On Continuous Local BDD-Based Search for Hybrid SAT Solving[*]

Anastasios Kyrillidis, Moshe Vardi, Zhiwei Zhang
Rice University, 6100 Main Street, Houston, TX, USA
{anastasios, vardi, zhiwei}@rice.edu

Abstract

We explore the potential of continuous local search (CLS) in SAT solving by proposing a novel approach for finding a solution of a hybrid system of Boolean constraints. The algorithm is based on CLS combined with belief propagation on binary decision diagrams (BDDs). Our framework accepts all Boolean constraints that admit compact BDDs, including symmetric Boolean constraints and small-coefficient pseudo-Boolean constraints as interesting families. We propose a novel algorithm for efficiently computing the gradient needed by CLS. We study the capabilities and limitations of our versatile CLS solver, `GradSAT`, by applying it on many benchmark instances. The experimental results indicate that `GradSAT` can be a useful addition to the portfolio of existing SAT and MaxSAT solvers for solving Boolean satisfiability and optimization problems.

Introduction

Constraint-satisfaction problems (CSPs) are fundamental in mathematics, physics, and computer science. The Boolean SATisfiability problem (SAT) is a paradigmatic 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, formulas in conjunctive normal form (CNFs) are a conjunction (`and`-ing) of disjunctions (`or`-ing) of literals. Despite the NP-completeness of CNF-SAT, there has been dramatic progress on the engineering side of CNF-SAT solvers (Vardi 2014). 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 nor can it prove unsatisfiability.

Most modern complete SAT solvers are based on the the Conflict-Driven Clause Learning (CDCL) algorithm introduced in GRASP (Marques-Silva and Sakallah 1999), an evolution of the backtracking Davis-Putnam-Logemann-Loveland (DPLL) algorithm (Davis and Putnam 1960; Davis, Logemann, and Loveland 1962). Examples of highly efficient complete SAT solvers include Chaff (Moskewicz et al. 2001), MiniSat (Eén and Sörensson 2003), PicoSAT (Biere 2008), Lingeling [1], Glucose (Audemard and Simon 2014), and machine learning-enhanced MapleSAT (Liang 2018). Overall, CDCL-based SAT solvers constitute a huge success for SAT problems, and have been dominating in the research of SAT solving.

Discrete local search (DLS) methods are used in incomplete SAT solvers. The number of unsatisfied constraints is often regarded as the objective function. DLS algorithms include greedy local search (GSAT) (Selman, Levesque, and Mitchell 1992) and random walk GSAT (WSAT) (Selman, Kautz, and Cohen 1999). Several efficient variants of GSAT and WSAT have been proposed, such as NSAT (McAllester, Selman, and Kautz 1997), Novelty+ (Hoos and Stützle 2000), SAPS (Hutter, Tompkins, and Hoos 2002), ProbSAT (Balint and Schöning 2012), and CCAnr (Cai, Luo, and Su 2015). While practical DLS solvers could be slower than CDCL solvers, they are useful for solving certain classes of instances, e.g., hard random 3-CNF and MaxSAT (Selman, Levesque, and Mitchell 1992).

Continuous local search (CLS) algorithms are much less studied in SAT community, compared with CDCL and DLS methods. In (Kamath et al. 1990), the SAT problem is regarded as an integer linear programming (ILP) problem and is solved by interior point method after relaxing the problem to linear programming (LP). Another work (Jun Gu 1994) converts SAT into an unconstrained global minimization problem and applies coordinate descent. Those methods, however, are only able to solve relatively easy random CNF instances and fail to provide interesting theoretical guarantees regarding rounding and convergence.

Non-CNF constraints are playing important roles in computer science and other engineering areas, *e.g.*, XOR constraints in cryptography (Bogdanov, Khovratovich, and Rechberger 2011), as well as Pseudo-Boolean constraints and Not-all-equal (NAE) constraints in discrete optimization (Costa et al. 2009; Dinur, Regev, and Smyth 2005). The combination of different types of constraints enhances the expressive power of Boolean formulas. Nevertheless, com-

[*]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.

[1]A. Biere, Lingeling , Plingeling , PicoSAT and PrecoSAT at SAT Race 2010

pared to that of CNF-SAT solving, efficient SAT solvers that can handle non-CNF constraints are less well studied.

There are two main approaches to handle hybrid Boolean systems, i.e., instances with non-CNF constraints: 1) CNF encoding and 2) extensions of existing SAT solvers. For the first approach, different encodings differ in size, the ability to detect inconsistencies by unit propagation (arc consistency), and solution density (Prestwich 2009). It is generally observed that the running time of SAT solvers relies heavily on the detail of encodings. Finding the best encoding for a solver usually requires considerable testing and comparison (Martins, Manquinho, and Lynce 2011). In addition, the encodings of different constraints generally do not share extra variables, which increases the number of variables. The second approach often requires different techniques for various types of constraints, e.g., cryptoMiniSAT (Soos, Nohl, and Castelluccia 2009) uses Gaussian Elimination to handle XOR constraints, while Pueblo (Sheini and Sakallah 2006) leverages cutting plane to take care of pseudo-Boolean constraints. Despite the efforts of Dixon et a. (Dixon et al. 2011) to develop a general satisfiability framework based on group theory, the study of a uniform approach for solving hybrid constraints is far from mature.

Recently, an algebraic framework named FourierSAT (Kyrillidis et al. 2020) attempted to address the issues with handling non-CNF constraints, bringing CLS methods back to the view of SAT community. As an algorithmic application of Walsh-Fourier analysis of Boolean functions, FourierSAT transforms Boolean constraints into multilinear polynomials, which are then used to construct the objective function, where a solution is found by applying vector-wise gradient-based local search in the real cube $[-1, 1]^n$. In this framework, different constraints are handled uniformly in the sense that they are all treated as polynomials. Compared with the earlier CLS algorithm (Jun Gu 1994), FourierSAT is able to handle more types of constraints than just random CNF formulas. Moreover, FourierSAT provides interesting theoretical properties of rounding as well as global convergence speed thanks to a better-behaved objective function. In addition, taking advantage of the infrastructure for training neural networks, including software libraries and hardware, to solve discrete problems has become a hot topic recently (Dudek and Vardi 2020). The gradient descent-based framework of FourierSAT provides the SAT community a possible way of benefiting from machine-learning methods and tools.

Though FourierSAT does bring attention back to CLS methods, it still has some limitations. First, as a CLS approach, FourierSAT suffers from inefficient gradient computation. Specifically, computing the gradient of n coordinates can take $O(n)$ function evaluations. Second, the types of constraints accepted by FourierSAT are still somehow limited—only constraints with closed-form Walsh-Fourier expansions can be handled. Third, FourierSAT uses random restart when getting stuck in a local optimum, and the information of the previous local optima is not leveraged.

