IEICE TRANS. FUNDAMENTALS, VOL.E105–A, NO.9 SEPTEMBER 2022

PAPER

A Satisfiability Algorithm for Deterministic Width-2 Branching Programs

Tomu MAKITA[†], *Nonmember*, **Atsuki NAGAO**[††a)], *Member*, **Tatsuki OKADA**[†], *Nonmember*, **Kazuhisa SETO**[†††b)], *and* **Junichi TERUYAMA**[††††c)], *Members*

SUMMARY A branching program is a well-studied model of computation and a representation for Boolean functions. It is a directed acyclic graph with a unique root node, some accepting nodes, and some rejecting nodes. Except for the accepting and rejecting nodes, each node has a label with a variable and each outgoing edge of the node has a label with a 0/1 assignment of the variable. The satisfiability problem for branching programs is, given a branching program with n variables and m nodes, to determine if there exists some assignment that activates a consistent path from the root to an accepting node. The width of a branching program is the maximum number of nodes at any level. The satisfiability problem for width-2 branching programs is known to be **NP**-complete. In this paper, we present a satisfiability algorithm for width-2 branching programs with n variables and cn nodes, and show that its running time is $\mathrm{poly}(n) \cdot 2^{(1-\mu(c))n}$, where $\mu(c) = 1/2^{O(c \log c)}$. Our algorithm consists of two phases. First, we transform a given width-2 branching program to a set of some structured formulas that consist of AND and Exclusive-OR gates. Then, we check the satisfiability of these formulas by a greedy restriction method depending on the frequency of the occurrence of variables.

key words: *satisfiability, width-k branching program, greedy restriction, moderately exponential time, exact algorithm*

1. Introduction

A branching program (BP) is a directed acyclic graph with a unique root node and some sink nodes. Each node except for the sink nodes is labeled by a variable, and each edge is labeled either 0 or 1 corresponding to a variable's value. Depending on the value of the output, each sink node is labeled either 0 or 1. A BP is deterministic if any node excluding the sink nodes has two edges: one edge is labeled 0, and the other edge is labeled 1. A BP computes a Boolean function naturally in the following way: it traces edges from the root node to a sink node corresponding to the value of the input. The bounded width BP is well studied. A width-k BP is a leveled BP and each level has at most k nodes. Barrington [2] showed that any function in $\mathbf{NC^1}$ can be computed by width-5 BPs of polynomial length. Thus, it suffices to show that some explicit function in **NP** has a super-polynomial lower bound on the length of width-5 BPs to prove $\mathbf{NP} \not\subseteq \mathbf{NC^1}$. Borodin, Dolev, Fich, and Paul [5] showed $\Omega(n^2/\log n)$ lower bounds for the half-exact function that outputs 1 if half of input is assigned 1. A first exponential lower bound of width-2 BPs for an explicit function in **NP** (actually in $\mathbf{NC^1}$) was shown by Yao [13]. Pseudorandom generators are known for width-2 and width-3 BPs. Bogdanov, Dvir, Verbin, and Yehudayoff [3] showed pseudorandom generators against width-2 BPs of polynomial length that read k bits of input at a time. A construction of pseudorandom generators against ordered read-once width-3 BPs of length n was shown by Meka, Reingold, and Tal [7].

In this paper, we study the satisfiability problem for bounded width BPs. The satisfiability problem for BPs (BP-SAT) is to determine whether there exists a consistent path from the root to a 1-sink. If any variable appears at most once in any path of a BP (called a read-once BP), by checking the reachability from the root to each 1-sink, we can easily check its satisfiability. However, when some variable appears twice, we cannot determine the satisfiability in the same way. Assume that, by solving the reachability of a given BP B, we get a path from the root to 1-sink that includes the edges $x_1 = 0$ and $x_1 = 1$. The assignment corresponding to this path does not satisfy B because $x_1 = 0$ and $x_1 = 1$ are inconsistent. In this way, the reachability for a given BP does not imply its satisfiability. Even if the width of a BP is bounded by 2, deciding its satisfiability becomes NP-complete. BP-SAT is a variant of Circuit Satisfiability (Circuit-SAT). The Circuit-SAT is, given a Boolean circuit, to determine whether there exists some assignment to the input variables such that the circuit outputs 1. The satisfiability problem for Boolean circuits in a class C (e.g. $\mathbf{AC^0}$, $\mathbf{ACC^0}$, $\mathbf{NC^1}$) is called C-SAT. Williams [12] showed that building an algorithm for C-SAT that is super-polynomially faster than the brute-force search implies $\mathbf{NEXP} \not\subseteq C$. By combining Barrington's theorem with this result, to prove that $\mathbf{NEXP} \not\subseteq \mathbf{NC^1}$, it is sufficient to develop an $O(2^{n-\omega(\log n)})$ time algorithm for width-5 BP-SAT.

As a first step toward this goal, we present the satisfiability algorithm for deterministic width-2 BPs with n variables and cn nodes.

