A Continuous Local Search Algorithms

A.1 FourierSAT

If the Walsh expansion is a k-th order polynomial, then the explicit form of the partial derivative will be a $(k-1)$-th order polynomial, in which the main operations are computing the ESPs of $k-1$ variables.

Algorithm A4: Differentiation with Explicit Form Gradient

Input: Boolean formula f with constraint set C and the current assignment x
Parameter: Fourier coefficient $\hat{f}$
Output: Gradient $\nabla_x F_f$

Initialize gradient $\nabla_x F_f = [0\ 0\ \cdots\ 0]$
for $c \in C$ **do**
  for $l \in c$ **do**
    Initialize empty sequence γ_c
    for $l' \in c \setminus l$ **do**
      $\gamma_c = \texttt{convolve}(\gamma_c, [1\ x_{l'}])$
    $\partial_{x_l} F_f += \hat{f}_c[:-1] \cdot \gamma_c$ ▷The constant term (last term) of the Walsh expansion becomes 0 in derivatives
Return $\nabla_x F_f$

Example 3 *(Explicit Gradient of the Walsh Expansion) For the constraint $x_1 + x_2 + x_3 + x_4 \geq 2$, the partial derivatives of Eq. 1 can be derived as:*

$$
\begin{aligned}
x_1' &= -\frac{3}{8}x_2x_3x_4 - \frac{1}{8}(x_2x_3 + x_2x_4 + x_3x_4) + \frac{1}{8}(x_2 + x_3 + x_4) + \frac{3}{8} = \frac{19}{64} \\
x_2' &= -\frac{3}{8}x_1x_3x_4 - \frac{1}{8}(x_1x_3 + x_1x_4 + x_3x_4) + \frac{1}{8}(x_1 + x_3 + x_4) + \frac{3}{8} = \frac{19}{64} \\
x_3' &= -\frac{3}{8}x_1x_2x_4 - \frac{1}{8}(x_1x_2 + x_1x_4 + x_2x_4) + \frac{1}{8}(x_1 + x_2 + x_4) + \frac{3}{8} = \frac{33}{64} \\
x_4' &= -\frac{3}{8}x_1x_2x_3 - \frac{1}{8}(x_1x_2 + x_1x_3 + x_2x_3) + \frac{1}{8}(x_1 + x_2 + x_3) + \frac{3}{8} = \frac{33}{64}
\end{aligned}
$$

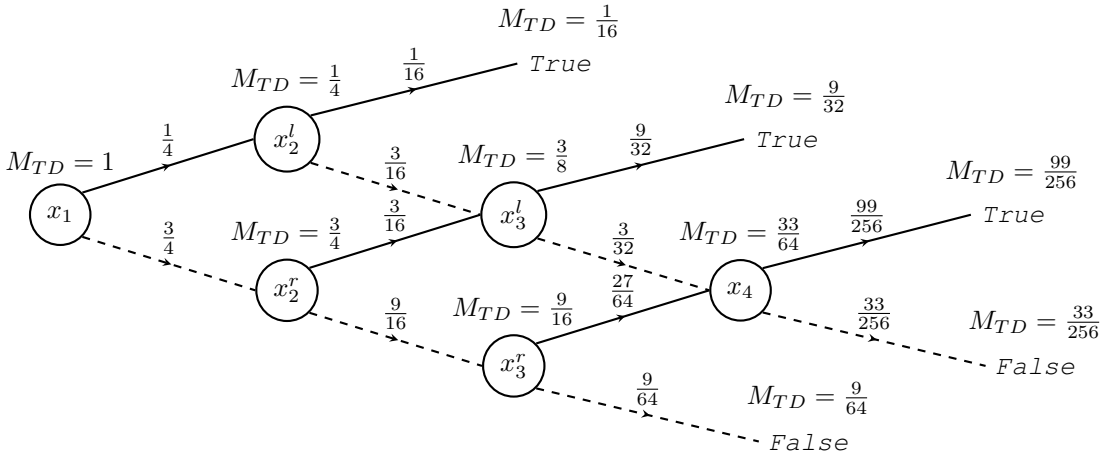
And hence, we will have $x' = (\frac{19}{64}, \frac{19}{64}, \frac{33}{64}, \frac{33}{64})$ The gradient guides all variables toward -1 to satisfy the constraint.

A.2 GradSAT

GradSAT evaluates the Boolean constraints with BDDs (Kyrillidis, Vardi, and Zhang 2021). By performing probabilistic inference on the BDD, the constraint can be evaluated and differentiated.

In the following, we give an example to demonstrate how the graph traversal in `GradSAT` works.