Contributions. We point out here that the core of CLS methods for SAT solving is the ability of efficiently evaluating and differentiating (computing the gradient) the objective, independent of the representations we use for Boolean constraints. We show that the evaluation and differentiation problem in the framework of FourierSAT are equivalent with weighted model counting and circuit-output probability. Model counting and circuit-output probability are hard computational problems for general Boolean functions (Valiant 1979). Nevertheless, if a Boolean function can be compactly represented by certain structures, e.g., Binary Decision Diagram (BDD) (Bryant 1995), then those tasks can be done efficiently. As a widely-used representation of Boolean constraints, BDD has applications in synthesis, verification and counting. Thus practical packages for handling BDDs, e.g., CUDD [2] have been developed. Such packages provide sub-function sharing between individual BDDs so that a set of BDDs can be stored compactly.

We propose a novel algorithm for computing the gradient of the objective based on *belief propagation* on BDDs. We prove that by our algorithm, the complexity of computing the gradient is linear in terms of the size of shared BDDs. In fact, computing gradient for n coordinates is as cheap as evaluating the objective function.

We then extend FourierSAT to a more versatile incomplete SAT solver, which we call `GradSAT` by using a more general data structure, BDDs, rather than the Walsh-Fourier expansions. As interesting families of Boolean functions that have compact BDDs, pseudo-Boolean constraints and symmetric Boolean constraints, including CNF, XOR and cardinality constraints, have abundant applications. We enhance the performance of `GradSAT` in a variety of ways. We borrow the idea of adaptive weighting form DLS (Morris 1993) and Discrete Lagrange Method (Wah and Shang 1996) to make use of solving history and search the continuous space more systematically. We also take advantage of a portfolio of continuous optimizers and parallelization.

In the experimental section, we aim to demonstrate the capabilities and limitations of CLS approaches represented by our tool, `GradSAT`. On one hand, our method owns nice versatility and can outperform CDCL and DLS solvers on certain problems, showing the power of CLS and hybrid Boolean encodings instead of pure CNF. On the other hand, we observe that on certain benchmarks, e.g., formulas with XORs, CDCL solvers still offer better performance, revealing potential limitations of CLS methods for handling instances with solution-space shattering. We conclude that CLS methods can be a useful complement to strengthen the SAT-solver portfolio. In addition, since local search-based SAT solvers can be naturally used to solve MaxSAT problems (Kautz, Sabharwal, and Selman 2009), we apply `GradSAT` to many MaxSAT instances and obtain encouraging results.

Preliminaries

Boolean Formulas, Constraints and BDDs

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 $\{\texttt{True}, \texttt{False}\}$. In this paper, we

[2]CUDD: CU Decision Diagram Package Release 3.0.0

define a Boolean function by $f : \{\pm 1\}^n \to \{0,1\}$, where for variables, -1 stands for `True` and $+1$ for `False` and for the value, 1 stands for `True` and 0 for `False`. A vector $a \in \{-1,1\}^n$ is called an assignment. A literal l_i is either a variable x_i or its negation $\neg x_i$. In the rest of this paper, we focus on the case where $f = c_1 \wedge c_2 \wedge \cdots \wedge c_m$. In other words, f is the logical-AND of m Boolean functions, where each c_i is called a constraint.

- *Symmetric Boolean constraints:* Assume every literal appears at most once in a constraint. Let $c(l_1 \cdots l_{n_c})$ be a constraint determined by literals $(l_1 \cdots l_{n_c})$. c is called **symmetric** if $c(l_1 \cdots l_{n_c}) \equiv c(\delta(l_1), \cdots, \delta(l_{n_c}))$ holds for every permutation δ on $(l_1 \cdots l_{n_c})$. Disjunctive clauses are symmetric. For example, $(x_1 \vee \neg x_2 \vee x_3) \equiv (\neg x_2 \vee x_3 \vee x_1)$. Other interesting classes of symmetric constraints include cardinality constraints (CARD), e.g., $x_1 + x_2 + x_3 \geq 2$; XOR, e.g., $x_1 \oplus x_2 \oplus x_3$; Not-all-equal constraints (NAE), which are satisfied when not all the variables have the same value.
- Pseudo-Boolean constraints (PB), e.g, $3x_1 + 5\neg x_2 - 6x_3 \geq 2$, where the coefficients are limited to be integers.

Let the set of constraints 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 constraints in C_f.

A Binary Decision Diagram, or BDD, is a data structure that represents a Boolean function as a directed acyclic graph B (Bryant 1995). We use $B.V$, $B.E$ and $S(B) = |B.V| + |B.E|$ to denote the node set, edge set and size of B. For a node $v \in B.V$, we use $v.T$, $v.F$ and i_v to denote the `True`, `False` child and the variable index associated with v, respectively. We use $B.one$ and $B.zero$ to denote the terminal nodes with value 1 and 0, respectively. We also use multi-rooted BDDs (MRBDDs) for representing a group of Boolean functions, in which the BDD can have multiple entries, stored in $B.entry : C_f \to B.V$. Each entry of the MRBDD is the root of the single-rooted BDD of one Boolean function.

Walsh-Fourier Expansion of a Boolean Function

Walsh-Fourier Transform is a method for transforming a Boolean function into a multilinear polynomial. The following theorem states that every function defined on a Boolean hyper-cube has an equivalent polynomial representation.

Theorem 1. ((O'Donnell 2014), Walsh-Fourier Expansion) *Given a function $f : \{\pm 1\}^n \to \mathbb{R}$, there is a unique way of expressing f as a multilinear polynomial, 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 $\widehat{f}(S) \in \mathbb{R}$ is called Walsh-Fourier coefficient, given S, and computed as:

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

*$x \sim \{\pm 1\}^n$ indicates x is uniformly sampled from $\{\pm 1\}^n$. The polynomial is called the **Walsh-Fourier expansion** of f.*

c	$\texttt{WFE}_c$
$x_1 \vee x_2$	$\frac{3}{4} - \frac{1}{4}x_1 - \frac{1}{4}x_2 - \frac{1}{4}x_1x_2$
$x_1 \oplus x_2 \oplus x_3$	$\frac{1}{2} - \frac{1}{2}x_1x_2x_3$
$\text{NAE}(x_1, x_2, x_3)$	$\frac{3}{4} - \frac{1}{4}x_1x_2 - \frac{1}{4}x_2x_3 - \frac{1}{4}x_1x_3$

Table 1: Examples of Walsh-Fourier Expansions

Given a formula f, for each constraint c of f, we use $\texttt{WFE}_c$ to denote the Walsh-Fourier expansion of c. Table 1 shows some examples of Walsh-Fourier expansions.

