle` and `useHessian` to escape saddle points. `DecInnerSaddle` uses the idea from Lemma 3 to give a negative direction, as long as the point is not feasible. `useHessian` leverages second order derivatives to give a negative direction, or identifies a local minimum when the first order derivatives are too small. Due to space limit, we leave subprocedures `DecInnerSaddle` and `useHessian`, as well as their properties, in the Appendix.

### Weighted Case

The weighted version of the objective function is defined as

$$F_f = \sum_{c \in C(f)} w_f(c) \cdot \mathtt{FE}_c$$

where $w_f : C(f) \to \mathbb{R}$ is the weight function. It is easy to verify that the weighted version of Theorem 2 and Lemma 4 still hold. The weighted case is useful in two cases:

- **Vanishing Gradient.** When a clause contains too many of variables (e.g., a global cardinality constraint), we observed that the gradient given by this clause on a single variable becomes negligible at most points. By assigning large weights to long clauses, gradient varnishing can be alleviated.
- **Weighted MAX-SAT.** By returning a local minimum, our tool can be used to solve weighted MAX-SAT problems.

## 6 Experimental Results

In this section we compare our tool, `FourierSAT` with other SAT solvers on random and realistic benchmarks. The objective of our experimental evaluation is to answer these three research questions:
**RQ1.** How does `FourierSAT` compare to local search SAT solvers with cardinality constraints and `XOR`s encoded in `CNF`?
**RQ2.** How does `FourierSAT` compare to specialized `CARD-CNF` solvers with respect to handling cardinality constraints?
**RQ3.** How does `FourierSAT` compare to CDCL SAT solvers with cardinality constraints/`XOR`s encoded in `CNF`?

### Experimental Setup

We choose `SLSQP` [41], implemented in `scipy` [42] as our optimization oracle, after comparing available algorithms for non-convex, constrained optimization.

Since random restart is used, different iterators are independent. Thus, we parallelized `FourierSAT` to take advantage of multicore computation resources. Each experiment was run on an exclusive node in a homogeneous Linux cluster. These nodes contain 24-processor cores at 2.63 GHz each with 1 GB of RAM per node. The time cap for each job is 1 minute.

We compare our method with the following SAT solvers:

- Cryptominisat [25] (CMS), an advanced SAT solver supporting `CNF+XOR` clauses by enabling Gauss Elimination.
- WalkSAT [12], an efficient local search SAT solver. We used a fast C implementation due to [43].
- MiniCARD [26], a MiniSAT-based `CARD-CNF` solver. It has been implemented in `pysat` [44].
- MonoSAT [29], a SAT Modulo Theory solver which supports a large set of graph predicates.

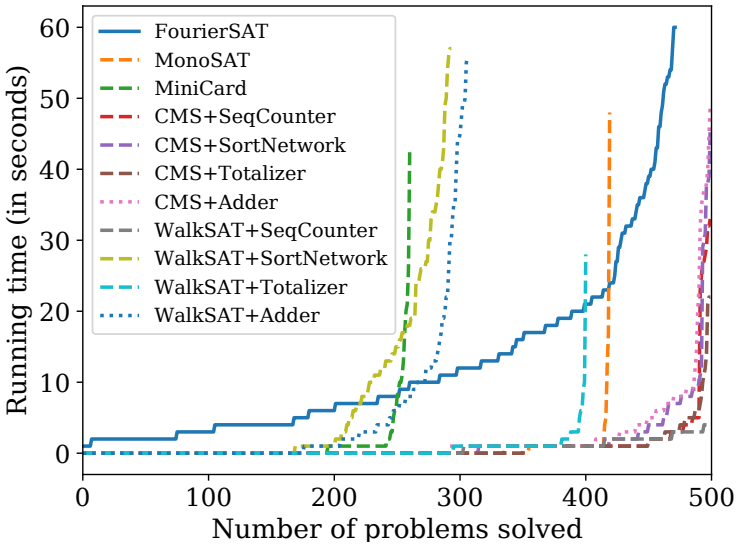


Figure 1: Results on 500 vertex covering problems

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 [45], Sorting Network [46], Totalizer [47] and Adder [48]. For solvers that do not support XORs, we used a linear encoding due to [49] to decompose them into 3-CNF. To address vanishing gradient, the weight of each clause equals to its length.

### Benchmarks

We generated hybrid Boolean constraints using natural encodings of the following benchmark problems.

**Benchmark 1: Approximate minimum vertex covering.** Minimum vertex covering is a well-known NP-optimization problem (NPO) problem. For each $n \in \{50, 100, 150, 200, 250\}$, we generated 100 random cubic graphs. For each graph, `Gurobi` [50] was used to compute the minimum vertex cover, denoted by `Opt`. Then we added one cardinality constraint, $D^{\leq k}(X)$ 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. Technically, in this task one needs to find a solution of an XOR system while tolerating some constraints to be violated. Suppose there are $n$ candidate variables and $m$ I/O samples (XOR constraints). 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 for SAT solvers as well, especially for local search solvers [51].

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 clauses where we allow at most $eM$ clauses to be violated. For WalkSAT and CMS, we used the encoding due to [52].

**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 clauses, $sn$ XOR clauses with length $ln$ and 1 global cardinality constraint $D^{\leq k}(X)$. Those parameters are chosen to guarantee that all problems are satisfiable [53, 54].

### Results

The experimental results of three benchmarks above are shown in Figure 1, 2 and 3 respectively.