Theorem 1. *There exists a deterministic satisfiability algo-*

Manuscript received September 21, 2021.
Manuscript revised December 28, 2021.
Manuscript publicized March 8, 2022.

[†]The authors are with Seikei University, Musashino-shi, 180-8633 Japan.
[††]The author is with Ochanomizu University, Tokyo, 112-8610 Japan.
[†††]The author is with Hokkaido University, Sapporo-shi, 060-0814 Japan.
[††††]The author is with University of Hyogo, Kobe-shi, 651-2197 Japan.
a) E-mail: a-nagao@is.ocha.ac.jp
b) E-mail: seto@ist.hokudai.ac.jp
c) E-mail: junichi.teruyama@gsis.u-hyogo.ac.jp
DOI: 10.1587/transfun.2021EAP1120

rithm for deterministic width-2 branching programs with n variables and cn nodes, and it runs in time $\mathrm{poly}(n)\cdot 2^{(1-\mu(c))n}$ *for* $\mu(c) = 1/2^{O(c\log c)}$.

An overview of our algorithm is as follows: First, we decompose a given width-2 BP into a family of sets of strict width-2 BPs (which contain exactly one 0-sink and one 1-sink) by the breadth-first search. Then, we transform each set of strict width-2 BPs to a formula with some structural properties such that it is satisfiable if and only if the given BP is satisfiable. Finally, we check the satisfiability of each formula by a greedy restriction algorithm depending on the frequency of the occurrence of the variables in a formula. Similar algorithms appear in the satisfiability of De Morgan formulas [10] and formulas over the full binary basis [11]. The analysis of our algorithm is based on a variant of Azuma's inequality in [6].

Related Work

There exist polynomial or moderately exponential time satisfiability algorithms for some restricted BPs. An ordered binary decision diagram (OBDD) is a BP that has the same permutation of variables in all paths from the root to any sink. Its satisfiability can be easily solved by checking the reachability from the root to 1-sink. A k-OBDD is a k-layered OBDD and all layers are OBDDs with the same permutation of variables. For any constant k, its satisfiability can be decided in polynomial time shown by Bollig, Sauerhoff, Sieling, and Wegener [4]. A k-indexed binary decision diagram (k-IBDD) is the same as a k-OBDD, except that each layer may have a different permutation of variables. A k-IBDD-SAT is known to be NP-complete when $k \geq 2$ [4]. Nagao, Seto, and Teruyama [8] proposed a satisfiability algorithm for k-IBDD with cn edges, and its running time is $O(2^{(1-\mu_k(c))n})$, where $\mu_k(c) = 1/O((\log c)^{2^{k-1}-1})$. They [9] also presented $O(2^{(1-1/4^{k-1})n})$ time satisfiability algorithm for syntactic read-k-times BPs. Chen, Kabanets, Kolokolova, Shaltiel, and Zuckerman [6] presented $O(2^{n-\omega(\log n)})$ satisfiability algorithm for general BPs with $o(n^2)$ nodes. In addition, the hardness of BP-SAT implies the hardness of the Edit Distance and Longest Common Subsequence problem [1].

Paper Organization

The remainder of this paper is organized as follows. In Sect. 2, we provide the notation and definitions. In Sect. 3, we provide a transforming algorithm from a width-2 BP to a set of formulas over AND and Exclusive-OR operations with some structural properties and a satisfiability algorithm for its formula.

2. Preliminaries

For a set S, $|S|$ denotes the cardinality of S. Let $X = \{x_1, \ldots, x_n\}$ be a set of Boolean variables.

A *branching program* (*BP*), denoted by $B = (V, E)$, is a rooted directed acyclic multigraph. The *length* of BP B is the length of the longest path in BP, denoted by $\ell(B)$. A BP has a unique root node r, and sink nodes without outgoing edges. We call a sink node labeled by **0** (resp. **1**) 0-sink (resp. 1-sink). A BP has at least one 0-sink, and at least one 1-sink. Each node except for the sink nodes is labeled from X. We call node v an x_i*-node* when v's label is x_i. Each edge $e \in E$ has a label 0 (*0-edge*) or 1 (*1-edge*). Figure 1(a)–(c) are examples of BPs.

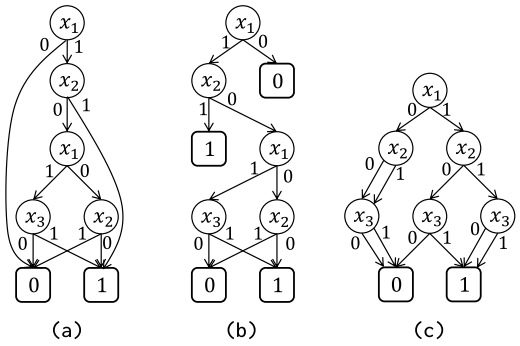

Fig. 1 General, leveled, and strict BPs.