Weighted Model Counting and Circuit-Output Probability

Definition 1. (Weighted model counting) *Let $f : \{-1,1\}^n \to \{0,1\}$ be a Boolean function over a set X of variables. Let $W : \{-1,1\}^n \to \mathbb{R}$ be an arbitrary function. The weighted model counting of f w.r.t. W is $W(f) = \sum_{a \in \{-1,1\}^n} f(a) \cdot W(a)$.*

W is called the weight function. In this work we focus on so-called literal-weight functions, where the weight of an assignment can be expressed as the product of weights associated with all satisfied literals: $W(a) = \prod_{a_i=-1} W^T(x_i) \cdot \prod_{a_i=1} W^F(x_i)$ for some literal weight functions $W^T, W^F : X \to \mathbb{R}$.

When the literal weight functions satisfy $W^T(x_i) + W^F(x_i) = 1$ and $W^T(x_i), W^F(x_i) \geq 0$ for each variable x_i, the literal weighted model counting problem reduces to the **circuit-output probability problem**.

Definition 2. (Circuit-output probability) *Let f be a Boolean function over a set X of variables. Let $P : X \to [0,1]$ be the variable input probability. The circuit-output-probability problem of f w.r.t. P, denoted by $\mathit{COP}(P,f)$ is the probability of f outputting 1 (`True`) given the value of each variable independently sampled from the binary distribution with probability $\mathbb{P}[x_i = -1] = P(x_i)$, i.e.,*

$$\mathit{COP}(P,f) = \sum_{a \in \{\pm 1\}^{|X|}} f(a) \cdot \prod_{a_i=-1} P(x_i) \cdot \prod_{a_i=1} (1 - P(x_i))$$

Theoretical Framework

In this section, we first recap a previous CLS framework, FourierSAT and give a weighted adaptation of it. Then we show how our new CLS-based method, `GradSAT`, can accept an abundant set of different types of constraints and run efficiently due to fast gradient computation. Proofs are delayed to the supplemental material [3].

Recap of FourierSAT and a Weighted Adaptation

The basic framework of FourierSAT (Kyrillidis et al. 2020) is as follows. Given a formula f, the **objective function** w.r.t f, denoted by $F_f : [-1,1]^n \to \mathbb{R}^+$, is the sum of Walsh-Fourier expansions of f's constraints, i.e., $F_f = \sum_{c \in C_f} \texttt{WFE}_c$, where $\texttt{WFE}_c$ is the Walsh-Fourier expansion

[3]available at https://github.com/vardigroup/GradSAT/blob/master/Appendix.pdf

Algorithm 1: FourierSAT, a CLS-based SAT Solver.

```
Input  : Boolean formula $f$ with constraint set $C_f$
Output: A discrete assignment $x \in \{-1,1\}^n$
1 for $j = 1, \ldots, J$ do
2     Randomly sample $x_0$ uniformly from $[-1,1]^n$.
3     Search for a local maximum $x_j^*$ of $F_f$ in $[-1,1]^n$,
        starting from $x_0$.
4     if $F_f(x_j^*) = |C_f|$ then return $x_j^*$;
5 return $x_j^*$ with the highest $F_f(x_j^*)$ after $J$ iterators
```

of constraint c. The following theorem reduces SAT to a multivariate optimization problem over $[-1,1]^n$.

Theorem 2. (Kyrillidis et al. 2020) [4] *A Boolean formula f is satisfiable if and only if $\max_{x \in [-1,1]^n} F_f(x) = |C_f|$.*

Theorem 2 suggests that we can search *inside* of the Boolean cube instead of only on vertices. Then a continuous optimizer is applied for finding the global maximum of the objective. The objective built by Walsh-Fourier expansions is multilinear, which is better-behaved than higher order polynomials. When getting stuck in a local maximum, FourierSAT simply invokes random restart, as Alg. 1 shows.

A limitation of Theorem 2 is that different relative importance of constraints is not taken into account. For example, constraints with low solution density are generally harder to satisfy than those with high solution density. We address this issue by involving a *constraint-weight function* $w : C_f \to \mathbb{R}^+$. We define the new objective as follows.

Definition 3. (Objective) *The objective function w.r.t. the formula f and constraint-weight function w, denoted by $F_{f,w} : [-1,1]^n \to \mathbb{R}$ is defined as $F_{f,w} = \sum_{c \in C_f} w(c) \cdot \mathtt{WFE}_c$.*

We have the weighted analogue of Theorem 2 as follows.

Theorem 3. *Given a constraint weight function $w : C_f \to \mathbb{R}^+$, a Boolean formula f is satisfiable if and only if*

$$\max_{x \in [-1,1]^n} F_{f,w}(x) = \sum_{c \in C_f} w(c).$$

Based on Theorem 3, we can design a CLS framework similar to Alg. 1 to search for a maximum of $F_{f,w}$.

Computing Gradients is Critical for CLS Methods

After constructing the objective function, a CLS approach has to find a global optimum. Global optimization on non-convex functions is NP-hard (Jain and Kar 2017), so converging to local optima and checking if any of them is global is more practical, as long as global optima can be identified efficiently. Algorithms that aim to find local optima can be categorized into several classes of local-search methods, such as gradient-free algorithms, gradient-based algorithms and Hessian-based algorithms (Nocedal and Wright 2006). Gradient-free algorithms are suitable for cases where the gradient is hard or impossible to obtain (Larson, Menickelly, and Wild 2019; Berahas et al. 2019). They are, however, generally inefficient and often their convergence rates depend on the dimensionality of the problem (Hare, Nutini, and Tesfamariam 2013). Hessian-based algorithms are able to escape saddle points but are usually too expensive to use (Nocedal and Wright 2006). Thus in practice, using gradient-based algorithms is the most typical choice, which is the case in FourierSAT. The performance of a specific gradient-based algorithm relies heavily on how fast the gradient can be computed. Since most computational work of CLS-based methods happens in the continuous optimizer (Jun Gu 1994), the efficiency of computing gradients is critical for the performance of CLS solvers.

GradSAT: from Walsh-Fourier Expansions to BDDs

We start this subsection by analyzing some limitations of FourierSAT. First, FourierSAT only accepts symmetric constraints.There are, however, useful constraints that cannot be reduced to symmetric ones, e.g., pseudo-Boolean constraints (PBs). Second, in FourierSAT, computing gradient costs $O(n)$ function evaluations (Kyrillidis et al. 2020), which can get expensive for constraints with more than a few hundred variables.

The issues above indicate that there might be better representations of Boolean constraints than Walsh-Fourier expansions. In the rest of this section, we discuss how our new framework, `GradSAT`, addresses those issues by choosing Binary Decision Diagrams (BDDs) as the representation to evaluate and differentiate the objective.