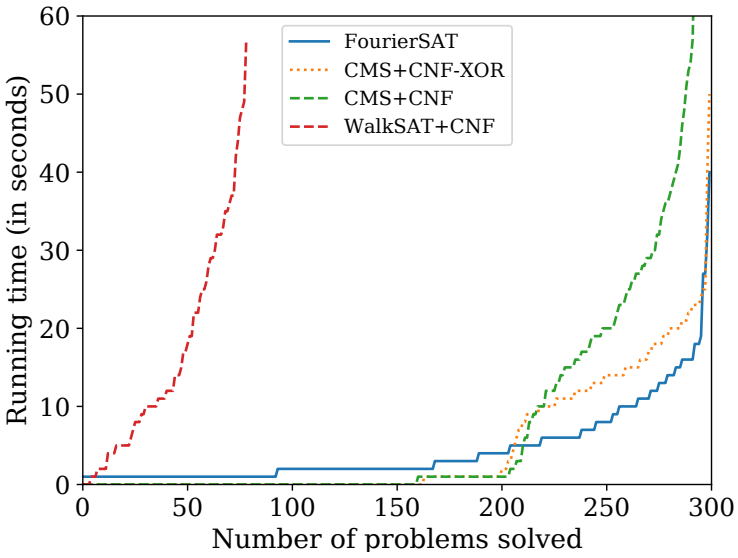


Figure 2: Results on 300 parity learning with error problems

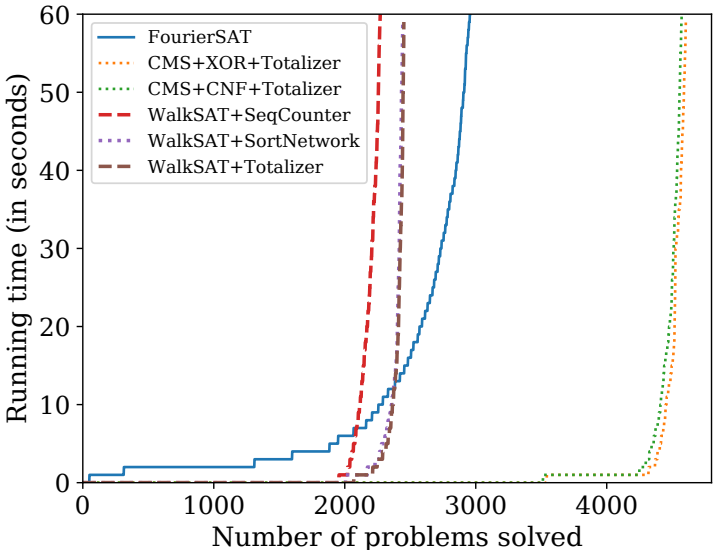


Figure 3: Results on 4800 random `CNF-XOR-CARD` problems

**RQ1.** Figure 1 shows `FourierSAT` solved more problems than WalkSAT did with all the cardinality encodings except SeqCounter. For parity learning problem, WalkSAT with `CNF` encoding only solved 79 problems, while `FourierSAT` was able to solved all 300 problems. For the random hybrid constraints benchmarks shown in Figure 3, `FourierSAT` solved 2957 problems while WalkSAT with SeqCounter, SortNetwork and Totalizer solved 2272, 2444 and 2452 problems respectively. In our experiment, `FourierSAT` is more robust than WalkSAT, which agrees on our intuition in Section 4.
**RQ2.** In Figure 1 it is clear that `FourierSAT` solved more problems (472) than MonoSAT (420) and MiniCard (261) did, which indicates that `FourierSAT` is capable of handling cardinality constraints efficiently.
**RQ3.** For the parity learning problem shown in Figure 2, the competition of `FourierSAT` and CMS with `CNF+XOR` formula is roughly a tie, while `FourierSAT` outperformed CMS with pure `CNF` formula. For other two benchmarks

shown in Figure 1 and 3, we observed that CMS has the best performance, especially for solving random hybrid constraints, despite of the choice of encodings.

Due to the maturity of CMS, it is not surprising that CMS performed better than `FourierSAT` did on some benchmarks, especially when dealing with long `XOR` clauses. However, the experimental result shows that as a versatile solver, `FourierSAT` is comparable with many existing, well-developed, specialized solvers.

Furthermore, we notice that although `FourierSAT` is usually slower than other solvers on easy problems, it seems to scale better. One potential reason is, due to the continuous nature of `FourierSAT`, it avoids suffering from scaling exponentially too early. This behavior of `FourierSAT` increases our confidence in that algebraic methods are worth exploring for SAT solving in the future.

## 7 Conclusion and Future Directions

In this paper, we propose a novel algebraic framework for solving Boolean formulas consisting of hybrid constraints. Fourier expansion is 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 guarantee. Finally, we implement our method as `FourierSAT`. The experimental results indicate that in practical, `FourierSAT` is comparable with state-of-the-art solvers on certain benchmarks of hybrid constraints by leveraging parallelization.

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 theoretical tools, such as Hilbert's Nullstellensatz and Gröebner basis (e.g., see [55]) can be used for giving an algebraic proof of unsatisfiability. Previous algorithmic work on this direction considered specific polynomial encodings on problems such as graph coloring [56, 57]. We are interested to see the behavior of these algebraic tools on more problems with Fourier transform as the encoding in practice.

- **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 as well as parallelization. We believe the idea of "sideway moves" from Local Search SAT solvers [11] 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 will also consider purely first-order gradient descent approaches, probably with random noise [37], and how they perform on escaping saddle points both in theory and practice.

- **Alternative objective functions.** The Fourier expansions of clauses with large number of variables and low solution density can be much less smooth, which is one of the main difficulties for 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 [58, 31], it is also promising to use the low-degree approximation of Fourier expansions as the objective function.

## Acknowledgement

Work supported in part by NSF grants IIS-1527668, CCF-1704883, and IIS-1830549.

## References

[1] Joao P Marques-Silva and Karem A Sakallah. GRASP: A Search Algorithm For Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[2] Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[3] Martin Davis, George Logemann, and Donald Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, 1962.

[4] Weiwei Gong and Xu Zhou. A Survey of SAT Solver. *AIP Conference Proceedings*, 1836(1):020059, 2017.

[5] Niklas Eén and Niklas Sörensson. An Extensible SAT-Solver. In *SAT*, pages 502–518, 2003.

[6] Eugene Goldberg and Yakov Novikov. BerkMin: A Fast and Robust Sat-solver. *Discrete Applied Mathematics*, 155(12):1549 – 1561, 2007. SAT 2001.

[7] Armin Biere. PicoSAT Essentials. *JSAT*, 4:75–97, 2008.

[8] Armin Biere. Lingeling , Plingeling , PicoSAT and PrecoSAT at SAT Race 2010, 2010.

[9] Gilles Audemard and Laurent Simon. Lazy Clause Exchange Policy for Parallel SAT Solvers. In *SAT 2014*, pages 197–205, Cham, 2014.

[10] Jia Hui Liang. *Machine Learning for SAT Solvers*. PhD thesis, University of Waterloo, December 2018.

[11] Bart Selman, Hector Levesque, and David Mitchell. A New Method for Solving Hard Satisfiability Problems, 1992.

[12] Bart Selman, Henry Kautz, and Bram Cohen. Local Search Strategies for Satisfiability Testing. *Second DIMACS Implementation Challenge*, 26, 09 1999.

[13] David McAllester, Bart Selman, and Henry Kautz. Evidence for Invariants in Local Search, 1997.

[14] Adrian Balint and Andreas Fröhlich. Improving Stochastic Local Search for SAT with a New Probability Distribution. In *SAT 2010*, pages 10–15, 2010.

[15] Adrian Balint and Uwe Schöning. Choosing Probability Distributions for Stochastic Local Search and the Role of Make versus Break. In *SAT 2012*, pages 16–29, 2012.

[16] Tobias Brueggemann and Walter Kern. An Improved Deterministic Local Search Algorithm for 3-SAT. *Theoretical Computer Science*, 329(1):303 – 313, 2004.