A BP is *leveled* if, for any node v, all paths from the root r to v have the same length. For a leveled BP, we say that a node v is at level d if the length of the shortest path from r to v is d. Note that, at least one sink node appears at level $\ell(B)$ and the other sink nodes can appear at any level except the root node. For all i ($0 \leq i < \ell(B)$), an edge leaving a node at level i except for the sink nodes ends at a node at level $i+1$. Figure 1(b) and (c) are leveled BPs, but Fig. 1(a) is not. A leveled BP is *strict* if it has exactly one 1-sink and one 0-sink. Figure 1(c) is a strict BP, but Fig. 1(b) is not. The *width* of a leveled BP is the maximum number of nodes at any level. If the width of BP is k, we call it width-k BP. Figure 1(b) is a width-2 BP and Fig. 1(c) is a width-3 BP. For a width-2 BP B, when B contains two nodes u and v at the same level, we call u (v, resp.) the *sibling* of v (u, resp.). A BP B is *deterministic* if the outgoing edges of each node except for the sink nodes in B are exactly one 0-edge and one 1-edge. Otherwise, B is *nondeterministic*. In this paper, any BP is a deterministic width-2 BP unless otherwise stated.

For a BP B on X, each input $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ *activates* all α_i-edges leaving the x_i-nodes in B, where $1 \leq i \leq n$. A *computation path* is a path from the root r to a 0-sink or a 1-sink using only activated edges. Note that, for each input $\alpha \in \{0,1\}^n$, the computation path of a deterministic BP exists uniquely. A BP B outputs 0 (resp. 1) if the computation path reaches a 0-sink (resp. 1-sink). In this way, a BP *represents* a Boolean function $\{0,1\}^n \to \{0,1\}$. For example, all three BPs in Fig. 1 represent the same Boolean function. A BP is satisfiable if there exists an assignment α such that B outputs 1. The task of BP-SAT is, given a BP, to determine if it is satisfiable. Any CNF formula can be represented by a width-2 BP as follows. Let F be a CNF with s clauses $C_1, C_2, \ldots, C_s$, respectively. First, we construct the strict width-2 BP B_i for each C_i ($1 \leq i \leq s$) (see the left

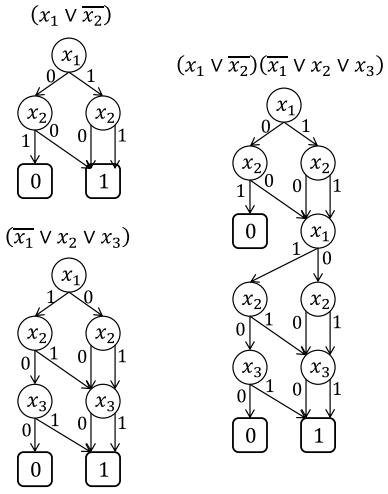

Fig. 2 Clauses and a CNF to width-2 BPs.

side in Fig. 2). Next, for each i ($1 \leq i < s$), we concatenate B_i with B_{i+1} replacing the 1-sink of B_i with the root of B_{i+1} (see the right side in Fig. 2). Then, we obtain the width-2 BP B corresponding to F. As this fact and CNF-SAT is **NP**-complete, the satisfiability problem for width-2 BPs is also **NP**-complete.

A Boolean function $\{0,1\}^n \to \{0,1\}$ can be represented as a *formula*. A formula is a rooted binary tree in which each leaf is labeled by a *literal* (which is either a variable x, or its negation $\bar{x}$), or a constant from $\{0,1\}$. We also use x^1 and x^0 as a positive literal x and a negative literal $\bar{x}$, respectively. Each internal node in a formula is called a *gate*, and is labeled by a binary function. The *size* of a formula f, denoted by $L(f)$, is defined as the number of literals in its leaves. It is known that formulas of polynomial size and width-5 branching programs with a polynomial number of nodes compute the same class of Boolean functions [2]. For a formula f, a *subformula* of f is defined as a formula which is a subtree in f. The *frequency* of a variable x in f, denoted by $\mathrm{freq}(f,x)$, is the total number of positive literal x and negative literal $\bar{x}$ appearing in the leaves of f. Let $\mathrm{var}(f)$ denote the set of variables in f. For any formula f, any set of variables $\{x_{i_1},\ldots,x_{i_k}\}$ and any constant $a_1,\ldots,a_k \in \{0,1\}$, we denote by $f|_{x_{i_1}=a_1,\ldots,x_{i_k}=a_k}$ the formula obtained from f by assigning the value a_j to each x_{i_j}. In this paper, we use only AND ($\wedge$) and Exclusive-OR ($\oplus$) gates in a formula.

3. Algorithm and Analysis

In this section, we give an algorithm for solving width-2 BP-SAT. Our algorithm consists mainly of two phases: (1) Transforming a given width-2 BP to a set of formulas with $\wedge$ and $\oplus$ gates. (2) Checking the satisfiability of each formula in the set obtained in phase 1.