In order to use a new representation of Boolean constraints, we need to have a deeper understanding of the objective function. Theorem 4 in below, adapted from Theorem 3 in (Kyrillidis et al. 2020) indicates the objective value can be interpreted as a measure of the progress of the algorithm.

Definition 4. (Randomized rounding) *We define the randomized rounding function $\mathcal{R} : [-1,1]^n \to \{-1,1\}^n$ by*

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

for $i \in \{1, \cdots, n\}$, where $\mathbb{P}$ denotes the probability.

Definition 5. (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 $b \in \{-1,1\}^n$, with respect to the randomized-rounding function $\mathcal{R}$. We use $b \sim \mathcal{S}_a$ to denote that $b \in \{-1,1\}^n$ is sampled from the probability space $\mathcal{S}_a$.

Theorem 4. *Given a formula f and constraint weight function $w : C_f \to \mathbb{R}^+$, for a real point $a \in [-1,1]^n$, we have*

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

where $\mathbb{E}$ denotes expectation.

[4] In (Kyrillidis et al. 2020) the problem is to minimize instead of maximize, due to the difference of the definition of Boolean functions, which, however, does not bring about an essential difference.

For a discrete point $b \in \{-1, 1\}^n$, $F_{f,w}(b)$ is the sum of weights of all constraints satisfied by b. Therefore, Theorem 4 reveals that for a real point a, the value $F_{f,w}(a)$ is in fact the expected solution quality at a. Thus, a CLS approach can make progress in the sense of expectation as long as it increases the objective value. Compared with DLS solvers and coordinate descent-based CLS solvers that only flip one bit per iteration, a CLS method is able to "flip" multiple variables, though probably by a small amount, to make progress.

We now interpret Theorem 4 from the perspective of circuit-output probability.

Corollary 1. *With f, w, a as in Theorem 4, we have*

$$F_{f,w}(a) = \mathop{\mathbb{E}}_{b \sim \mathcal{S}_a}[F_{f,w}(b)] = \sum_{c \in C_f} w(c) \cdot \texttt{COP}(P_a, c),$$

where $P_a : X \to \mathbb{R}$ is the variable input probability function defined by $P_a(x_i) = \frac{1-a_i}{2}$ for all $i \in \{1, \cdots, n\}$.

Corollary 1 states that to evaluate the objective function, it suffices to compute the circuit-output probability of each constraint. Calculating the circuit-output probability, as a special case of literal weighted model counting, is #P-hard for general representations such as CNF formulas (Valiant 1979). Nevertheless, if a Boolean function is represented by a BDD B of size $S(B)$, then the circuit-output probability on it can be solved in time $O(S(B))$ by a probability assignment algorithm (Thornton and Nair 1994) shown in Alg. 2.

Lemma 1. (Thornton and Nair 1994) *Alg. 2 returns the circuit-output probability $\texttt{COP}(P, f)$, where $P \equiv \frac{1}{2}$ and runs in time $O(S(B))$ given the BDD B of f.*

Alg. 2 deals with the uniform variable probability, $P(x_i) = \frac{1}{2}$ for $i \in \{1, \ldots, n\}$. In the following we adapt it for admitting a general variable input probability function $P_a : X \to [0, 1]$.

To use the idea of Alg. 2, we need to construct the BDDs for the constraints of f. Note that since different constraints can share same sub-functions, equivalent nodes only need to be stored once. In BDD packages such as CUDD, this property is realized by a decision-diagram manager. After combining equivalent nodes, the BDD forest generated from all constraints forms a multi-rooted BDD (MRBDD). The size of the MRBDD can be significantly smaller than the sum of the size of each individual BDD when sub-function sharing frequently appears. Alg. 3 shows how to use ideas above to evaluate our objective $F_{f,w}$.

Theorem 5. *Alg. 3 returns the correct objective value $F_{f,w}(a)$ and runs in time O(S(B)), for the MRBDD B of f.*

Alg. 3 traverses the MRBDD in a top-down style while the same effect can be achieved by a bottom-up algorithm as well, as shown in Alg. 4. Similar ideas of traversing BDDs bottom-up have been used in BDD-based SAT solving (Pan and Vardi 2006) and weighted model counting (Dudek, Phan, and Vardi 2019).

Theorem 6. *Alg. 4 returns the correct objective value $F_{f,w}(a)$ and runs in time O(S(B)), for the MRBDD B of f.*

So far we have demonstrated how to evaluate the objective by BDDs. In the rest of this section we will show the combination of the bottle-up and top-down algorithms inspires an efficient algorithm for gradient computation.

Algorithm 2: The Probability-Assignment Algorithm

```
Input  : BDD $B$ of Boolean function $f$;
Output: COP$(P, f)$ with $P \equiv \frac{1}{2}$
1 Step 1. Assign probability 1 for the root node of $B$.
2 Step 2. If the probability of node $v = M[v]$, assign
    probability $\frac{1}{2}M[v]$ to the outgoing arcs from $v$.
3 Step 3. The probability $M[u]$ of node $u$ is the sum of
    the probabilities of the incoming arcs.
4 return $M[B.one]$
```

Algorithm 3: Objective Evaluation (Top-Down)

```
Input  : MRBDD $B$, real point $a \in [-1, 1]^n$,
         constraint weight function $w : C_f \to \mathbb{R}$.
Output: value of $F_{f,w}(a)$
1 Let $M_{TD} : B.V \to \mathbb{R}$ be the bottom-up messages
    and $M_{TD}[u] = 0$ for all the non-entry nodes
    $u \in B.V$.
2 for each constraint $c \in C_f$ do
3     $M_{TD}[B.entry(c)] = w(c)$
4 Sort all nodes of $B$ by topological order to list $L$.
5 for each node $v \in L$ do
6     $M_{TD}[v.T] += \frac{1-a[i_u]}{2} \cdot M_{TD}[v]$
7     $M_{TD}[v.F] += \frac{1+a[i_u]}{2} \cdot M_{TD}[v]$
8 return $M_{TD}[B.one]$
```

Algorithm 4: Objective Evaluation (Bottom-Up)

```
Input, Output: Same with Algorithm 3
1 Let $M_{BU} : B.V \to \mathbb{R}$ be the bottom-up messages
    and $M_{BU}[u] = 0$ for all nodes $u \in B.V$ except
    $M_{BU}[B.one] = 1$.
2 Sort all nodes of $B$ by topological order to list $L$.
3 for each node $v \in L$ do
4     for each node $u$ such that $u.T = v$ do
5         $M_{BU}[u] += \frac{1-a[i_u]}{2} \cdot M_{BU}[v]$
6     for each node $u$ such that $u.F = v$ do
7         $M_{BU}[u] += \frac{1+a[i_u]}{2} \cdot M_{BU}[v]$
8 return $\sum_{c \in C_f} (M_{BU}[B.entry(c)] \cdot w(c))$
```

Gradient is as Cheap as Evaluation in GradSAT

In this subsection, we propose a fast algorithm for computing the gradient, based on message passing on BDDs. The algorithm runs in $O(S(B))$, given B the MRBDD of formula f. Roughly speaking, computing the gradient is as cheap as evaluating the objective function.