[17] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In *ASIACRYPT 2011*, 2011.

[18] M.-C. Costa, D. de Werra, C. Picouleau, and B. Ries. Graph Coloring with Cardinality Constraints on the Neighborhoods. *Discrete Optimization*, 6(4):362 – 369, 2009.

[19] Irit Dinur, Oded Regev, and Clifford Smyth. The Hardness of 3-Uniform Hypergraph Coloring. *Combinatorica*, 25(5):519–535, Sep 2005.

[20] J. Feldman, M. J. Wainwright, and D. R. Karger. Using Linear Programming to Decode Binary Linear Codes. *IEEE Transactions on Information Theory*, 51(3):954–972, March 2005.

[21] S. Prestwich. CNF Encodings, Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications, 2009.

[22] Claude-Guy Quimper and Toby Walsh. Decomposing Global Grammar Constraints. In *CP 2007*, pages 590–604, 2007.

[23] Henry Kautz, David Mcallester, and Bart Selman. Exploiting Variable Dependency in Local Search. In *Abstracts of the Poster Sessions of IJCAI-97*, pages 9–7, 1997.

[24] R. Martins, V. Manquinho, and I. Lynce. Exploiting Cardinality Encodings in Parallel Maximum Satisfiability. In *ICTAI*, pages 313–320, Nov 2011.

[25] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT Solvers to Cryptographic Problems. In *SAT*, pages 244–257, 2009.

[26] Mark H. Liffiton and Jordyn C. Maglalang. A Cardinality Solver: More Expressive Constraints for Free. In *SAT 2012*, 2012.

[27] Hossein M. Sheini and Karem A. Sakallah. Pueblo: A hybrid pseudo-boolean sat solver. *JSAT*, 2:165–189, 2006.

[28] Jan Elffers and Jakob Nordström. Divide and Conquer: Towards Faster Pseudo-Boolean Solving, 7 2018.

[29] Sam Bayless, Noah Bayless, Holger H. Hoos, and Alan J. Hu. SAT Modulo Monotonic Theories. In *AAAI*, 2015.

[30] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, New York, NY, USA, 2014.

[31] N. Linial, Y. Mansour, and N. Nisan. Constant Depth Circuits, Fourier Transform, and Learnability. In *ASFCS*, Oct 1989.

[32] Colin Wei and Stefano Ermon. General Bounds on Satisfiability Thresholds for Random CSPs via Fourier Analysis, 2017.

[33] Tudor Achim, Ashish Sabharwal, and Stefano Ermon. Beyond Parity Constraints: Fourier Analysis of Hash Functions for Inference. In *ICML*, pages 2254–2262, 2016.

[34] Yexiang Xue, Stefano Ermon, Ronan Le Bras, Carla P. Gomes, and Bart Selman. Variable Elimination in the Fourier Domain. *arXiv e-prints*, page arXiv:1508.04032, Aug 2015.

[35] Joost P. Warners and Hans van Maaren. A Two-phase Algorithm for Solving a Class of Hard Satisfiability Problems. *Operations Research Letters*, 23(3):81 – 88, 1998.

[36] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition. *arXiv e-prints*, page arXiv:1503.02101, Mar 2015.

[37] Chi Jin, Praneeth Netrapalli, Rong Ge, Sham M. Kakade, and Michael I. Jordan. Stochastic Gradient Descent Escapes Saddle Points Efficiently. *arXiv e-prints*, page arXiv:1902.04811, Feb 2019.

[38] Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. Gradient Descent Converges to Minimizers. *arXiv e-prints*, page arXiv:1602.04915, Feb 2016.

[39] Yurii Nesterov. Introductory lectures on convex optimization: A basic course, 2014.

[40] Niao He. Lecture notes in IE 598: Big Data Optimization, 2016.

[41] Dieter Kraft. A Software Package for Sequential Quadratic Programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[42] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

[43] Shaowei Cai, Kaile Su, and Chuan Luo. Improving Walksat for Random k-Satisfiability Problem with k $>$ 3. In *AAAI*, 2013.

[44] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, pages 428–437, 2018.

[45] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *CP 2005*, pages 827–831, 2005.

[46] Batcher. Sorting Networks and Their Applications. In *Spring Joint Computer Conference*, AFIPS '68 (Spring), 1968.

[47] Olivier Bailleux and Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In *CP*, 2003.

[48] Niklas Eén and Niklas Sörensson. Translating Pseudo-Boolean Constraints into SAT. *JSAT*, 2:1–26, 2006.

[49] Chu Min Li. Integrating Equivalency Reasoning into Davis-Putnam Procedure. In *AAAI*, pages 291–296. AAAI Press, 2000.

[50] LLC Gurobi Optimization. Gurobi Optimizer Reference Manual, 2019.

[51] James M Crawford and Michael J Kearns. The Minimal Disagreement Parity Problem as a Hard Satisfiability Problem, 1994.

[52] Holger Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT, 04 2000.

[53] Jeffrey M. Dudek, Kuldeep S. Meel, and Moshe Y. Vardi. Combining the $k$-CNF and XOR Phase-Transitions. In *IJCAI*, Feb 2016.

[54] Yash Pote, Saurabh Joshi, and Kuldeep S. Meel. Phase Transition Behavior of Cardinality and XOR Constraints. In *IJCAI-19*, 7 2019.

[55] David A Cox, John Little, and Donal O'shea. Ideals,Varieties and Algorithms, 1994.

[56] Romero Barbosa, Julian. Applied Hilbert's Nullstellensatz for Combinatorial Problems, 2016.

[57] Jesús A. De Loera, Jon Lee, Peter N. Malkin, and Susan Margulies. Computing Infeasibility Certificates for Combinatorial Problems through Hilbert's Nullstellensatz. *J. Symb. Comput.*, 46:1260–1283, 2011.

[58] Jehoshua Bruck and Roman Smolensky. Polynomial Threshold Functions, AC0 Functions, and Spectral Norms. *SIAM J. Comput.*, 21(1):33–42, February 1992.