Example 4 *(Evaluation by Forward Traversal) Given a constraint $x_1 + x_2 + x_3 + x_4 \geq 2$, the BDD can be constructed (Eén and Sörensson 2006). Given $x = (\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$, the forward traversal of BDD can be described as:*



Algorithm A5: Differentiation with Belief Propagation

Input: Boolean formula f with constraint set C and the current assignment x
Parameter: BDDs B which encode the constraints
Output: Gradient $\nabla_x F_f$

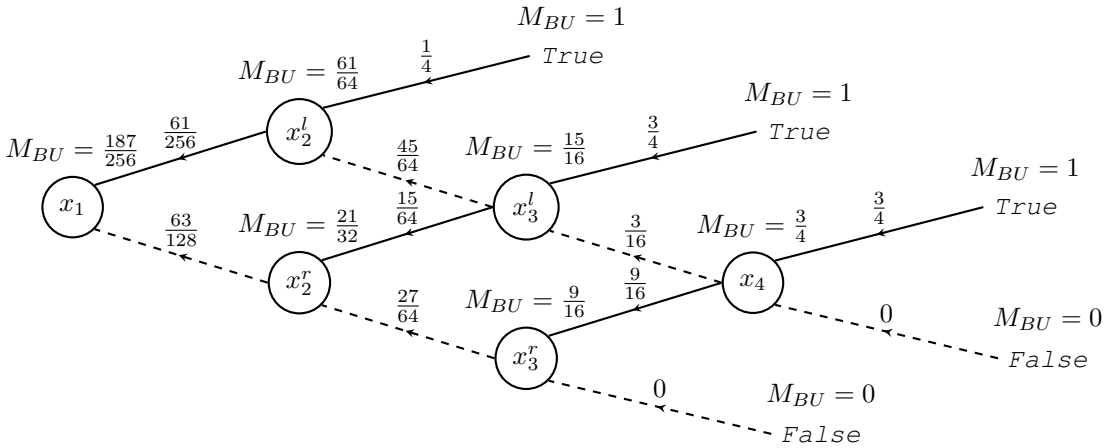
1: Initialize gradient $\nabla_x F_f = [0\ 0\ \cdots\ 0]$
2: Initialize the top-down and bottom-up messages M_{TD}, M_{BU} as 0. Except $M_{TD}(root) = M_{BU}(\texttt{true}) = 1$
3: **for** $c \in C$ **do**
4: Sort the nodes of B_c literal by literal to a list L
5: For each node maintain an index i to the literal
6: **for** each node $v \in L$ **do**
7: $M_{TD}[v.left] += \frac{1-x[i_v]}{2} M_{TD}[v]$
8: $M_{TD}[v.right] += \frac{1+x[i_v]}{2} M_{TD}[v]$
9: **for** each node $v \in L$ **do**
10: **for** each node u such that $u.left = v$ **do**
11: $M_{BU}[u] += \frac{1-x[i_u]}{2} M_{BU}[v]$
12: **for** each node u such that $u.right = v$ **do**
13: $M_{BU}[u] += \frac{1+x[i_u]}{2} M_{BU}[v]$
14: $\partial_{x[i_v]} F_f += M_{TD}[v](M_{BU}[v.left] - M_{BU}[v.right])$
15: **Return** $\nabla_x F_f$

By summing up the top-down message (M_{TD}) at the leaf nodes, we can find the probability of this constraint being satisfied or unsatisfied by:

$$P(true) = \sum M_{TD}(\texttt{true}) = \frac{1}{16} + \frac{9}{32} + \frac{99}{256} = \frac{187}{256}$$
$$P(false) = \sum M_{TD}(\texttt{false}) = \frac{33}{256} + \frac{9}{64} = \frac{69}{256}$$

Example 4 explains the forward traversal process (Lines 6-8 of Algorithm A5), where the top-down messages M_{TD}'s are accumulated. Note that $x[i_v] \in [-1, 1]$ encodes the truth value of a Boolean variable, as stated in Sec. 2.2. In `GradSAT`, the probabilistic inference is propagating the probabilities, in which the randomized rounding is defined by $\mathbb{P}[x_i = \texttt{true}] = \frac{1-x_i}{2}$ and $\mathbb{P}[x_i = \texttt{false}] = \frac{1+x_i}{2}$ (Def. 4 in (Kyrillidis, Vardi, and Zhang 2021)). As a result, M_{TD}'s are propagating downstream with $M_{TD}(v.left) += \mathbb{P}[v = \texttt{true}] M_{TD}(v)$ and $M_{TD}(v.right) += \mathbb{P}[v = \texttt{false}] M_{TD}(v)$. And hence the top-down messages at leaf nodes ($M_{TD}(\texttt{true})$, $M_{TD}(\texttt{false})$) encode the probability of the constraint being satisfied or unsatisfied.

Example 5 *(Differentiation by Backward Traversal) To compute the gradient, we need to perform backward traversal for the BDD.*



With $M_{BU}(\texttt{true})$ as 1 and $M_{BU}(\texttt{false})$ as 0, the backward traversal is propagating the probability of the constraint being satisfied to the root node (Lines 9-13 of Algorithm A5, Theorem 6 of (Kyrillidis, Vardi, and Zhang 2021)): $M_{BU}(root) = P(\texttt{true})$. If the bottom-up messages are initialized the other way around, i.e., with $M_{BU}(\texttt{true})$ as 0 and $M_{BU}(\texttt{false})$ as 1, $M_{BU}(root)$ should encode $P(\texttt{false})$.