The basic idea is to traverse the MRBDD twice, once top-down and once bottom-up. Then, enough information is generated for applying the differentiate operation on each BDD node. The gradient of a specific variable, say x_i, is obtained from all nodes associated with x_i. This idea is borrowed

Algorithm 5: Efficient Gradient Computation

```
Input  : Same with Algorithm 3
Output: Gradient vector $g \in \mathbb{R}^n$ at point $a$
1 Let $g$ be a zero vector with length $n$.
2 Run Algorithm 3 with $(B, a, w)$
3 Run Algorithm 4 with $(B, a, w)$
4 Collect $M_{TD}, M_{BU} : B.V \to \mathbb{R}$ from Alg. 3 and 4
5 for each non-terminal node $v \in B.V$ do
6     $g[i_v] += \frac{1}{2} \cdot M_{TD}[v] \cdot (M_{BU}[v.F] - M_{BU}[v.T])$
7 return $g$
```

from belief-propagation methods on graph model in probabilistic inference (Pearl 1988; Shafer and Shenoy 1990). Compared with FourierSAT, where the computation of the gradient for different variables has no overlap, Alg. 5 exploits the shared work between different variables.

Theorem 7. *Alg. 5 returns the gradient $g \in \mathbb{R}^n$ of $F_{f,w}$ at real point $a \in [-1, 1]^n$ correctly and runs in time $O(S(B))$, given the MRBDD B of f.* [5]

Roughly speaking, $M_{TD}[v]$ (Top-Down) has connection with the probability of a BDD node v being reached from the root, while $M_{BU}[v]$ (Bottom-Up) contains information of the circuit-output probability of the sub-function defined by the sub-BDD with root v. Algorithm 5 can be interpreted as applying the differentiation operation on each node.

The Versatility of GradSAT

Now that we have an efficient algorithm to compute the gradient for formulas with compact BDDs, the next natural question to ask is for what types of constraints can we enjoy small size BDDs. Here we list the following results.

Proposition 1. (Sasao and Fujita 1996) *Symmetric Boolean constraints with n variables admits BDDs with size $O(n^2)$.*

Proposition 2. (Aavani 2011) *Pseudo-Boolean constraints with n variables and magnitude of coefficients bounded by M admits BDDs with size $O(n \cdot M)$.*

Thus `GradSAT` can accept symmetric constraints and small-coefficient PBs, since they have reasonable BDD size.

It is well known that most of the local search-based incomplete SAT solvers can also work as a solution approach for the MaxSAT problem, by providing the "best found" truth assignment upon termination (Kautz, Sabharwal, and Selman 2009). Thus we also use `GradSAT` as a partial MaxSAT solver to solve Boolean optimization problems.

Implementation Techniques

Adaptive Constraint Weighting

When dealing with satisfaction problem, the constraint-weight function $w : C_f \to \mathbb{R}^+$ has a great impact on the performance of CLS method. In practice we find that an appropriate weight function can make it much faster for a continuous optimizer to reach a global maximum. How to predefine a static weight function for a given class of formulas is, however, still more art than science thus far.

[5]In fact, line 5-7 of Algorithm 5 can be integrated to Algorithm 4 once the top-down traverse is done. Thus computing the gradient costs traversing the MRBDD twice instead of three times. We use this trick in our implementation.

Algorithm 6: GradSAT

```
Input  : Boolean formula $f$ with constraint set $C_f$;
         Hyperparameters $r$ and $T$
Output: A discrete assignment $x \in \{-1, 1\}^n$
1 Build the MRBDD $B$ of $f$.
2 for $j = 1, \ldots, J$ do
3     Sample $x_0$ uniformly at random from $[-1, 1]^n$.
4     Set $w(c) = length(c)$ for all constraints $c$.
5     for $t = 1, \ldots, T$ do
6         Starting from $x_0$, search for a local maximum
          $x^*_{jt}$ of $F_{f,w}$ in $[-1, 1]^n$ with Alg. 3 for
          function value and Alg. 5 for gradient.
7         if $F_{f,w}(x^*_{jt}) = \sum_{c \in C_f} w(c)$ then return $x^*_{jt}$;
8         for $\forall c \in C_f$ that is not satisfied by $x^*_{jt}$ do
9             Set $w(c) = w(c) \cdot r$
10 return $x^*_{jt}$ with the highest $F_{f,w}(x^*_{jt})$
```

In CLS approaches for SAT solving, solving history is not leveraged if random restart is invoked every time after getting stuck in local optima. We are inspired from the Discrete Lagrange Method (Wah and Shang 1996) and the "break-out" heuristic (Morris 1993) in DLS to design the following dynamic weighting scheme. The constraint weights are initialized to be the length of the constraint. When a local optimum is reached, we multiply the weights of unsatisfied constraints by r and start from the same point once again. `GradSAT` only does random restart when T trials have been made for the same starting point, as shown in Alg. 6.[6]

A Portfolio of Optimizers and Parallelization

The specific optimizer to maximize the objective also influences the performance of a CLS solver. Different optimizers are good at different types of instances. `GradSAT` runs a portfolio of four different optimizers, including SLSQP (Kraft 1994), MMA (Svanberg 2002) from NLopt [7] and CG (Hestenes and Stiefel 1952), BFGS (Fletcher 1987) from Dlib (King 2009), on multiple cores and terminates when any of optimizers returns a solution.

Experimental Results

We aim to answer the following research questions:
RQ1. Is the cost for computing gradient reduced by using the BDD-based approach (Algorithm 5) ?
RQ2. What is the advantages and limitations of `GradSAT` on randomly generated hybrid Boolean formulas?

[6]We tuned r and T by running `GradSAT` on random 3-CNF benchmarks with $r \in \{1, 1.5, 2, 2.5, 3\}$ and $T \in \{1, 2, 4, 8, 16\}$ and choosing (r, T) that solves most cases. We got $r = 2, T = 8$.

[7]S. Johnson, The NLopt nonlinear-optimization package.

instance type	n	m	max/avg. constraint length	FourierSAT/s	GradSAT/s	Acceleration ($\times$)
3-CNF	500	2000	3/3	0.187	0.012	15.58
10-CNF	1000	32000	10/10	18.81	2.89	6.26
2-CNF+1 CARD	300	451	300/2.66	0.40	0.086	4.65
3-CNF+XORs	150	360	75/15	0.036	0.029	1.24

Table 2: Gradient speedup. FourierSAT/s and GradSAT/s denote the gradient cost in seconds. GradSAT significantly improves the efficiency of computing gradient.