# Appendix

### Proof of Proposition 1

**Proof Sketch.** *We reduce counting 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.*

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 weights of items $w_1, w_2, ..., w_n$ and capacity $c$, we construct the pseudo-Boolean constraint $C$: $\sum_i w_i x_i \geq \sum_i w_i - 2c$ where $x \in \{\pm 1\}^n$. It is easy to see that there is a bijection between solutions of $C$ and the knapsack problem. Let the corresponding Boolean function of $C$ be $F_C$ ($F_C = -1$ if $C$ is satisfied) and the number of solutions of $C$ be $N(C)$. By Theorem 1, the constant term $\hat{F}_C(\emptyset) = \mathop{\mathbb{E}}_{x\sim\{\pm 1\}^n}[F_C] = \frac{(-1)\cdot N(C)+1\cdot(2^n - N(C))}{2^n}$ and $N(C) = 2^{n-1}(1 - \hat{F}_C(\emptyset))$. Thus there is a reduction from counting the solutions of a knapsack problem to computing Fourier Coefficients of a pseudo-Boolean function. □

### Proof of Proposition 2

**Fourier expansion of cardinality constraints**

$$\widehat{D^{\geq k}}(\emptyset) = 1 - \frac{\sum_{i=k}^{n}\binom{n}{i}}{2^{n-1}}$$

For every nonempty $S \subseteq [n]$, let $\theta$ be an arbitrary variable, the Fourier coefficient of $D^{\geq k}(S)$ is given by:

$$\widehat{D^{\geq k}}(S) = \frac{\binom{n-1}{k-1}\left((1+\theta)^{n-k}(1-\theta)^{k-1}\right)_{[\theta^{|S|-1}]}}{\binom{n-1}{|S|-1}2^{n-1}},$$

where $\left((1+\theta)^{n-k}(1-\theta)^{k-1}\right)_{[\theta^{|S|-1}]}$ is the coefficient of $\theta^{|S|-1}$ in the expansion of polynomial $\left((1+\theta)^{n-k}(1-\theta)^{k-1}\right)$.

**Derivation of Fourier expansion of cardinality constraints** The following derivation is a generalization of that of Theorem 5.19 in [30].

For the case $|S| = 0$. By the formula of computing Fourier coefficients in Theorem 1, the constant term $\widehat{D^{\geq k}}(\emptyset)$ equals to:

$$\begin{aligned}
\widehat{D^{\geq k}}(\emptyset) &= \mathop{\mathbb{E}}_{x\sim\{\pm 1\}^n}\left[D^{\geq k}(x)\right] \\
&= \tfrac{1}{2^n}\left(\mathop{\#}_{x\in\{\pm 1\}^n} x \text{ has no less than } i \text{ } (-1)\text{'s}\right)\cdot(-1) \\
&+ \tfrac{1}{2^n}\left(\mathop{\#}_{x\in\{\pm 1\}^n} x \text{ has less than } i \text{ } (-1)\text{'s}\right)\cdot 1 \\
&= \frac{\sum_{i=k}^{n}\binom{n}{i}}{2^n}\cdot(-1) + \left(1 - \frac{\sum_{i=k}^{n}\binom{n}{i}}{2^n}\right)\cdot 1 \\
&= 1 - \frac{\sum_{i=k}^{n}\binom{n}{i}}{2^{n-1}}
\end{aligned}$$

For $|S| \neq 0$, we compute the derivative of $D^{\geq k}(x)$, denoted by $\nabla_n D^{\geq k}(x)$.

$$\nabla_n D^{\geq k}(x) = \frac{1}{2}((D^{\geq k}_{n\leftarrow 1}(x)) - (D^{\geq k}_{n\leftarrow -1}(x)))$$

Since $D^{\geq k}(x)$ is a monotonic function, $\nabla_n D^{\geq k}(x)$ is a function from $\{\pm 1\}^{n-1}$ to $\{0, 1\}$, the 0-1 indicator of the set of $(n-1)$-bits strings with exactly $k-1$ coordinates equal to $-1$. By the derivative formula and the fact that $D^{\geq k}(x)$ is symmetric,

$$\widehat{D^{\geq k}}(S) = \widehat{\nabla_n D^{\geq k}}(T)$$

for any $T \subseteq [n-1]$ with $|T| = |S| - 1$.

By the probabilistic definition of $T_\rho$, we have

$$T_\rho(\nabla_n D^{\geq k})(1,1,...,1) = \mathop{E}_{x \sim N_p(1,1,...,1)}[\nabla_n D^{\geq k}(x)] = \mathbb{P}[x \text{ has } (k-1) \text{ } -1\text{'s}]$$

where each coordinate of $x$ is 1 with prob. $\frac{1}{2} + \frac{1}{2}\rho$. Thus

$$T_\rho(\nabla_n D^{\geq k})(1,1,...,1) = \binom{n-1}{k-1}(\frac{1}{2} + \frac{1}{2}\rho)^{k-1}(\frac{1}{2} - \frac{1}{2}\rho)^{n-k}$$

On the other hand, by the Fourier formula for $T_\rho$ and the fact that $\nabla_n D^{\geq k}$ is symmetric we have

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

The equation for computing Fourier coefficients can be obtained by equating coefficients of $\rho$.

**Fourier Coefficients of CNF clauses** First regard all negative literals as positive ones. Since a CNF clause with all positive literals is a special case of cardinality clause, its Fourier expansion can be computed.

Note that the Fourier expansion of $\neg x_i$ is $-x_i$. Thus for each negative literal $\neg x_i$, we just need to flip the sign of all Fourier coefficients $\widehat{F}(S)$ where $i \in S$.

**Fourier Coefficients of XOR clauses** The Fourier expansion of an XOR clause is simply the product of all its literals.

**Fourier Coefficients of NAE clauses.**

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

**Derivation of Fourier expansion of NAE clauses** Note that NAE is a even function, i.e., $\texttt{NAE}(x) = \texttt{NAE}(-x)$, thus $\widehat{\texttt{NAE}}(S) = 0$ for all $S$ s.t. $|S|$ is odd.

For $S$ with even cardinality:

- if $S = \emptyset$, by Theorem 1,

$$\widehat{\texttt{NAE}}(\emptyset) = \mathop{\mathbb{E}}_{x \sim \{\pm 1\}^n}[\texttt{NAE}(x)] = \tfrac{1}{2^n}(2 + (-1) \cdot (2^n - 2)) = (\frac{1}{2})^{n-2} - 1$$

- else if $|S|$ is even and non-zero,

$$\widehat{\texttt{NAE}}(S) = \frac{1}{2^n} \sum_{x \sim \{\pm 1\}^n} [\texttt{NAE}(x) \cdot \chi_S(x)] = \tfrac{1}{2^n} \left( 2 + (-1) \cdot \sum_{\substack{x \in \{\pm 1\}^n \\ x \notin \{\pm \mathbf{1}\}}} \chi_S(x) \right)$$

Since $\sum_{x \in \{\pm 1\}^n} \chi_S(x) = 0$ for all $S \neq \emptyset$, $\sum_{\substack{x \in \{\pm 1\}^n \\ x \notin \{\pm \mathbf{1}\}}} \chi_S(x) = 0 - 1 - (-1)^{|S|} = -2$. Thus, for $S$ with even cardinality,

$$\widehat{\texttt{NAE}}(S) = (\frac{1}{2})^{n-2}$$

□

To prove Theorem 2, we will need to first introduce and prove Lemma 1 and Corollary 2.

Lemma 3 indicates multilinear polynomials are well-behaved in the cube $[-1, 1]^n$.