3.1 Transformation of Width-2 BPs

We present an algorithm to transform a given width-2 BP B to a set of formula F. Let s be the number of 1-sink nodes in B. First, we decompose B into s BPs $B_1, B_2, \ldots, B_s$. Let the levels of s 1-sink nodes be $\ell_1, \ell_2, \ldots, \ell_s$ ($\ell_1 < \ell_2 < \cdots < \ell_s$), respectively. For each i ($1 \leq i \leq s$), let u_i be the sibling of 1-sink at ℓ_i and let u_0 be the root node of B. Note that u_i must be neither 0-sink nor 1-sink except for u_s. For each i, B_i consists of node u_{i-1} as the root node and all nodes at level from $\ell_{i-1}+1$ to ℓ_i except u_i, and all incoming edges to u_i are changed to connect a new 0-sink node. Thus, we have the set S of BPs $\{B_1, B_2, \ldots, B_s\}$, and B is satisfiable if and only if some B_i is satisfiable. Then, we decompose each width-2 BP $B_i \in S$ to a set of strict width-2 BPs in a similar way by considering the levels of 0-sink nodes in B_i. For each i, we obtain the set of strict width-2 BPs $\{B_{i,1}, B_{i,2}, \ldots, B_{i,s_i}\}$, where s_i is the number of 0-sinks of B_i, and B_i is satisfiable if and only if all $B_{i,j}$ are satisfiable. These operations are done by Decomposition as Algorithm 1.

Algorithm 1: Decomposition(B)

Input: A width-2 BP B
Output: A set of strict width-2 BPs T
Let s be the number of 1-sinks in B.
Compute levels $\ell_1, \ell_2, \ldots, \ell_s$ of 1-sinks in B by breadth-first search.
Let $S = \emptyset$ and $\ell_0 = 0$.
for $i = 1, \ldots, s$ **do**
  Let u_i be the sibling of the 1-sink at level ℓ_i.
  Let B_i be a copy of B from level ℓ_{i-1} to level ℓ_i except for the 1-sink at level ℓ_{i-1}.
  if $i < s$ **then**
    Create new 0-sink node w.
    Change all incoming edges to u_i of B_i to connect to w.
  Add B_i to the set S.
Let $T = \emptyset$
for *each* $B_i \in S$ **do**
  Let s_i be the number of 0-sinks in B_i.
  Compute levels $\ell_{i,j}$ ($j = 1, \ldots, s_i$) of 0-sinks in B_i by breadth-first search.
  Let $T_i = \emptyset$ and $\ell_{i,0} = 0$.
  for $j = 1, \ldots, s_i$ **do**
    Let $u_{i,j}$ be the sibling of the 0-sink at level $\ell_{i,j}$.
    Let $B_{i,j}$ be a copy of B_i from level $\ell_{i,j-1}$ to level $\ell_{i,j}$ except for the 0-sink at level $\ell_{i,j-1}$.
    if $j < s_i$ **then**
      Create new 1-sink node w.
      Change all incoming edges to $u_{i,j}$ of $B_{i,j}$ to connect to w.
    Add $B_{i,j}$ to the set T_j.
  Add T_i to T.
return T

Finally, we transform each strict width-2 BP to a formula that consists of $\wedge$ and $\oplus$ gates. The idea of transformation is based on the proof of Theorem 1 shown by Borodin et al. [5]. Let us suppose that $\ell(B_{i,j}) = \ell > 1$. The last two levels ($\ell - 1$ and ℓ) of $B_{i,j}$ should fit one of three cases in Fig. 3. We create a part of the formula from $B_{i,j}$ corresponding to the case in Fig. 3 and a new strict width-2 BP with length $\ell - 1$ by removing the nodes at ℓ level and replacing u and v with 0-sink and 1-sink, respectively. We continue this

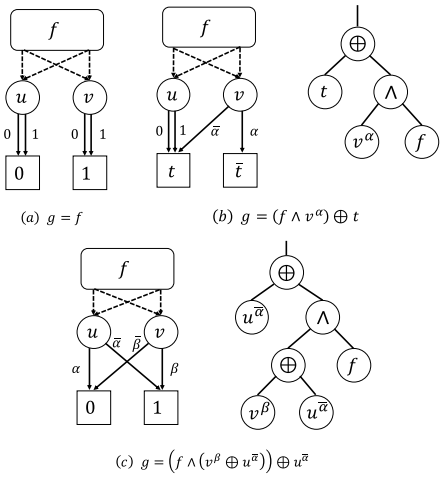

Fig. 3 Case analysis.