RQ3. Can GradSAT perform well in solving discrete optimization problems, encoded by MaxSAT?

RQ4. Can MRBDD with nodes sharing significantly reduce the total number of nodes?

Solver Competitors. 1) CDCL solvers including Glucose, cryptoMiniSAT (CMS) and ML-enhanced solver MapleSAT; 2) DLS solvers, including WalkSATlm (Cai, Su, and Luo 2013), CCAnr and ProbSAT; 3) PB solvers including CDCL-based MiniSAT+ (Aavani 2011), Open-WBO (Martins, Manquinho, and Lynce 2014), NAPS (Sakai and Nabeshima 2015) and Conflict-Driven-Pseudo-Boolean-Search-based (CDPB) solver RoundingSAT (Elffers and Nordström 2018); 4) Partial MaxSAT solvers, including Loandra (Berg, Demirović, and Stuckey 2019), the best solver of MaxSAT Competition 2019 (incomplete track), WalkSAT (v56) and Mixing Method (Wang, Chang, and Kolter 2017), an SDP-based solver.

Benchmark 1: Hybrid random formulas consisting of disjunctive clauses, XORs, CARDs and PBs. We generate four types of satisfiable random hybrid benchmarks: CNF-XORs (360 instances), XORs-1CARD (270 instances), CARDs (360 instances) and PBs (720 instances) with variable number $n \in \{50, 100, 150\}$ [8]. When a constraint is not accepted by a traditional CNF solver, we use CNF encodings to resolve the issue. Specifically, we let pySAT (Ignatiev, Morgado, and Marques-Silva 2018) choose the CNF encoding for CARDs and PBs, while using the methods in (Li 2000) to encode XORs. Time limit is set to be 100s.

Benchmark 2: MaxSAT problems. We gathered 570 instances from the crafted (198 instances) and random track (377 instances) of MaxSAT Competition 2016-2019. Those instances encode random problems as well as combinatorial problems such as set covering and MaxCut. We set the time limit to 60s. We also used problems from the incomplete track of MaxSAT Competition 2019 as the industrial benchmarks (296 instances).

Implementation of GradSAT. We implement GradSAT in C++ [9]. Each experiment is run on an exclusive node in a Linux cluster with 16-processor cores at 2.63 GHz and 1 GB of RAM per processor. We generated five versions of GradSAT, including using single-core on each of the four optimizers (SLSQP, MMA, CG, BFGS) and using 16 cores on the portfolio, with each optimizer getting 4 cores.

RQ1. Table 2 lists the running time for computing gradient on several instances. Each running time is the average on 100 random points in $[-1, 1]^n$. The results indicate that GradSAT significantly accelerates the computation of gradient compared with FourierSAT.

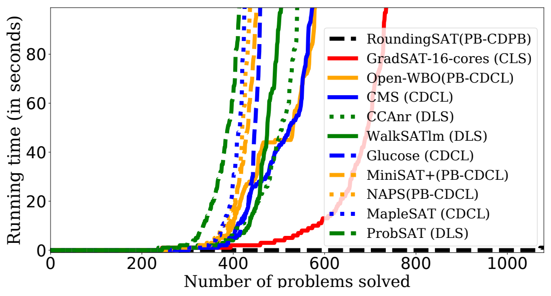

Figure 1: Results on random CARDs and PBs. GradSAT is second only to CDPB-based RoundingSAT.

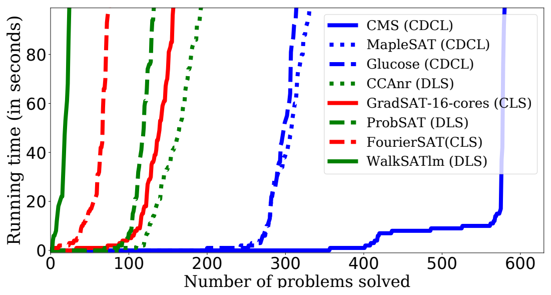

Figure 2: Results on random XORs-1CARD and CNF-XORs. CDCL solvers outperform local-search solvers.

RQ2. We classify the four types of benchmarks in Benchmark 1 into two groups: 1) PBs and CARDs; 2) XORs-1CARD and CNF-XORs; because solvers have similar performance on instances from the same group. We summarize the results in Figure 1 and 2. Figure 1 shows the results on CARDs and PBs instances. Though CDPB-based solver RoundingSAT easily solves all instances, GradSAT on 16 cores is able to solve 732 instances, more than all other CDCL, DLS SAT solvers and CDCL-based PB solvers. From Figure 2, we observe that on benchmarks involving XORs, CDCL solvers, especially CMS, perform better than local-search solvers. GradSAT on 16 cores solves 155 instances, only second to CCAnr (191) among all local search solvers. We conjecture that the solution space shattering

[8]A detailed description of benchmarks generation can be found in the supplemental material.

[9]Available at https://github.com/vardigroup/GradSAT

Benchmarks	Avg. total # nodes of indi. BDDs	Avg. # nodes of MRBDD	Avg. redu. ratio (×)
MaxSAT-Random	6683	2044	3.22
MaxSAT-Crafted	7435	3237	2.51
MaxSAT-Industrial	3428810	1548877	2.29
Hybrid SAT-CARDs	31714	31454	1.02
Hybrid SAT-PBs	602664	602393	1.00
Hybrid SAT-CNF-XORs	7254	5600	1.30
Hybrid SAT-XORs-1CARD	5976	5794	1.03

Table 3: Nodes sharing in MRBDD. The average reduction ratio is considerable on MaxSAT instances while unsatisfactory on hybrid SAT instances.

Methods	Avg. Score	#Win
GradSAT-Portfolio-16cores	0.970	465
GradSAT-BFGS-1core	0.954	293
GradSAT-MMA-1core	0.950	248
GradSAT-CG-1core	0.949	234
GradSAT-SLSQP-1core	0.937	206
WalkSAT	0.917	124
FourierSAT	0.908	108
Mixing Method	0.892	121
Loandra	0.874	41
VBS	1	575
VBS without GradSAT	0.990	254

Table 4: Results on MaxSAT instances. `GradSAT` implementations achieve better avg. score as well as #Win and improve the VBS.

(Dudek, Meel, and Vardi 2016; Pote, Joshi, and Meel 2019) brought by XORs might be the reason for the weakness of local search solvers in handling XORs.

RQ3. Table 4 showcases the results on MaxSAT benchmarks. We use the incomplete score (Berg, Demirović, and Stuckey 2019) and the number of instances where a solver provides the solution with the best cost (#Win) as the metric for evaluation. All five versions of `GradSAT` achieve better average score and #Win than other solvers. Moreover, `GradSAT` improves the score of Virtual Best Solver (VBS) with 0.01 and #Win with 321.