**Proof of Lemma 1**

**Proof Sketch.** *The first argument is a direct result of the definition of the Fourier expansion of Boolean functions. We proved the second and the third arguments by induction on the number of variables $n$ using the decomposition $p(a) = \frac{1-a_n}{2} \cdot p_{n\leftarrow(-1)}(a_{[n-1]}) + \frac{1+a_n}{2} \cdot p_{n\leftarrow 1}(a_{[n-1]})$.*

The first argument is a direct result of the definition of the Fourier expansion of Boolean functions. We will prove other two arguments by induction on the number of variables $n$.

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

`Inductive step`: Suppose $n \geq 2$. Then, $p(a)$ can be expanded as :

$$p(a) = \tfrac{1-a_n}{2} \cdot p_{n\leftarrow(-1)}(a_{[n-1]}) + \tfrac{1+a_n}{2} \cdot p_{n\leftarrow 1}(a_{[n-1]})$$

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

- If $p_{n\leftarrow(-1)}(a_{[n-1]}) \cdot p_{n\leftarrow 1}(a_{[n-1]}) < 0$. Then,

$$|p(a)| \leq \max\left\{ \tfrac{1-a_n}{2} \cdot \left|p_{n\leftarrow(-1)}(a_{[n-1]})\right|, \tfrac{1+a_n}{2} \cdot \left|p_{n\leftarrow 1}(a_{[n-1]})\right| \right\} \leq 1$$

- If $p_{n\leftarrow(-1)}(a_{[n-1]}) \cdot p_{n\leftarrow 1}(a_{[n-1]}) \geq 0$, then we have,

$$|p(a)| \leq \left(\tfrac{1-a_n}{2} + \tfrac{1+a_n}{2}\right) \cdot \max\left\{ \left|p_{n\leftarrow(-1)}(a_{[n-1]})\right|, \left|p_{n\leftarrow 1}(a_{[n-1]})\right| \right\} \leq 1$$

  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 $p_{n\leftarrow(-1)}(a_{[n-1]}) \cdot p_{n\leftarrow 1}(a_{[n-1]}) < 0$, then, since $|\frac{1+a_n}{2}| < 1, |\frac{1-a_n}{2}| < 1$, what we have by inductive hypothesis is,

$$|p(a)| \leq \max\left\{ \tfrac{1-a_n}{2} \cdot \left|p_{n\leftarrow(-1)}(a_{[n-1]})\right|, \tfrac{1-a_n}{2} \cdot \left|p_{n\leftarrow 1}(a_{[n-1]})\right| \right\} < 1.$$

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

$$|p_{n\leftarrow(-1)}| < 1 \text{ or } |p_{n\leftarrow 1}| < 1.$$

  Therefore both $p_{n\leftarrow(-1)}$ and $p_{n\leftarrow 1}$ are constant and $p$ only relies on one variable, which is covered by the basis.

□

**Proof of Lemma 3**

**Proof Sketch.** *Intuitively, the basic idea of this proof is to show that, for every critical point, there exist two directions such that moving by a small step towards the first one will increase the function value and towards the second one will decrease the value.*

In order to simplify the proof, for a multilinear polynomial $F$ and its critical point $a$, we define the following polynomial $F_a^0$ :

$$F_a^0(x) = F(x + a) - F(a).$$

Notice that $F_a^0(0) = F(a) - F(a) = 0$. It is obvious that if 0 is a saddle point of $F_a^0$ holds, then $a$ is a saddle point of $F$.

Instead of directly proving things about arbitrary multilinear polynomials, we will prove that, for every non-constant multilinear function $f$ with $f(0) = 0$, there exist $\epsilon > 0$ and $v^+, v^- \in \mathbb{R}^n$, such that, for every $0 < \delta < \epsilon$:

$$f(\delta v^+) > 0 \text{ and } f(\delta v^-) < 0.$$

Thus if 0 is a critical point of $f$, it must be a saddle point. We prove the statement above by induction on the number of variables $n$ in $f$.

`Basis step`: Assume $n = 1$. If $f(0) = 0$ and $f$ is non-constant, then $f = \beta x_1$ where $\beta \neq 0$. By assigning $v^+ = (\texttt{sgn}(\beta))$, $v^- = (-\texttt{sgn}(\beta))$ and $\epsilon$ be any positive real number, then, for any $\delta$ such that $0 < \delta < \epsilon$, we have:

$$f(\delta v^+) = |\beta| \cdot \delta > 0,$$
$$f(\delta v^-) = -|\beta| \cdot \delta < 0.$$

`Inductive Step`: Every multilinear function with $n$ ($n \geq 2$) variables can be decomposed as:

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

Note that both $g$ and $h$ are multilinear functions not depending on $x_1$. Since $f$ is non-constant, without loss of generality, we assume $g(x_2, ..., x_n) \not\equiv 0$.

Therefore $g \not\equiv 0$. The rest of this proof is provided case by case. Note that $h(0) = 0$ because $f(0) = 0 \cdot g(0) + h(0) = 0$.

- $h$ is constant, i.e., $h \equiv 0$. Then $f = x_1 g(x_2, ..., x_n)$.
  - $g(x_2, ..., x_n)$ is constant, say $\beta$. We are left with $f = \beta x_1$, a case covered by basis.
  - $g(x_2, ..., x_n)$ is non-constant.
    - ☐ If $g(0) = 0$, then by inductive hypothesis, $\exists v_g^+$, $v_g^-$ and $\epsilon_g$, such that $g(\delta v_g^+) > 0$, $g(\delta v_g^-) < 0$ hold for every $0 < \delta < \epsilon_g$. Set $v^+ = (1, v_g^+)$, $v^- = (1, v_g^-)$ or and $\epsilon = \epsilon_g$. Now for any $\delta$ such that $0 < \delta < \epsilon$,
    $$f(\delta v^+) = \delta \cdot g(\delta v_g^+) > 0$$
    $$f(\delta v^-) = \delta \cdot g(\delta v_g^-) < 0$$
    It is worth mentioning that we could also set $v^+ = (-1, v_g^-)$ and $v^- = (-1, v_g^+)$.
    - ☐ Else if $g(0) \neq 0$, let $v_+ = (\texttt{sgn}(g(0)), \ 0, \ \ldots, \ 0)$, $v_- = (-\texttt{sgn}(g(0)), \ 0, \ \ldots, \ 0)$ and $\epsilon$ be any positive number. For any $\delta$ such that $0 < \delta < \epsilon$,
    $$f(\delta v^+) = \delta \cdot \texttt{sgn}(g(0)) \cdot g(0) > 0,$$
    $$f(\delta v^-) = \delta \cdot (-\texttt{sgn}(g(0))) \cdot g(0) < 0.$$
- $h$ is non-constant, by inductive hypothesis, $\exists v_h^+$, $v_h^-$ and $\epsilon_h$ such that, $h(\delta v_h^+) > 0$, $h(\delta v_h^-) < 0$ hold for every $0 < \delta < \epsilon_h$. We construct $v^+ = (0, v_h^+)$, $v^- = (0, v_h^-)$ and set $\epsilon = \epsilon_h$. For any $\delta$ such that $0 < \delta < \epsilon$,
$$f(\delta v^+) = h(\delta v_h^+) > 0,$$
$$f(\delta v^-) = h(\delta v_h^-) < 0.$$