After performing both forward and backward traversals, both top-down and bottom-up messages (M_{TD}, M_{BU}) have been obtained. Then the partial derivatives can be computed by [4]*:*

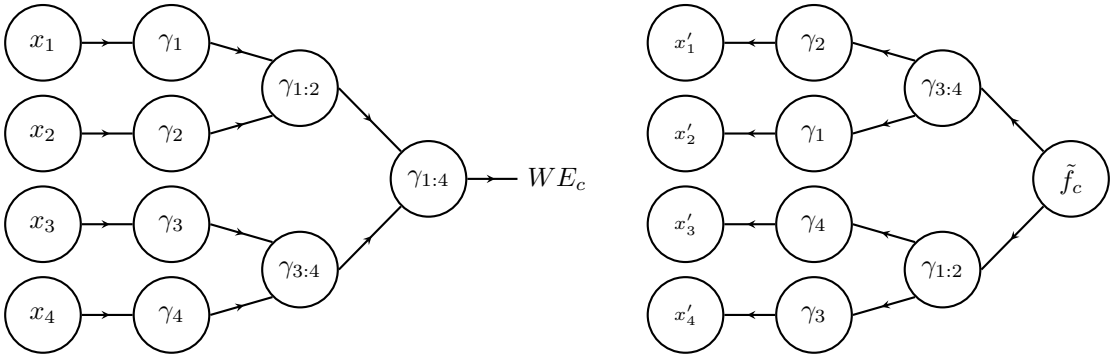
$$
\begin{aligned}
x_1' &= M_{TD}[x_1](M_{BU}[x_2^l] - M_{BU}[x_2^r]) = \frac{1}{2} \cdot 1 \cdot \left(\frac{21}{32} - \frac{61}{64}\right) = \frac{19}{64} \\
x_2' &= M_{TD}[x_2^l](M_{BU}[true] - M_{BU}[x_3^l]) + M_{TD}[x_2^r](M_{BU}[x_3^l] - M_{BU}[x_3^r]) \\
&= \frac{1}{4} \cdot \left(1 - \frac{15}{16}\right) + \frac{3}{4} \cdot \left(\frac{15}{16} - \frac{9}{16}\right) = \frac{1}{64} + \frac{9}{32} = \frac{19}{64} \\
x_3' &= \frac{1}{2} M_{TD}[x_3^l](M_{BU}[true] - M_{BU}[x_4]) + M_{TD}[false](M_{BU}[x_3^r] - M_{BU}[x_4]) \\
&= \frac{1}{4} \cdot \left(1 - \frac{3}{4}\right) + \frac{9}{16} \cdot \left(\frac{3}{4} - 0\right) = \frac{1}{16} + \frac{27}{64} = \frac{33}{64} \\
x_4' &= M_{TD}[x_4](M_{BU}[true] - M_{BU}[false]) = \frac{1}{2} \cdot \frac{33}{64} \cdot (1 - 0) = \frac{33}{64}
\end{aligned}
$$

Eventually, the resulting gradient is $x' = (\frac{19}{64}, \frac{19}{64}, \frac{33}{64}, \frac{33}{64})$.

A.3 FastFourierSAT

In this subsection, we give the graph view of Example 2.

Example 6 *(Evaluation and Differentiation Trace of Fourier Transform-based Implicit Gradient) Given a constraint* $x_1 + x_2 + x_3 + x_4 \geq 2$*, we can find the evaluation (left) and differentiation (right) traces for FastFourierSAT.*



The left graph is corresponding to step i.-iii. in Example 2. Following the evaluation trace from left to right, is equivalent to solving Eq. 6, Eq. 7, and Eq. 8. The right graph is corresponding to step iv. in Example 2. Following the differentiation trace from right to left, FastFourierSAT can compute the gradient implicitly.

A.4 Projected Gradient Descent

At each iteration in CLS, the optimizer tries to find a local minimum of the weighted objective function (Eq. 4). In gradient descent, using the first order method to update the variable x_i's can be described as:

$$x_{k+1} = x_k - \eta \sum_{c \in C} w_c \nabla F E_c(x_k)$$

where η is the step size of gradient descent and $\nabla(\cdot)$ is the is a differential operator.

In CLS-based SAT solving, the variables are bounding within $[-1, 1]^n$ due to Theorem 1. And hence the optimization problem becomes:

$$x^* = \arg\min \sum_{c \in C_f} w_c F E_c(x) + I_{[-1,1]^n}(x) \tag{10}$$

where $I_{[-1,1]^n(x)}$ is the indication function, which is equal to 0 when $x \in [-1, 1]^n$, otherwise equal to $+\infty$. Since the indication function is not differentiable, a proximal gradient fixed-point (PGFP) algorithm is utilized to solve Eq. 10 implicitly (Griewank

[4]Due to the difference in the definition of Boolean functions, the exact gradient computation method in (Kyrillidis, Vardi, and Zhang 2021) should modify Line 14 of Algorithm A5 to $\partial_{x[i_v]} F_f += \frac{1}{2} M_{TD}[v](M_{BU}[v.right] - M_{TD}[v.left])$. However, this does not bring about any essential difference since `FourierSAT` and `FastFourierSAT` is performing gradient descent, while `GradSAT` is performing gradient accent.

and Walther 2008). PGFP iteratively updates the solution and applies the proximal operator, which we denote as z and Π, respectively. Using the Moreau-Yosida regularization (Niculae and Blondel 2017), the proximity operator is given as:

$$\Pi(z) = \underset{x\in[-1,1]^n}{\arg\min} \|x - z\|_2^2$$

As we related the proximal gradient with gradient descent as $z = x_{k+1}$, the optimization problem in CLS-based SAT can be derived as:

$$x_{k+1} = \underset{x\in[-1,1]^n}{\arg\min} \|x - x_k - \eta \sum_{c\in C_f} \nabla F E_c(x_k)\|_2^2$$

Hence, the gradient descent in CLS can be described as Algorithm A6.