Algorithm 2: MAKEFORMULA(B)

Input: A strict width-2 BP B
Output: A formula f which represents the same function as B
if $\ell(B) = 1$ **then**
  return a pos/neg literal for the label with the root node of B.
Set u, v, t, α and β such that they correspond to one of three cases in Fig. 3.
Remove from B all outgoing edge of u and v.
Replace u and v with 0-sink and 1-sink, respectively.
$f \leftarrow$ MAKEFORMULA(B).
if *Case (a) holds* **then**
  return f
else if *Case (b) holds* **then**
  return $(f \wedge v^{\alpha}) \oplus t$
else # *Case (c) holds*
  return $(f \wedge (v^{\beta} \oplus u^{\bar{\alpha}})) \oplus u^{\bar{\alpha}}$

Algorithm 3: TRANSFORMATION(B)

Input: A width-2 BP B
Output: A set of formulas F
Let $T = \emptyset$.
$T \leftarrow$ DECOMPOSITION(B)
Let $F = \emptyset$.
for $i = 1, \ldots, |T|$ **do**
  Let $f_i = 1$.
  for *each* $B_{i,j} \in T_i$ **do**
    $g_{i,j} \leftarrow$MAKEFORMULA($B_{i,j}$)
    $f_i \leftarrow f_i \wedge g_{i,j}$
  Add f_i to F.
return F

operation recursively until we have a BP of length one, which corresponds to a formula with one literal. MAKEFORMULA as Algorithm 2 shows the detail of the transformation from a strict width-2 BP to a structured formula.

Combining DECOMPOSITION and MAKEFORMULA, we obtain TRANSFORMATION as Algorithm 3 that can transform a given width-2 BP to a set of some structured formulas. We claim as follows.

Lemma 1. *Given a width-2 BP with n variables and m nodes,* TRANSFORMATION *runs in time $O(m)$ and outputs a set of formulas. The total size of the output formulas is at most* $1.5m$.

Proof. Case (c) in Fig. 3 produces three leaves of a formula from two nodes of a width-2 branching program. Thus, the size of the formula is 1.5 times as large as the number of nodes of a given width-2 BP. The other cases do not increase the size of the formula. Therefore, the total size of the output formulas is at most $1.5m$. The rest of the proof is to estimate the running time of TRANSFORMATION. Letting ℓ be the length of a given BP, DECOMPOSITION runs in time $O(\ell) = O(m)$ since $\ell < m$ holds. For each strict width-2 BP with length ℓ', MAKEFORMULA runs in time $O(\ell')$. Because the sum of length of all strict width-2 BPs is $\ell < m$, the total running time of MAKEFORMULA is also $O(m)$. Thus, TRANSFOMATION runs in time $O(m)$ and this completes the proof. □

3.2 Satisfiability Checking of Formulas

The rest of our task is to check the satisfiability of each formula f in set F obtained by TRANSFORMATION in the previous section. The basic operation of checking is the greedy restriction which picks up the most frequently appearing variable x in formula f and checking the satisfiability of two formulas $f|_{x=0}$ and $f|_{x=1}$.

After the assignment to variable x, we call the procedure SIMPLIFY to reduce the size of a formula by applying rules to eliminate constants, redundant literals, and gates. See Algorithm 4. This procedure is almost the same one in [11], but we skip the operation of replacing $1 \oplus f'$ by a negation of a subformula f'. The reason is that De Morgan's laws for a negation of f' replaces an AND gate with an OR gate and it does not preserve the formula that consists of AND and Exclusive-OR gates. For any formula g, it is easy to see that SIMPLIFY runs in time polynomial in the size of g and the resulting formula computes the same function as g.

From MAKEFORMULA and SIMPLIFY, we obtain the following structural lemma about formulas.

Lemma 2. *Let f be a formula in the set created by* TRANSFORMATION. *For any set of variables $\{x_{i_1}, \ldots, x_{i_k}\}$ and any constant $a_1, \ldots, a_k \in \{0, 1\}$, $g =$ SIMPLIFY$(f|_{x_{i_1}=a_1,\ldots,x_{i_k}=a_k})$ satisfies one of the following cases.*

1. *There exists a variable x such that* $\mathrm{freq}(g, x) \geq L(g)/|\mathrm{var}(g)| + 1/4$.
2. *There exists a variable x whose parent is $\wedge$, and* $\mathrm{freq}(g, x) \geq L(g)/|\mathrm{var}(g)|$.
3. *There exists an $x \oplus y$ and its parent is $\wedge$, and* $\mathrm{freq}(g, x) = \mathrm{freq}(g, y) \geq L(g)/|\mathrm{var}(g)|$. *Let v be the sibling of $x \oplus y$ and g^v be a subformula with root v.*
   a. *There exists a variable z other than x and y in g^v.*
   b. *There exist no variable except x and y in g^v.*
4. *The number of variables that connect $\wedge$ whose parent*

Algorithm 4: SIMPLIFY(f)

Input: A formula f
Output: A formula f which represents the same function as the input formula
while *Until there is no decrease in size of* f **do**
  if $0 \wedge f'$ *(*$1 \wedge f'$*, resp.) occurs as a subformula, where* f' *is any formula* **then**
    Replace this subformula by 0 (f', resp.)
  if $y \wedge f'$ *occurs as a subformula, where* f' *is a formula and* y *is a literal* **then**
    Replace all occurrences of y in f' by 1 and all occurrences of $\bar{y}$ by 0.
  if $0 \oplus f'$ *occurs as a subformula, where* f' *is any formula* **then**
    Replace this subformula by f'
  if $1 \oplus y$ *occurs as a subformula, where* y *is a literal or a constant* **then**
    Replace this subformula by $\bar{y}$
  if $y \oplus y$ *(*$y \oplus \bar{y}$*, resp.) occurs as a subformula, where* y *is a literal* **then**
    Replace this subformula by 0 (1, resp.)
return f

$\oplus$*, or consist of* $x \oplus y$ *whose parent is* $\wedge$ *is at most* $|var(g)|/2$.