☐

**Proof of Lemma 4**

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

Suppose $F_{I \leftarrow a_I^\star}$ is non-constant. Since $a^\star_{[n]-I}$ is a critical point inside the cube, by Lemma 3, $a^\star_{[n]-I}$ is a saddle point of $F_{I \leftarrow a_I^\star}$, which means $a^\star$ is not a local minimum of $F$. Thus $F_{I \leftarrow a_I^\star}$ must be constant and $a^\star$ is feasible follows by Definition 3.

☐

**Proof of Proposition 3**

- If $y_i \in [-1,1]$, suppose $\Pi_{[-1,1]^n}(y)_i \neq y_i$, then let $z_i = y_i$ and $z_j = \Pi_{[-1,1]^n}(y)_j$ for $j \neq i$. We have $z \in [-1,1]^n$ and $\|z-y\|_2^2 \leq \left\|\Pi_{[-1,1]^n}(y) - y\right\|_2^2$, a contradiction with the definition of the projection.
- If $|y_i| > 1$, without loss of generality, suppose $y_i > 1$ and $\Pi_{[-1,1]^n}(y)_i < 1$. Let $z_i = 1$ and $z_j = \Pi_{[-1,1]^n}(y)_j$ for $j \neq i$. Similarly, $z \in [-1,1]^n$ and $\|z-y\|_2^2 \leq \left\|\Pi_{[-1,1]^n}(y) - y\right\|_2^2$ still hold.

□

To prove Theorem 4, we need to introduce and prove Proposition 4, Proposition 6 and Lemma 5.

**Proof of Proposition 4**

1. By triangle inequality,

$$\begin{aligned}|F(x) - F(y)| &\leq |F(x_1,...,x_n) - F(y_1,x_2,...,x_n)| + |F(y_1,x_2,...,x_n) - F(y_1,y_2,x_3,...,x_n)| \\ &+ \cdots + |F(y_1,...y_{n-1},x_n) - F(y_1,...,y_n)| \\ &= |(x_1-y_1)\nabla_1 F(x_1,...,x_n)| + \cdots + |(x_n-y_n)\nabla_n F(y_1,...,y_{n-1},x_n)|\end{aligned}$$

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

$$\begin{aligned}&|(x_1-y_1)\nabla_1 F(x_1,...,x_n)| + \cdots + |(x_n-y_n)\nabla_n F(y_1,...,y_{n-1},x_n)| \leq \alpha(\sum_{i=1}^n |x_i - y_i|) \\ \leq &\alpha n^{\frac{1}{2}}||x-y||_2\end{aligned}$$

2. Since $\nabla_i F(x) = \frac{\partial F}{\partial x_i}(x) \in [-\alpha,\alpha]$ for all $i \in [n]$, by 1. of Proposition 4 we have

$$||\nabla F(x) - \nabla F(y)|| = (\sum_{i=1}^n (\nabla F_i(x) - \nabla F_i(y))^2)^{\frac{1}{2}} \leq (n \cdot (\alpha n^{\frac{1}{2}}||x-y||_2)^2)^{\frac{1}{2}} = \alpha n||x-y||_2$$

3. Similarly, $\nabla^2_{i,j} F(x) \in [-\alpha,\alpha]$ for all $i,j \in [n]$. By applying 1 of Proposition 4. again and the fact that l2 norm is no more than Frobenius norm, we get:

$$\begin{aligned}||\nabla^2 F(x) - \nabla^2 F(y)||_2 \leq &||\nabla^2 F(x) - \nabla^2 F(y)||_F = \sum_{i,j\in[n]} (\nabla^2_{i,j} F(x) - \nabla^2_{i,j} F(y))^2 \\ \leq &(n^2(\alpha n^{\frac{1}{2}}||x-y||_2)^2)^{\frac{1}{2}} = \alpha n^{\frac{3}{2}}||x-y||_2\end{aligned}$$

□

**Proposition 6.** *(Inequality of gradient mapping) For a function $F$ and every $x \in \texttt{dom}(F)$,*

$$\langle \nabla F(x),\ G(x)\rangle \geq \|G(x)\|_2^2$$

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

**Proof of Proposition 6**

Since $x_{t+1}$ is the optimum of the Euclidean projection onto convex set $\Delta$, we have

$$\langle x_{t+1} - x'_{t+1}, z - x_{t+1}\rangle \geq 0,$$

for all $z \in \Delta$. Let $z = x_t$. Notice that $x_{t+1} - x'_{t+1} = G(x_t) - \nabla F(x_t)$ and $x_t - x_{t+1} = -G(x_t)$, we get:

$$\langle G(x_t) - \nabla F(x_t), -G(x_t)\rangle \geq 0,$$

and the statement follows directly. □

**Lemma 5.** *(Descent Lemma). For a multilinear polynomial $F : [-1,1]^n \to [-\alpha,\alpha]$ 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 $x_t$ satisfies:*

$$F(x_{t+1}) - F(x_t) \leq -\frac{\eta}{2}\|G(x_t)\|_2^2.$$

**Proof of Lemma 5**

According to the $(\alpha n)$-gradient Lipschitz continuity property Proposition 4, we have:

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

Note that the last inequality follows by Proposition 6. □

**Proof of Theorem 4**

Assume that $\eta = \frac{1}{nm}$. Then, the recursion in Lemma 5 satisfies:

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

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

$$\begin{aligned} F(x_{T+1}) &\leq F(x_T) - \tfrac{1}{2nm} \|G(x_T)\|^2 \\ F(x_T) &\leq F(x_{T-1}) - \tfrac{1}{2nm} \|G(x_{T-1})\|^2 \\ &\vdots \\ F(x_1) &\leq F(x_0) - \tfrac{1}{2nm} \|G(x_0)\|^2 \end{aligned}$$

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

$$\tfrac{1}{2nm} \sum_{t=0}^{T} \|G(x_t)\|^2 \leq F(x_0) - F(x^\star).$$

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:

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

Then,

$$\begin{aligned} \tfrac{T+1}{2nm} \cdot \min_t \|G(x_t)\|^2 \leq \tfrac{1}{2nm} \sum_{t=0}^{T} \|G(x_t)\|^2 &\leq F(x_0) - F(x^\star) \\ \Rightarrow \min_t \|G(x_t)\|^2 &\leq \tfrac{2nm}{T+1} \cdot (F(x_0) - F(x^\star)) \\ \min_t \|G(x_t)\| &\leq \sqrt{\tfrac{2nm}{T+1}} \cdot (F(x_0) - F(x^\star))^{\frac{1}{2}} = O\left(\tfrac{1}{\sqrt{T}}\right). \end{aligned}$$

Note that $|F(x_0) - F(x^*)| \leq m$. Thus, to achieve a point such that $\|G(x_T)\| \leq \varepsilon$, we require:

$$\min_t \|G(x_t)\| \leq \varepsilon \Rightarrow \sqrt{\tfrac{2nm}{T+1}} \cdot (F(x_0) - F(x^\star))^{\frac{1}{2}} \leq \varepsilon \Rightarrow O\left(\tfrac{nm^2}{\varepsilon^2}\right) \text{ iterations.}$$

□

**Proof of Proposition 5**

For every $i \in [n]$, by definition of gradient mapping, $G(x)_i = 0$ means

$$x_i = \Pi_{[-1,1]}(x_i - \eta \nabla F(x)_i)$$

Since $x$ is feasible, if $x_i \in (-1, 1)$. i.e. $i \in [n] - I$, then $\nabla F(x)_i = 0$. When $\nabla F(x)_i \neq 0$, $x_i \neq x_i - \eta \nabla F(x)_i$. By Proposition 3, if $x_i = -1$, then $\nabla F(x)_i > 0$; if $x_i = 1$, then $\nabla F(x)_i < 0$.

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

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

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

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

  $$\nabla_v F(x) = \sum_{i \in [n]} v_i \nabla F(x)_i.$$

  First, note that $v_i \nabla F(x)_i \geq 0$ for all $i \in [n]$ ,thus $\nabla_v F(x) \geq 0$. Since $\nabla F(x)_{i^*} \neq 0$ and $v_{i^*} \neq 0$ we have $\nabla_v F(x) > 0$ for any $v$ and $F(x) < F(x + \eta \cdot v)$.

Therefore, $x$ is a local minimum. □

**Algorithm 2:** `DecInnerSaddle`$(F,\ x,\ \eta)$

**Input :** Polynomial $F$, non-feasible critical point $x$ of $F$, step size $\eta > 0$.
**Output :** $x$ after moving towards a negative direction.

$I \leftarrow \{i \mid x_i \in \{-1, 1\}\}$
$F_N(y) := F_{I \leftarrow x_I}(y + x_{[n]-I}) - F(x)$ //Lemma 3
$v =$ `NegDirectionSaddle`$(F_N)$
**return** $x + \eta \cdot v$

**Algorithm 3:** `NegDirectionSaddle`$(F)$

**Input :** Polynomial $F$ such that $F(0) = 0$.
**Output :** A negative direction $v^-$ of $F$ at point 0.

// See proof of Lemma 3
**if** *isZero*$(F_{1 \leftarrow 0}) =$ *True* **then**
  **if** *isZero*$(F_{1 \leftarrow 1} - F(1, 0, ..., 0)) =$ *True* **then**
    $v^- = (\text{sgn}(F_{1 \leftarrow 1}), 0, ..., 0)$
  **else**
    **if** $F(1, 0, ..., 0) = 0$ **then**
      $v^- = (1,$ `NegDirectionSaddle`$(F(1, 0, ..., 0)))$
    **else**
      $v^- = (-\text{sgn}(F(1, 0, ..., 0), 0, ..., 0))$
**else**
  $v^- \leftarrow (0,$ `NegDirectionSaddle`$(F_{1 \leftarrow 0}))$
**return** $v^-$

**Algorithm 4:** `isZero`$(F)$

**Input :** Polynomial $F$
**Output :** `True`/`False` on $F \equiv 0$

$x \sim \mathcal{U}[-1, 1]^n$
**return** `True` if $F(x) = 0$, otherwise `False`

**Proposition 7.** *When algorithm 5 reaches line 4 or 7, $v$ is a negative direction of $F$. I.e., $F(x + \eta v) < F(x)$ and $x + \eta v$ is within the cube $[-1, 1]^n$.*

**Algorithm 5:** `useHessian`$(F, x)$

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

1 $J = \{j \mid \nabla F(x)_j = 0\}$ and $I = \{i \mid x_i \in \{\pm 1\} \wedge i \in J\}$
2 $H = \nabla^2 F_{[n]-J \leftarrow x_{[n]-J}}(x_J)$
3 **if** $\exists i \in I,\ j \in J - I$ *such that* $H_{ij} \neq 0$ **then**
4 $\quad v \leftarrow (0, 0, ..., v_i = -\mathtt{sgn}(x_i), 0, ..., 0, v_j = \mathtt{sgn}(x_i \cdot H_{ij}), 0, ..., 0)$ // see Prop 7
5 $\quad$ **return** $(x + \eta \cdot v,$ `False`$)$
6 **else if** $\exists i_1, i_2 \in I$ *such that* $H_{ij} \cdot x_{i_1} x_{i_2} < 0$ **then**
7 $\quad v \leftarrow (0, 0, ..., v_i = -\mathtt{sgn}(x_{i_1}), 0, ..., 0, v_j = -\mathtt{sgn}(x_{i_2}), 0, ..., 0)$ // see Prop 7
8 $\quad$ **return** $(x + \eta \cdot v,$ `False`$)$
9 **else if** $\forall i, j,\ i \neq j$ *and at least one of* $i$ *and* $j$ *is in* $I$ *such that* $H_{ij} \neq 0$ **then**
10 $\quad$ **return** $(x,$ `True`$)$ // See Prop 8
11 **else**
12 $\quad$ **return** $(x,$ `Unknown`$)$ //`We meet a very rare degenerate saddle point at the corner. Do random restart`

**Proof of Proposition 7**

Consider partially assigned function $F' = F_{[n]-J \leftarrow x_{[n]-J}}$. By definition of $J$, $\nabla F'(x_J) = \mathbf{0}$. In the following cases, we will use the **second directional derivative** of $F'$ at $x_J$ in direction $v$, denoted by $\nabla^2 F'_v(x_J)$ as our tool, which can be computed as:

$$\nabla^2 F'_v = \sum_{i,j \in J} v_i v_j H_{ij} = 2 \sum_{i,j \in J, i \neq j} v_i v_j H_{ij},$$

where $H$ is the Hessian of $F'$ at $x_J$ 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 $\eta$ being small enough, $x_J + \eta \cdot v \in [-1,1]^{|J|}$ because $x_i \in \{\pm 1\}$ and $x_i - \eta \cdot \mathtt{sgn}(x_i) \in [-1,1]$; $x_j \in (-1,1)$ thus $x_j + \mathtt{sgn}(x_i \cdot H_{ij}) \in [-1,1]$. Moreover,

$$\nabla^2 F'_v = 2 \cdot (-\mathtt{sgn}(x_i))\mathtt{sgn}(x_i \cdot H_{ij}) \cdot H_{ij} < 0$$

Thus, moving towards $v$ will decrease $\nabla F'_v$. Since $\nabla F'(x_J) = \mathbf{0}$ we have $\nabla F'_v(x_J) = 0$. Therefore $\nabla F'_v(x_J + \eta v) < 0$ and $F'(x_J + v) < F'(x_J)$. Hence $v$ is a also negative direction of $F$ at $x$ since $v$ keeps coordinates of $x$ in $[n] - J$ unchanged.