Algorithm A6: Using projected gradient descent to update the variables.

Input: The current assignment x, the evaluation function $F(\cdot)$ and gradient function $G(\cdot)$
Parameter: The current step size for gradient descent η
Output: The updated assignment x

for $j = 1, \cdots, T$ **do**
  $x, \eta = \text{lineSearch}\,(F(\cdot), G(\cdot), x, \eta)$ ▷Default method in (Blondel et al. 2022) is FISTA (Beck and Teboulle 2009)
  $x = \text{clip}(x, -1, 1)$ ▷PGFP in CLS
  if $\eta < 10^{-12}$ **then**
    Return x
Return x

`FourierSAT`, `GradSAT`, and `FastFourierSAT` apply different evaluation $F(\cdot)$ and differentiation $G(\cdot)$ methods to Algorithm A6. The line search in Algorithm A6 needs to iteratively call $F(\cdot)$ and $G(\cdot)$ until either the step size η is smaller than a threshold or the maximum iteration limit is reached, whichever occurs first.

B Generalizing Theorem 3 for MaxSAT

To generalize Theorem 3 for MaxSAT, we first define a formalism for problem-solving a multiset of weighted Boolean constraints.

Definition 5 *Consider a set of Boolean variables $x \in \{\texttt{true}, \texttt{false}\}^n$ occurring in the hybrid Boolean constraint set C. A literal l_i is either a variable $x_i \in x$ or its negation $\neg x_i$. A constraint $c_j \in C$ is described by a finite set of literals. Given a weighted constraint (c_j, w_j), w_j is the associated weight (positive numbers) indicating the penalty for falsifying c_j. A constraint is called hard if the weight is infinity, otherwise is called soft.*

$$\phi_{soft} = \{(c_1, w_1), \ldots, (c_m, w_m)\}$$
$$\phi_{hard} = \{(c_{m+1}, \infty), \ldots, (c_{m+m'}, \infty)\}$$
$$\phi = \phi_{soft} \cup \phi_{hard}.$$

where ϕ (ϕ_{soft}, ϕ_{hard}) is the weighted (soft, hard) constraint set. Then the optimal assignment $\boldsymbol{x}^*$ is an assignment that has minimal cost:

$$x^* = \arg\min \sum_{(c_i,w_i)\in\phi} w_i I_i(x),$$

where $I_i : x \to \{-1, 1\}$ is an indication function which will be -1 (resp. 1) if c_i is satisfied (resp. falsified).

Then Theorem 3 can be generalized as follows.

Theorem 6 *(Reduction) Given a weighted constraint set ϕ of a hybrid MaxSAT problem, the solution x^* is said to be valid if and only if*

$$F_\phi(x^*) \le w_{th} = \sum_{(c,w_c)\in\phi_{soft}} w_c - |\phi_{hard}| \left(1 + \sum_{(c,w_c)\in\phi_{soft}} w_c\right), \tag{11}$$

where w_{th} is the weight threshold to determine if the solution is valid. When a valid solution satisfies soft constraints with a weighted sum of w^*, the Ising spins σ^* encode the solution if and only if

$$F_\phi(x^*) = w_{th} - 2w^*. \tag{12}$$

When the Boolean formula is the conjunction of hard constraints, the problem is reduced to a hybrid SAT problem, which is satisfiable if and only if

$$\min F_\phi(x) = -|\phi_{hard}|. \tag{13}$$

C Walsh Coefficient of XOR Constraints

The Walsh coefficient of some constraints might have zero entries, making part of the computation trivial. By eliminating the trivial operations, the complexity can be further reduced. For example, the Walsh coefficient of an XOR constraint is $\hat{f}_{XOR} = [1\ 0\ \cdots\ 0]$, *i.e.,* the Walsh expansion only has the highest order term and all other entries are 0. Then, the conjugated Walsh coefficient becomes $\tilde{f}_{XOR} = \hat{f}_{XOR} \cdot W^{-1} = [1\ 1\ \cdots\ 1]$. By writing out and rearranging Eq. 8, the Walsh expansions of XOR constraints can be evaluated as:

$$WE_{XOR} = \frac{1}{k+1}\Big(\sum_{a=0}^{k}\omega^{ka} + \sum_{a=0}^{k}\omega^{(k-1)a}\sum_{b=0}^{k}x_b + \cdots + \sum_{a=0}^{k}\omega^{0a}\prod_{b=0}^{k}x_b\Big) \tag{14}$$

It can be proved by the Euler formula and trigonometric identities that, $\sum_{a=0}^{k}\omega^{ab} = 0,\ \forall b \in \mathbb{Z}^+$. By eliminating the trivial computations, `FastFourierSAT` will equivalently bypass the transform operators (*step i and iii*). Then, the complexity only needs to account for traversing the binary tree forward and backward (*step ii*).