We also try `GradSAT` on large-size instances from the `industrial` MaxSAT problems, which contain both hard and soft clauses. But `GradSAT` is not able to give an answer for about $2/3$ of all instances. On instances which `GradSAT` successfully solves, an average score of 0.82 is achieved. We conjecture two reasons for the unsatisfactory performance of `GradSAT` on `industrial` benchmarks as follows. First, `GradSAT` sets the weight of hard clauses to be (#soft clauses + 1). It has been shown (Lei and Cai 2018) that this natural weighting performs poorly without exploiting the structure of partial MaxSAT problems. Second, industrial benchmarks contain a large number of variables, which also the non-stochastic DLS solver, e.g., GSAT, could hardly handle. Thus it is not surprising that GradSAT, as a proof-of-concept implementation demonstrating how to calculate the full gradients simultaneously, is less competitive. We leave handling industrial partial MaxSAT problems by CLS approaches as a future direction.

RQ4. Table 3 shows the total number of nodes of individual BDDs and the number of nodes of MRBDD for each type of benchmarks In order to allow more nodes sharing, we use the same variable order for all individual BDDS, corresponding to the original variable indices. We leave planning and applying a better variable order as a future direction. For all tracks of MaxSAT instances in CNF format, using MRBDD significantly reduced the number of nodes, yielding a average reduction ratio from 2.29 to 3.22. However, on non-CNF hybrid SAT instances, nodes sharing in MRBDD is negligible, probably due to relatively large constraint length.

Conclusion and Future Directions

In this paper, we proposed a new BDD-based continuous local search (CLS) framework for solving Boolean formulas consisting of hybrid constraints, including symmetric constraints and pseudo-Boolean constraints. The core of our approach is a novel algorithm for computing the gradient as fast as evaluating the objective function. We implemented our method as `GradSAT` on top of a portfolio of continuous optimizers. The experimental results demonstrate the power of `GradSAT` at handling CARD/PB constraints and small-size MaxSAT problems, while we also observe the limitations of our tool on large instances and XOR-rich instances.

Since for large instances, the cost for computing the gradient is still too high, a future direction is applying stochastic gradient descent to reduce the cost. Another future direction is to combine the advantages of CLS methods, i.e., the capability of "searching inside" and those of DLS methods, including "sideway walk", which searches the space quickly, in order to design stronger solvers. We also want to design better weighting mechanism as in (Lei and Cai 2018) to handle partial MaxSAT problems with hard clauses.

Acknowledgments

Work supported in part by NSF grants IIS-1527668, CCF-1704883, IIS-1830549, CCF-1907936, CNS-2003137, and an award from the Maryland Procurement Office.

References

Aavani, A. 2011. Translating Pseudo-Boolean Constraints into CNF. In Sakallah, K. A.; and Simon, L., eds., *Theory* *and Applications of Satisfiability Testing - SAT 2011*, 357–359. Berlin, Heidelberg: Springer Berlin Heidelberg.

Audemard, G.; and Simon, L. 2014. Lazy Clause Exchange Policy for Parallel SAT Solvers. In *SAT 2014*, 197–205. Cham. ISBN 978-3-319-09284-3.

Balint, A.; and Schöning, U. 2012. Choosing Probability Distributions for Stochastic Local Search and the Role of Make versus Break. In *SAT 2012*, 16–29. ISBN 978-3-642-31612-8.

Berahas, A. S.; Cao, L.; Choromanski, K.; and Scheinberg, K. 2019. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *arXiv preprint arXiv:1905.01332* .

Berg, J.; Demirović, E.; and Stuckey, P. J. 2019. Core-Boosted Linear Search for Incomplete MaxSAT. In Rousseau, L.-M.; and Stergiou, K., eds., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 39–56. Cham: Springer International Publishing. ISBN 978-3-030-19212-9.

Biere, A. 2008. PicoSAT Essentials. *JSAT* 4: 75–97.

Bogdanov, A.; Khovratovich, D.; and Rechberger, C. 2011. Biclique Cryptanalysis of the Full AES. In *ASIACRYPT 2011*. ISBN 978-3-642-25385-0.

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

Cai, S.; Luo, C.; and Su, K. 2015. CCAnr: A Configuration Checking Based Local Search Solver for Non-random Satisfiability. In Heule, M.; and Weaver, S., eds., *Theory and Applications of Satisfiability Testing – SAT 2015*, 1–8. Cham: Springer International Publishing. ISBN 978-3-319-24318-4.

Cai, S.; Su, K.; and Luo, C. 2013. Improving WalkSAT for Random k-Satisfiability Problem with k > 3. In *AAAI*.

Costa, M.-C.; de Werra, D.; Picouleau, C.; and Ries, B. 2009. Graph Coloring with Cardinality Constraints on the Neighborhoods. *Discrete Optimization* 6(4): 362 – 369. ISSN 1572-5286. doi:https://doi.org/10.1016/j.disopt.2009.04.005. URL http://www.sciencedirect.com/science/article/pii/S1572528609000231.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A Machine Program for Theorem-Proving. *Communications of the ACM* 5(7): 394–397.

Davis, M.; and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *Journal of the ACM (JACM)* 7(3): 201–215.

Dinur, I.; Regev, O.; and Smyth, C. 2005. The Hardness of 3-Uniform Hypergraph Coloring. *Combinatorica* 25(5): 519–535. ISSN 1439-6912. doi:10.1007/s00493-005-0032-4. URL https://doi.org/10.1007/s00493-005-0032-4.

Dixon, H. E.; Ginsberg, M. L.; Luks, E. M.; and Parkes, A. J. 2011. Generalizing Boolean Satisfiability II: Theory. *arXiv e-prints* arXiv:1109.2134.

Dudek, J. M.; Meel, K. S.; and Vardi, M. Y. 2016. Combining the k-CNF and XOR Phase-Transitions. In *IJCAI*.

Dudek, J. M.; Phan, V. H. N.; and Vardi, M. Y. 2019. ADDMC: Weighted Model Counting with Algebraic Decision Diagrams. *arXiv e-prints* arXiv:1907.05000.

Dudek, J. M.; and Vardi, M. Y. 2020. Parallel Weighted Model Counting with Tensor Networks. *arXiv e-prints* arXiv:2006.15512.

Eén, N.; and Sörensson, N. 2003. An Extensible SAT-Solver. In *SAT*, 502–518. ISBN 978-3-540-24605-3.

Elffers, J.; and Nordström, J. 2018. Divide and Conquer: Towards Faster Pseudo-Boolean Solving.