5. g *is a conjunction of formulas that consist of only* $\oplus$ *gates.*

Proof. Note that the average frequency of g is $L(g)/|\mathrm{var}(g)|$. Let us assume that cases 1–3 are false. Since case 1 is false, all the variable appears less than $L(g)/|\mathrm{var}(g)| + 1/4$ times in the formula. Let an integer γ be $\lceil L(g)/|\mathrm{var}(g)| \rceil$ and we have $L(g)/|\mathrm{var}(g)| \leq \gamma < L(g)/|\mathrm{var}(g)| + 1/4$. This inequality is led by the assumption that case 1 is false. We divide the variable set $\mathrm{var}(g)$ into two sets $X_1 = \{x \mid \mathrm{freq}(g, x) = \gamma\}$ and $X_2 = \{x \mid \mathrm{freq}(g, x) \leq \gamma - 1\}$.

Next we discuss the structural property of g. Cases (b) and (c) in Fig. 3 imply that if there exist some $\oplus$s that have an $\wedge$ as their child, then at least one child of such $\wedge$ must be a literal or $x \oplus y$. Since cases 2 and 3 are false, (i) all variables that connect $\wedge$-gate belong to X_2 and (ii) for each $x \oplus y$ whose parent is $\wedge$-gate, at least one of variables (x, y) belong to X_2. This implies that the number of variables that connect $\wedge$ whose parent is $\oplus$, or consist of $x \oplus y$ whose parent is $\wedge$ is at most $2|X_2|$. We show that $|X_2| \leq |\mathrm{var}(g)|/4$ holds. Since the size of g is the sum of the frequencies of all variables, $L(g) \leq \gamma|X_1| + (\gamma - 1)|X_2| = \gamma|\mathrm{var}(g)| - |X_2|$ holds. By the definition of γ, $\gamma < L(g)/|\mathrm{var}(g)| + 1/4$, that is, $L(g) > (\gamma - 1/4)|\mathrm{var}(g)|$. Thus, we have

$$|X_2| \leq \gamma|\mathrm{var}(g)| - L(g) < \gamma|\mathrm{var}(g)| - \left(\gamma - \frac{1}{4}\right)|\mathrm{var}(g)| = \frac{|\mathrm{var}(g)|}{4}.$$

Therefore, if there exist some $\oplus$s that have an $\wedge$ as their child, case 4 occurs.

Otherwise, there exists no $\oplus$ that has an $\wedge$ as its child. This means that any child of every $\oplus$ is an $\oplus$ or a literal, or a constant. Thus, g is a conjunction of $\oplus$s. □

Algorithm 5: EVALFORMULA(f, k)

Input: A formula f and a constant k
Output: True (if f is satisfiable) or False (otherwise)
SIMPLIFY(f)
if *Case 5 in Lemma 2 holds,* **then**
  Check the satisfiability of f by Gaussian Elimination.
  if f *is satisfiable* **then**
    return True
  else
    return False
if $|\mathrm{var}(f)| < k/2$, **then**
  Check the satisfiability of f by brute-force search.
  if f *is satisfiable* **then**
    return True
  else
    return False
if *Case 4 in Lemma 2 holds,* **then**
  Let X' be a set of variables that connect $\wedge$ whose parent $\oplus$, or consist of $x \oplus y$ whose parent is $\wedge$.
  Check the satisfiability of f by brute-force search on X' and Gaussian Elimination.
  if f *is satisfiable* **then**
    return True
  else
    return False
Let x be the most frequent variable in f.
if *Case 1 or Case 2 in Lemma 2 holds,* **then**
  $\mathrm{val}_0 \leftarrow$ EVALFORMULA($f|_{x=0}, k$)
  $\mathrm{val}_1 \leftarrow$ EVALFORMULA($f|_{x=1}, k$)
  return val_0 **or** val_1
else # Case 3 in Lemma 2 holds
  Let y be the sibling of x.
  Let v be the sibling of $x \oplus y$.
  Let f^v be the subformula of f with the root v.
  if *Case 3(a) holds,* **then**
    $g \leftarrow f|_{x=0}$
    $\mathrm{val}_{00} \leftarrow$ EVALFORMULA($g|_{y=0}, k$)
    $\mathrm{val}_{01} \leftarrow$ EVALFORMULA($g|_{y=1}, k$)
    $h \leftarrow f|_{x=1}$
    $\mathrm{val}_{10} \leftarrow$ EVALFORMULA($h|_{y=0}, k$)
    $\mathrm{val}_{11} \leftarrow$ EVALFORMULA($h|_{y=1}, k$)
    return val_{00} **or** val_{01} **or** val_{10} **or** val_{11}
  else if y *appears in* f^v **then**
    $\mathrm{val}_0 \leftarrow$ EVALFORMULA($f|_{x=0}, k$)
    $\mathrm{val}_1 \leftarrow$ EVALFORMULA($f|_{x=1}, k$)
    return val_0 **or** val_1
  else
    $\mathrm{val}_0 \leftarrow$ EVALFORMULA($f|_{y=0}, k$)
    $\mathrm{val}_1 \leftarrow$ EVALFORMULA($f|_{y=1}, k$)
    return val_0 **or** val_1