Corollary 2 *(Reduction) For XOR constraints, the complexity of running Autodiff for Algorithm 2 can be reduced to $O(k)$.*

D Examples of Weight Adaptation

The essence of CLS-based hybrid SAT solving is using convex optimization to find the ground states of the non-convex energy landscape. Hence, the optimization results depend heavily on the initialization (Jain, Netrapalli, and Sanghavi 2013). A CLS approach can find a global optimum only if the neighboring convex set of the initialization point (*e.g.*, the red dot in Fig. 2a) includes a solution to the hybrid SAT formula, such that the local optimum searched by gradient descent will be the global optimum.

Hence, the solution quality of CLS can be significantly improved by using random restart. By leveraging the parallelism of GPUs, one can instantiate a batch of kernels to optimize the objective function with a batch of random initialization points. Instead of blind random restarts, we propose to incorporate weighting and rephasing heuristics into the parallel search to help `FastFourierSAT` explore the search space in a more efficient way.

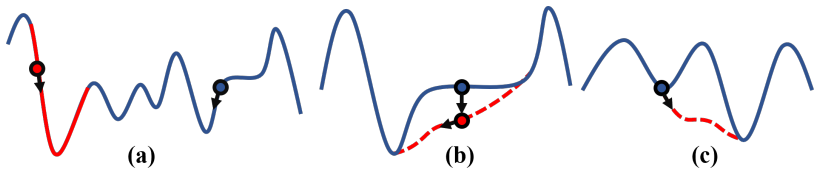


Figure 2: (a) Different initialization points will converge to different local optima. When the search trajectory is stuck at (b) a saddle point or (c) a local optimum, the weighting heuristic adapts the weights.

From the below examples, we can observe how the adaptive weight can escape the saddle points or local optima.

Example 7 *(Saddle points) Consider a simple example that only has two clauses $\{c_1 = x_1 \vee \neg x_2,\ c_2 = \neg x_1 \oplus x_2\}$, where the corresponding objective function is $F = -\frac{1}{2} + \frac{x_1}{2} - \frac{x_2}{2} - \frac{3x_1x_2}{2}$. At $\left(-\frac{1}{3}, \frac{1}{3}\right)$, the optimization stops at an inner saddle point (blue dot in Fig. 2b with unsatisfaction scores of $r_1 = 0$ and $r_2 = 1$. If we choose $\alpha = 0.6$ to adapt the weights. The weights will then adapt to $w_1' = 0.6$, $w_2' = 1$, resulting in a new objective function $F' = -\frac{3}{10} + \frac{3x_1}{10} - \frac{3x_2}{10} - \frac{13x_1x_2}{10}$. As a result, the gradient becomes $(\partial_{x_1}F', \partial_{x_2}F') = (-\frac{3}{4}, \frac{3}{4})$ (red dot in Fig. 2b).*

Example 8 *(Local optima) Consider another simple example $\{c_1 = x_1 \oplus x_2,\ c_2 = x_2 \oplus x_3,\ c_3 = x_3 \oplus x_4\}$, where the corresponding objective function is $F = x_1x_2 + x_2x_3 + x_3x_4$. At $(1, -1, -1, 1)$, the optimizer stops at the boundary (blue dot at Fig. 2c) with unsatisfaction scores of $r_1 = 0$, $r_2 = 1$, $r_3 = 0$. If we choose $\alpha = 0.6$ to adapt the weights. The weights will then adapt to $w_1' = w_3' = 0.6$, $w_2' = 1$, resulting in a new objective function $F' = \frac{3}{5}x_1x_2 + x_2x_3 + \frac{3}{5}x_3x_4$. As a result, the gradient becomes $(\partial_{x_1}F', \partial_{x_2}F', \partial_{x_3}F', \partial_{x_4}F') = (-\frac{3}{5}, -\frac{2}{5}, -\frac{2}{5}, -\frac{3}{5})$ and the search trajectory will move away from the boundary at $x_2 = x_4 = -1$.*

E Technical Proofs

E.1 Recap of Theorem 2

Let's denote two one-dimension sequences as $g_i = [x_{i0}, x_{i1}, \cdots, x_{in}]^T$ and $g_j = [x_{j0}, x_{j1}, \cdots, x_{jm}]^T$, where $n < m$. To prove Eq. 5 is to show the equivalence between $\mathcal{F}(g_i * g_j)$ and $\mathcal{F}(g_i)\,\mathcal{F}(g_j)$. We first show $\mathcal{F}(g_i * g_j)$:

$$g_i * g_j = \left[x_{i0}x_{j0}, \cdots, \sum_{q=0}^{k} x_{iq}x_{j(k-q)}, \cdots, x_{in}x_{jm}\right]^T$$

Then the convoluted sequence in the Fourier domain will be:

$$\mathcal{F}(g_i * g_j) = \left[\sum_{k=0}^{n+m}\sum_{q=0}^{k} x_{iq}x_{j(k-q)}, \cdots, \sum_{k=0}^{n+m}\omega^{pk}\sum_{q=0}^{k} x_{iq}x_{j(k-q)}, \cdots, \sum_{k=0}^{n+m}\omega^{(n+m)k}\sum_{q=0}^{k} x_{iq}x_{j(k-q)}\right]^T \tag{15}$$