Fletcher, R. 1987. *Practical Methods of Optimization*. Number v. 2 in A Wiley-Interscience publication. Wiley.

Hare, W.; Nutini, J.; and Tesfamariam, S. 2013. A survey of non-gradient optimization methods in structural engineering. *Advances in Engineering Software* 59: 19 – 28. ISSN 0965-9978. doi:https://doi.org/10.1016/j.advengsoft.2013.03.001.

Hestenes, M. R.; and Stiefel, E. 1952. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards* 49: 409–435.

Hoos, H.; and Stützle, T. 2000. Local Search Algorithms for SAT: An Empirical Evaluation. *J. Automated Reasoning* 24: 421–481. doi:10.1023/A:1006350622830.

Hutter, F.; Tompkins, D. A. D.; and Hoos, H. 2002. Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT. In *CP*.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, 428–437. doi:10.1007/978-3-319-94144-8_26. URL https://doi.org/10.1007/978-3-319-94144-8_26.

Jain, P.; and Kar, P. 2017. Non-convex Optimization for Machine Learning. *arXiv e-prints* arXiv:1712.07897.

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

Kamath, A.; Karmarkar, N.; Ramakrishnan, K. G.; and Resende, M. C. 1990. Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research* 25: 43–58.

Kautz, H. A.; Sabharwal, A.; and Selman, B. 2009. Incomplete Algorithms. In *Handbook of Satisfiability*.

King, D. E. 2009. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research* 10: 1755–1758.

Kraft, D. 1994. Algorithm 733: TOMP–Fortran Modules for Optimal Control Calculations. *ACM Trans. Math. Softw.* 20(3): 262–281. ISSN 0098-3500. doi:10.1145/192115.192124. URL https://doi.org/10.1145/192115.192124.

Kyrillidis, A.; Shrivastava, A.; Vardi, M. Y.; and Zhang, Z. 2020. FourierSAT: A Fourier Expansion-Based Algebraic Framework for Solving Hybrid Boolean Constraints. In *AAAI 2020*.

Larson, J.; Menickelly, M.; and Wild, S. M. 2019. Derivative-free optimization methods. *Acta Numerica* 28: 287–404.

Lei, Z.; and Cai, S. 2018. Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 1346–1352. International Joint Conferences on Artificial Intelligence Organization.

Li, C. M. 2000. Integrating Equivalency Reasoning into Davis-Putnam Procedure. In *AAAI*, 291–296. AAAI Press. ISBN 0-262-51112-6. URL http://dl.acm.org/citation.cfm?id=647288.760210.

Liang, J. H. 2018. *Machine Learning for SAT Solvers*. Ph.D. thesis, University of Waterloo.

Marques-Silva, J. P.; and Sakallah, K. A. 1999. GRASP: A Search Algorithm For Propositional Satisfiability. *IEEE Transactions on Computers* 48(5): 506–521.

Martins, R.; Manquinho, V.; and Lynce, I. 2011. Exploiting Cardinality Encodings in Parallel Maximum Satisfiability. In *ICTAI*, 313–320.

Martins, R.; Manquinho, V.; and Lynce, I. 2014. Open-WBO: A Modular MaxSAT Solver,. In Sinz, C.; and Egly, U., eds., *Theory and Applications of Satisfiability Testing – SAT 2014*, 438–445. Cham: Springer International Publishing. ISBN 978-3-319-09284-3.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for Invariants in Local Search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, 321–326. AAAI Press. ISBN 0262510952.

Morris, P. 1993. The Breakout Method for Escaping from Local Minima. In *AAAI*.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. doi:10.1145/378239.379017. URL http://doi.acm.org/10.1145/378239.379017.

Nocedal, J.; and Wright, S. 2006. *Numerical optimization*. Springer Science & Business Media.

O'Donnell, R. 2014. *Analysis of Boolean Functions*. New York, NY, USA: Cambridge University Press. ISBN 1107038324, 9781107038325.

Pan, G.; and Vardi, M. Y. 2006. Symbolic Techniques in Satisfiability Solving. In Giunchiglia, E.; and Walsh, T., eds., *SAT 2005*, 25–50. Dordrecht: Springer Netherlands.

Pearl, J. 1988. Chapter 4 - Belief Updating by Network Propagation. In Pearl, J., ed., *Probabilistic Reasoning in Intelligent Systems*, 143 – 237. San Francisco (CA): Morgan Kaufmann. ISBN 978-0-08-051489-5.

Pote, Y.; Joshi, S.; and Meel, K. S. 2019. Phase Transition Behavior of Cardinality and XOR Constraints. In *IJCAI-19*.

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

Sakai, M.; and Nabeshima, H. 2015. Construction of an ROBDD for a PB-Constraint in Band Form and Related Techniques for PB-Solvers. *IEICE Transactions on Information and Systems* E98.D(6): 1121–1127.

Sasao, T.; and Fujita, M. 1996. *Representations of Discrete Functions*. USA: Kluwer Academic Publishers. ISBN 0792397207.

Selman, B.; Kautz, H.; and Cohen, B. 1999. Local Search Strategies for Satisfiability Testing. *Second DIMACS Implementation Challenge* 26.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, 440–446. AAAI Press. ISBN 0262510634.

Shafer, G.; and Shenoy, P. 1990. Probability propagation. *Ann Math Artif Intell* 2: 327–351. doi:10.1007/BF01531015.

Sheini, H. M.; and Sakallah, K. A. 2006. Pueblo: A Hybrid Pseudo-Boolean SAT Solver. *JSAT* 2: 165–189.

Soos, M.; Nohl, K.; and Castelluccia, C. 2009. Extending SAT Solvers to Cryptographic Problems. In *SAT*, 244–257.

Svanberg, K. 2002. A Class of Globally Convergent Optimization Methods Based on Conservative Convex Separable Approximations. *SIAM Journal on Optimization* 12(2): 555–573. doi:10.1137/S1052623499362822. URL https://doi.org/10.1137/S1052623499362822.

Thornton, M.; and Nair, V. 1994. Efficient Spectral Coefficient Calculation Using Circuit Output Probabilities. *Digital Signal Processing* 4(4): 245 – 254. ISSN 1051-2004. doi:https://doi.org/10.1006/dspr.1994.1024.

Valiant, L. G. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing* 8(3): 410–421. doi:10.1137/0208032.

Vardi, M. Y. 2014. Boolean satisfiability: theory and engineering. *Commun. ACM* 57(3): 5.

Wah, B.; and Shang, Y. 1996. A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems. *J Global Optimization* 12(1). doi:10.1023/A:1008287028851.

Wang, P.-W.; Chang, W.-C.; and Kolter, J. Z. 2017. The Mixing method: coordinate descent for low-rank semidefinite programming. *arXiv preprint arXiv:1706.00476* .