We propose the satisfiability algorithm EVALFORMULA as Algorithm 5 that determines the satisfiability of formulas created by TRANSFORMATION and the satisfiability algorithm W2BP-SAT as Algorithm 6 which determines the satisfiability of deterministic width-2 BP-SAT. It is easy to see the correctness of EVALFORMULA. Our algorithm is a simple branch and bound, thus it can check the satisfiability of g correctly.

Let g be a formula satisfying Lemma 2. EVALFORMULA terminates the branching in the following three cases: (1) If g satisfies case 5 in Lemma 2, we solve a system of at

Algorithm 6: W2BP-SAT(B)

Input: A width-2 BP B
Output: Yes (if B is satisfiable) or No (otherwise)
Let $F = \emptyset$.
$F \leftarrow$ TRANSFORMATION(B)
for *each* $f_i \in F$ **do**
  Let $c = L(f_i)/|\mathrm{var}(f_i)|$.
  Let $k = |\mathrm{var}(f_i)|/2(2\sqrt{2}c)^{10c}$.
  if EVALFORMULA(f_i, k) **then**
    return Yes (Satisfiable)
return No (Unsatisfiable)

most cn linear equations using the Gaussian Elimination. Because the Gaussian Elimination solves a system of m linear equations in time $O(m^3)$, the time complexity of this case is bounded by $O(c^3n^3)$. (2) If g has at most $k/2$ variables (k is a parameter fixed in Algorithm 6), by a brute-force search for all variables, we can compute the satisfiability of g in time $O(2^{k/2})$. (3) If g satisfies case 4, we select a set X' of variables that connect $\wedge$ whose parent $\oplus$, or consist of $x \oplus y$ whose parent is $\wedge$. By a brute-force search on X', we have a system of at most cn linear equations. Thus, we can compute the satisfiability of g in time $O(c^3n^3 2^{|\mathrm{var}(g)|/2})$.

If the above cases do not occur, cases 1–3 must happen by Lemma 2. For these cases, the algorithm restricts one variable or two variables step-by-step and reduces the size of the formula non-trivially at each step. EVALFORMULA picks up a variable x which is the most frequent variable in g. For case 1, when we restrict variable x, EVALFORMULA always eliminates at least $L(g)/|\mathrm{var}(g)| + 1/4$ leaves.

For case 2, when we restrict variable x, EVALFORMULA always eliminates at least $L(g)/|\mathrm{var}(g)|$ leaves. Furthermore, if x takes value 0 (1, resp.) and the parent of a literal x^0 (x^1, resp.) is an $\wedge$ gate, the sibling node of x is eliminated. Thus, the size of the formula are reduced by at least $L(g)/|\mathrm{var}(g)|+1$ with a probability of at least 1/2 if we randomly restrict variable x.

For case 3(a), there exists a variable z in f^v other than x and y. In this case, we randomly restrict the variables x and y. Since x and y appear at least $L(g)/|\mathrm{var}(g)|$ times, we always eliminate at least $2L(g)/|\mathrm{var}(g)|$ leaves. Furthermore, since $x \oplus y$ takes value 0 with a probability of 1/2, and the parent of $\oplus$ is an $\wedge$ gate, the sibling node of $\oplus$ can be eliminated with probability 1/2. Then, by eliminating at least one extra variable z, we can eliminate at least $2L(g)/|\mathrm{var}(g)| + 1$ leaves with probability 1/2.

For case 3(b), there is no variable other than x and y in f^v. In this case, if there exists y in f^v, we randomly restrict the variable x. Then, both $f^v|_{x=0}$ and $f^v|_{x=1}$ have only the variable y. By SIMPLIFY, EVALFORMULA eliminates at least one y. Thus, we can eliminate at least $L(g)/|\mathrm{var}(g)| + 1$ leaves with probability 1/2. Otherwise, (if there does not exist y in f^v,) we can eliminate at least $L(g)/|\mathrm{var}(g)| + 1$ leaves with probability 1/2 by interchanging x and y in the above argument.