Then we show $\mathcal{F}(g_i)\,\mathcal{F}(g_j)$:

$$\mathcal{F}(g_i) = \left[\sum_{q=0}^{n} x_{iq}, \cdots, \sum_{q=0}^{n}\omega^{pq}x_{iq}, \cdots, \sum_{q=0}^{n}\omega^{(n+m)q}x_{iq}\right]^T$$

$$\mathcal{F}(g_j) = \left[\sum_{q=0}^{m} x_{jq}, \cdots, \sum_{q=0}^{m}\omega^{pq}x_{jq}, \cdots, \sum_{q=0}^{m}\omega^{(n+m)q}x_{jq}\right]^T$$

Then the multiplied sequence in the Fourier domain will be:

$$\begin{aligned}&\mathcal{F}(g_i) \circ \mathcal{F}(g_j) \\ &= \left[\left(\sum_{q=0}^{n} x_{iq}\right)\left(\sum_{q'=0}^{m} x_{jq'}\right), \cdots, \left(\sum_{q=0}^{n}\omega^{pq}x_{iq}\right)\left(\sum_{q'=0}^{m}\omega^{pq'}x_{jq'}\right), \cdots, \left(\sum_{q=0}^{n}\omega^{(n+m)q}x_{iq}\right)\left(\sum_{q'=0}^{m}\omega^{(n+m)q'}x_{jq'}\right)\right]^T\end{aligned} \tag{16}$$

The p-th term in Eq. 16 can be written as:

$$\left(\sum_{q=0}^{n}\omega^{qp}x_{iq}\right)\left(\sum_{q'=0}^{m}\omega^{q'p}x_{jq'}\right) = \sum_{q=0}^{n}\sum_{q'=0}^{m}\omega^{(q+q')p}x_{iq}x_{jq'}$$

Consider $k = q + q'$:

$$\sum_{q=0}^{n}\sum_{q'=0}^{m}\omega^{(q+q')p}x_{iq}x_{jq'} = \sum_{k=0}^{n+m}\omega^{pk}\sum_{q=0}^{k} x_{iq}x_{j(k-q)}$$

As a result, the p-th term in Eq. 16 is equivalent to p-th term in Eq. 15. This result can be generalized to any p, and hence $\mathcal{F}(g_i * g_j)$ is equivalent to $\mathcal{F}(g_i) \circ \mathcal{F}(g_j)$.

Besides, the proof above also shows:

Corollary 3 *(Communitivity) With g and h as two one-dimension sequences. The linear convolutions ($*$) in space and the pointwise multiplications ($\circ$) in the Fourier domain are commutative.*

$$\mathcal{F}(g * h) \Leftrightarrow \mathcal{F}(h * g) \Leftrightarrow \mathcal{F}(g) \circ \mathcal{F}(h) \Leftrightarrow \mathcal{F}(h) \circ \mathcal{F}(g)$$

E.2 Proof of Corollary 1

Let's denote the one-dimension sequences in space as $g_i = [x_{i0}, x_{i1}, \cdots, x_{in}]^T$. To prove Eq. 3 is to show the equivalence between $\mathcal{F}(g_1 * \cdots * g_k)$ and $\mathcal{F}(g_1) \circ \cdots \circ \mathcal{F}(g_k)$.

Due to Corollary 3, the second term can be written as:

$$\begin{aligned}&\mathcal{F}(g_1) \circ \mathcal{F}(g_2) \circ \mathcal{F}(g_3) \circ \cdots \circ \mathcal{F}(g_k) \\ =&((\mathcal{F}(g_1) \circ \mathcal{F}(g_2)) \circ \mathcal{F}(g_3)) \circ \cdots) \circ \mathcal{F}(g_k) \\ =&((\mathcal{F}(g_1 * g_2) \circ \mathcal{F}(g_3)) \circ \cdots) \circ \mathcal{F}(g_k) \\ =&(\mathcal{F}(g_1 * g_2 * g_3) \circ \cdots) \circ \mathcal{F}(g_k)\end{aligned} \tag{17}$$

Eventually, Eq. 17 will be equivalent to the first term.

E.3 Proof of Theorem 3

Definition 4 defines the objective function of a Boolean formula F_f with a constraint set C represented with Walsh expansions. Due to Theorem 1, Eq. 4 can take value from $-\sum_{c\in C} w_c$ to $\sum_{c\in C} w_c$.

- "$\Rightarrow$": Suppose the Boolean formula is satisfiable and an assignment $a^* \in \{\texttt{True}, \texttt{False}\}^n$ satisfies all the constraints. Correspondingly, we will have a ground state $x^* \in [-1, 1]^n$ encodes a^*. For $c \in C$, $WE_c(x^*) = -1$ due to Theorem 1 and hence, $F_f(x^*) = -\sum_{c\in C} w_c$. Therefore, the minimum value of Eq. 4 is obtained.
- "$\Leftarrow$": Suppose the minimum value of Eq. 4 is attainable. Thus, $\exists x^*$ such that $F_f(x^*) = -\sum_{c\in C} w_c$. Since $WE_c(x^*)$ can only take value from -1 to 1, Eq. 4 can be minimal only if $WE_c(x^*) = -1, \forall c \in C$. Note that $WE_c(x^*) = -1$ encodes `True` of a constraint c, and hence all constraints in the constraint set C is satisfied.