– Suppose Algorithm 5 reaches line 7. Also it is easy to confirm that $x_J + \eta \cdot v \in [-1,1]^{|J|}$. Meanwhile,

$$\nabla^2 F'_v = 2 \cdot (-\mathtt{sgn}(x_{i_1}))(-\mathtt{sgn}(x_{i_2})) \cdot H_{ij} < 0$$

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 8.** *When algorithm 5 reaches line 10, $x$ is a local minimum.*

**Proof of Proposition 8**

Similar with proof of Proposition 7, let $F' = F_{[n]-J \leftarrow x_{[n]-J}}$. We have $\nabla F'(x_J) = \mathbf{0}$. Recall that

$$\nabla^2 F'_v = 2 \sum_{i,j \in J, i \neq j} v_i v_j H_{ij}$$

Consider all the directions $v$ ($\|v\| \neq 0$) where $x + \eta \cdot v \in [-1,1]^{|J|}$. When the algorithm runs line 10, neither of condition in line 3 nor 6 holds. Also $H_{ij} = 0$ for every $i, j \in J - I$ because $x_J$ is a feasible solution of $F'$ so that $F'_{I \leftarrow x_I}$ is constant. Therefore $v_i v_j H_{ij} \geq 0$ for all $i, j \in J$ and $\nabla^2 F'_v = 2 \sum_{i,j \in J, i \neq j} v_i v_j H_{ij} \geq 0$.

– If $v$ has two or more non-zero elements then $\nabla^2 F'_v > 0$. Since $\nabla F'_v = 0$, $F'(x_J) < F'(x_J + \eta v)$.

– Else If $v$ has only one non-zero element, say $v_1 \neq 0$, then $\nabla^2 F'_v(x_J + \eta \cdot v) = H_{11} = 0$. Thus moving towards $v$ does not change $\nabla F'_v$, combining which with $\nabla F'_1(x_J) = 0$ we know that moving towards $v$ does not change $F'$, i.e., $F'(x_J) = F'(x_J + \eta v)$.

Therefore $x_J$ is a local minimum of $F'$ and we will use it to show $x$ is a local minimum of $F$. Suppose $x$ is not a local minimum of $F$ and there exists $v$ s.t. $x + \eta \cdot v \in [-1, 1]^n$ and $F(x + \eta \cdot v) < F(x)$. Then there exists $i \in [n] - J$ s.t. $v_i \neq 0$, otherwise $x_J$ is not a local minimum of $F'$. However, recall $G(x) = \mathbf{0}$ and $\nabla F(x)_i \neq 0$ because $i \in [n] - J$, the directional derivative $\nabla F_v(x) > 0$ by similar analysis in the proof of Proposition 5, which conflicts with $F(x + \eta \cdot v) < F(x)$. □

In Algorithm 1, we used Algorithm 4 to test whether a multilinear polynomial is equivalent to 0 probabilistically. In this way, we bypass tedious representations of multilinear polynomials. Proposition 9 shows this testing method is always correct in the sense of probability.

**Proposition 9.** *(Constant Testing) For a multilinear polynomial $F \not\equiv 0$ and a uniformly random vector $x$ from $[-1, 1]^n$,*

$$\mathop{\mathbb{P}}_{x \sim [-1,1]^n}[F(x) = 0] = 0$$

**Proof of Proposition 9**

If $F$ is a constant other than 0, the statement holds trivially. Thus we assume $F$ is non-constant.

`Basis`: $n = 1$. $F(x) = ax_1 + b$ where $a \neq 0$. $F(x) = 0$ has only one root $x_1^* = -\frac{b}{a}$. Thus $\mathop{\mathbb{P}}_{x \sim [-1,1]^n}[F(x) = 0] = 0$.

`Inductive Step`: suppose $F$ has $n$ variables ($n \geq 2$). Decompose $F$ as:

$$F(x) = x_1 g(x_2, ..., x_n) + h(x_2, ..., x_n)$$

- If both $h$ and $g$ are constant, the case degenerates to the basis.
- If $g(x)$ is constant and $h(x)$ is non-constant, let $g(x) \equiv a$. Then we assume $a \neq 0$, otherwise $F(x)$ degenerates to a polynomial with $n - 1$ variables.

$$\mathop{\mathbb{P}}_{x \sim [-1,1]^n}[F(x) = 0] = \mathop{\mathbb{P}}_{x_1 \sim [-1,1]}[x_1 = -\frac{h(x)}{a}] = 0$$

- If $g(x)$ is non-constant and $h(x)$ is constant. Let $h(x) \equiv b$

  – If $b = 0$,

  $$\mathop{\mathbb{P}}_{x \sim [-1,1]^n}[F(x) = 0] \leq \mathop{\mathbb{P}}_{x \sim [-1,1]^{n-1}}[g(x) = 0] + \mathop{\mathbb{P}}_{x_1 \sim [-1,1]}[x_1 = 0] = 0$$

  – If $b \neq 0$,

  $$\mathop{\mathbb{P}}_{x \sim [-1,1]^n}[F(x) = 0] = \mathop{\mathbb{P}}_{x \sim [-1,1]^n}[g(x) \neq 0] \cdot \mathop{\mathbb{P}}_{x_1 \sim [-1,1]}[x_1 = -\frac{h(x)}{g(x)}] = 0$$

- If both $h$ and $g$ are non-constant:

$$\mathop{\mathbb{P}}_{x \sim [-1,1]^n}[F(x) = 0] = \mathop{\mathbb{P}}_{x \sim [-1,1]^{n-1}}[g(x) = 0] \cdot \mathop{\mathbb{P}}_{x \sim [-1,1]^{n-1}}[h(x) = 0]$$
$$+ \mathop{\mathbb{P}}_{x \sim [-1,1]^n}[g(x) \neq 0] \cdot \mathop{\mathbb{P}}_{x_1 \sim [-1,1]}[x_1 = -\frac{h(x)}{g(x)}] = 0$$

□