E.4 Proof of Theorem 6

- "⇒": Suppose the Boolean formula is satisfiable and an assignment $a^* \in \{\texttt{True}, \texttt{False}\}^n$ satisfies all the hard constraints and some soft constraints with a weighted sum of w^*. Correspondingly, we will have $x^* \in [-1, 1]^n$ encodes a^*. For all the satisfied constraints c, $F_{\phi_c}(x^*) = -1$ due to Theorem 1.

$$\begin{aligned} F'_\phi(x) &= \sum_{(c,w_c)\in\phi_{soft,sat}} w_c F_{\phi_c}(x) + \left(1 + \sum_{(c,w_c)\in\phi_{soft}} w_c\right) \sum_{(c,w_c)\in\phi_{hard}} F_{\phi_c}(x) \\ &= -w^* - |\phi_{hard}| \left(1 + \sum_{(c,w_c)\in\phi_{soft}} w_c\right) \end{aligned} \tag{18}$$

  For all the falsified constraints, $F_{\phi_c}(x^*) = 1$ due to Theorem 1.

$$\begin{aligned} F''_\phi(\sigma) &= \sum_{(c,w_c)\notin\phi_{soft,sat}} w_c F_{\phi_c}(x) \\ &= \sum_{(c,w_c)\in\phi_{soft}} w_c - w^* \end{aligned} \tag{19}$$

  Hence, $F_\phi(x^*) = F'_\phi(x^*) + F''_\phi(x^*) = w_{th} - 2w^*$. Therefore, Eq. 12 is obtained.

- "⇐": Suppose Eq. 18 and Eq. 19 are attainable. Thus, $\exists\sigma^*$ such that $F'_\phi(x^*) = -w^* - |\phi_{hard}| \left(1 + \sum_{(c,w_c)\in\phi_{soft}} w_c\right)$ and $F''_\phi(x^*) = \sum_{(c,w_c)\in\phi_{soft}} w_c - w^*$.

  Since $F_{\phi_c}(x^*)$ can only take value from -1 to 1, Eq. 18 can be minimal only if $F_{\phi_c}(x^*) = -1, \forall(c, w_c) \in \phi_{soft,sat} \cup \phi_{hard}$. Note that $F_{\phi_c}(x^*) = -1$ encodes `True` of a constraint c, and hence all constraints in $\phi_{soft,sat} \cup \phi_{hard}$ is satisfied. Correspondingly, Eq. 18 can be maximal only if $F_{\phi_c}(x^*) = 1, \forall(c, w_c) \in \phi_{soft,fal}$. Since $F_{\phi_c}(x^*) = 1$ encodes `False`, all constraints in $\phi_{soft,fal}$ is falsified.

E.5 Proof of Theorem 4

In section 3.2 we have shown that the three-step process can evaluate the Walsh expansion of a symmetric Boolean constraint. Given a variable $x_i \in \{x_1, x_2, \cdots, x_k\}$, the ESPs can be computed as Eq. 5.

Step i is performing the Fourier transform on the sequences in right hand side (RHS) of Eq. 5. The overall computation can be simplified as the outer additions of a column vector and a row vector as in Eq. 6. Since the column vector has a length of $k + 1$ and the row vector has a length of k, $k(k + 1)$ additions are required. Hence the complexity of this step is in $O(k^2)$.

Step ii is performing reduce product along the row, in which the length is k and requires k multiplications for each. So there will be $k(k + 1)$ multiplications for all the rows in this step.

In the end, step iii uses Eq. 8 to evaluate the Walsh expansion. The vector-vector multiplication consists of $k + 1$ multiplications and k additions.

We can observe that the floating point operations of the overall procedure consist of $(k + 1)^2$ additions and $(k + 1)^2$ multiplication. Therefore, the complexity of this algorithm scales at $O(k^2)$.

E.6 Proof of Theorem 5

Example 2 and Example 6 showcase the gradient computation process in `FourierSAT`. The computation graph is constructed based on Algorithm 2. The Walsh expansion WE_c and the gradients x can be obtained by forward and backward traversal of the graph. In Appendix E.5 we have seen the complexity of forward traversal is $O(k^2)$.

For analyzing the backward traversal, we first write out all the terms Eq. 9 as:

$$
\begin{aligned}
\gamma_c' &= \frac{\partial WE_c}{\partial \gamma_c} = \tilde{f}_c \\
\gamma_{1:2}' &= \gamma_c' \circ \frac{\partial \gamma_c}{\partial \gamma_{1:2}} = \tilde{f}_c \circ \gamma_{3:4} \\
\gamma_{3:4}' &= \gamma_c' \circ \frac{\partial \gamma_c}{\partial \gamma_{3:4}} = \tilde{f}_c \circ \gamma_{1:2} \\
\gamma_1' &= \gamma_{1:2}' \circ \frac{\partial \gamma_{1:2}}{\partial \gamma_2} = \gamma_{1:2}' \circ \gamma_2 \\
\gamma_2' &= \gamma_{1:2}' \circ \frac{\partial \gamma_{1:2}}{\partial \gamma_1} = \gamma_{1:2}' \circ \gamma_1 \\
\gamma_3' &= \gamma_{3:4}' \circ \frac{\partial \gamma_{3:4}}{\partial \gamma_4} = \gamma_{3:4}' \circ \gamma_4 \\
\gamma_4' &= \gamma_{3:4}' \circ \frac{\partial \gamma_{3:4}}{\partial \gamma_3} = \gamma_{3:4}' \circ \gamma_3 \\
x_1' &= \frac{\partial \gamma_1}{\partial x_1} = \mathbb{1} \cdot \gamma_i' \\
x_2' &= \frac{\partial \gamma_2}{\partial x_2} = \mathbb{1} \cdot \gamma_i' \\
x_3' &= \frac{\partial \gamma_3}{\partial x_3} = \mathbb{1} \cdot \gamma_i' \\
x_4' &= \frac{\partial \gamma_4}{\partial x_4} = \mathbb{1} \cdot \gamma_i'
\end{aligned}
\tag{20}
$$

All the intermediate variables required for the computation at a certain node can be obtained from upstream. Line 1 of Eq. 20 (output node in Example 6) is the conjugated Fourier coefficient computed in the preprocessing stage and hence, no floating point operations are needed. Lines 2-7 (intermediate nodes) are point-wise multiplications of two vectors. Lines 8-11 (input nodes) are taking the sum of column vectors.

We can observe from Eq. 20 that, given a constraint with k literal, there will be $2k-2$ intermediate nodes and k input nodes. The computation at an intermediate node consists of $k+1$ multiplications. The computation at an input node consists of k additions. The floating point operations of the overall procedure consist of k^2 additions and $2k^2-2$ multiplications. Therefore, the complexity of this algorithm scales at $O(k^2)$.

E.7 Proof of Proposition 2

We consider the examples in Example 2 and Example 6 to show the proof but we generalize to any constraint c with k literals. The ideal execution time $O^*(\cdot)$ depends on the parallelizable component in the graph, which has a layer-wise topology.

First, we analyze the evaluation trace (Example 6, left), which is related to Appx. E.5. At the input layer (x_i's), the outer addition can run in $O^*(1)$ time. At the intermediate layers (γ_i's), the computation within the node is the point-wise multiplication of two vectors, which can be obtained concurrently. However, the overall time depends on the graph topology, *i.e.*, the depth of the binary tree is $\log k$. Thus, the best theoretical execution time to traverse the intermediate layers is $O^*(\log k)$. At the output layer, the vector-vector multiplication ($\tilde{f}_c \cdot \gamma_c$) runs in $O^*(\log k)$.

Next, we analyze the differentiation trace (Example 6, right), which is related to Appx. E.6. At the output layer, no floating point operations are required. The computation at the intermediate layers is similar to the left half of the graph, as well as the best theoretical execution time. At the input layer, the parallel sum reduction ($\mathbb{1} \cdot \gamma_i'$) runs in $O^*(\log k)$;

Therefore, the ideal execution time of differentiating the Walsh expansion of a symmetric Boolean constraint with k literals is $O^*(\log k)$.

E.8 Proof of Corollory 2

Lemma 1 *(Euler's formula) For any real number x:*

$$\exp(ix) = \cos(x) + i\sin(x)$$

Given $\omega^{ab} = \exp\left(\frac{i2\pi ab}{k+1}\right)$, it can be transformed into $\cos\left(\frac{2\pi ab}{k+1}\right) + i\sin\left(\frac{2\pi ab}{k+1}\right)$.

Lemma 2 *(Lagrange's trigonometric identities) Given* $\theta \not\equiv 0(\mod 2\pi)$*:*

$$\sum_{k=0}^{n} \sin k\theta = \frac{\cos\frac{\theta}{2} - \cos\left(\left(n+\frac{1}{2}\right)\theta\right)}{2\sin\frac{\theta}{2}}$$
$$\sum_{k=0}^{n} \cos k\theta = \frac{\sin\frac{\theta}{2} + \sin\left(\left(n+\frac{1}{2}\right)\theta\right)}{2\sin\frac{\theta}{2}}$$

Therefore, the coefficients in Eq. 14 can be written as:

$$\begin{aligned}\sum_{a=0}^{k} \omega^{ab} =& \sum_{a=0}^{k} \cos\left(\frac{2\pi ab}{k+1}\right) + i\sum_{a=0}^{k} \sin\left(\frac{2\pi ab}{k+1}\right) \\ =& \frac{\sin\frac{\pi b}{k+1} - \sin\left(\pi b + \frac{\pi b}{k+1}\right)}{2\sin\frac{\pi b}{k+1}} + i\frac{\cos\frac{\pi b}{k+1} + \cos\left(\pi b + \frac{\pi b}{k+1}\right)}{2\cos\frac{\pi b}{k+1}}\end{aligned} \tag{21}$$

When $b \in \mathbb{Z}^+$, Eq. 21 is 0, *i.e.*, all the computation related to $b \in \mathbb{Z}^+$ is trivial computation.

When $b = 0$, $\sum_{a=0}^{k} \omega^{ab} = k + 1$. As a result, Eq. 14 will be simplified as $WE_{XOR} = \prod_{b=0}^{k} x_b$, *i.e.*, purely multiplication. Thus, the complexity will be reduced to $O(k)$.