To analyze the running time of EVALFORMULA, we make

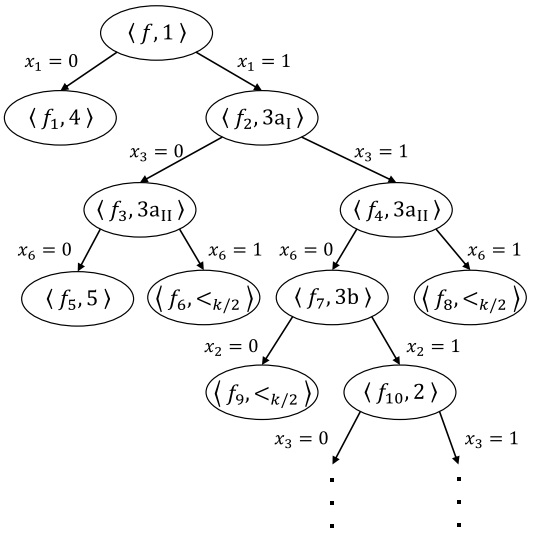

Fig. 4 Example of a computation tree.

a *computation tree* for a process of adaptive restriction in EVALFORMULA. A computation tree is a rooted binary tree and each node is labeled by a pair of a formula g and a symbol s in $\{1, 2, 3a_I, 3a_{II}, 3b, 4, 5, <_{k/2}\}$ that corresponds to which case happens on g, denoted by $\langle g, s\rangle$. See Fig. 4. The formula for the label of the root is the original input formula f. Let p be a node in the computation tree and g be a formula in the label of p. If g satisfies cases 1, 2, or 3(b), then p is labeled with $\langle g, 1\rangle$, $\langle g, 2\rangle$, or $\langle g, 3b\rangle$, respectively. Node p has two children: One is labeled with the formula SIMPLIFY($g|_{x=0}$) and the other is labeled with SIMPLIFY($g|_{x=1}$), where x is the restricted variable assigned by EVALFORMULA.

If g satisfies 3(a), then p is labeled with $\langle g, 3a_I\rangle$ and has also two children p_0 and p_1: p_0 has the label $\langle g|_{x=0}, 3a_{II}\rangle$ and p_1 has $\langle g|_{x=1}, 3a_{II}\rangle$, where x is the first restricted variable assigned by EVALFORMULA. Moreover, p_0 (p_1, resp.) has two children: One child has the formula SIMPLIFY($g|_{x=0,y=0}$) (SIMPLIFY($g|_{x=1,y=0}$), resp.) and the other has SIMPLIFY($g|_{x=0,y=1}$) (SIMPLIFY($g|_{x=1,y=1}$), resp.), where y is the second restricted variable. If g satisfies cases 4 or 5, or $\mathrm{var}(g) < k/2$ holds, then p is a leaf and labeled with $\langle g, 4\rangle$ or $\langle g, 5\rangle$, or $\langle g, <_{k/2}\rangle$.

We can see that any path from the root to a leaf represents a sequence of restrictions in cases 1–3. For a node p in the computational tree, we call the *depth* of p the length of a path from the root to p. Note that, if a formula g exists in the node of the depth d, then $|\mathrm{var}(g)| \le |\mathrm{var}(f)| - d$ holds.

Consider the computation tree divided virtually into layers of height 2, which means that at each layer, there are exactly two variables being restricted. Consider a node at the top of one layer; let g be the formula labeling the node, and suppose g is over $\mathrm{var}(g)$ variables with size $L(g)$. Let g' be the new formula after adaptively restricting two variables (at the bottom of the layer). Then, we have the following bounds on the size of g'.

Lemma 3. *It holds that* $L(g') \leq L(g)\left(1 - \frac{2}{|\mathrm{var}(g)|}\right)$. *Moreover, if* $|\mathrm{var}(g)| \geq 11$, *with probability at least 1/2,*

$$L(g') \leq L(g)\left(1 - \frac{2}{|\mathrm{var}(g)|}\right)^{1+\frac{|\mathrm{var}(g)|}{5L(g)}}.$$

Proof. For any layer, without loss of generality, we assume that we assign two variables x and y in this order. Let g'' be a formula after restricting variable x.

First, we prove the first inequality. If g satisfies case 3(a), then $\mathrm{freq}(g, x) = \mathrm{freq}(g, y) \geq L(g)/|\mathrm{var}(g)|$ holds. By restricting variables x and y, we reduce the size of formula g by at least $\mathrm{freq}(g, x) + \mathrm{freq}(g, y)$. Thus, we have

$$L(g) - L(g') \geq 2L(g)/|\mathrm{var}(g)|.$$

This implies that the first inequality holds. Next, let us suppose that g satisfies cases 1, 2 or 3(b). By restricting variable y, we can reduce the size of the formula by at least $\mathrm{freq}(g'', y)$, then we have

$$L(g) - L(g') \geq L(g) - L(g'') + \mathrm{freq}(g'', y).$$

By the behavior of EvalFormula, we have

$$L(g) - L(g'') \geq \mathrm{freq}(g, x) \geq \frac{L(g)}{|\mathrm{var}(g)|} \tag{1}$$

and

$$\mathrm{freq}(g'', y) \geq \frac{L(g'')}{|\mathrm{var}(g)| - 1}. \tag{2}$$

Therefore,

$$\begin{aligned}
&L(g) - L(g'') + \mathrm{freq}(g'', y)\\
\geq &L(g) - L(g'') + \frac{L(g'')}{|\mathrm{var}(g)| - 1}\\
= &\frac{|\mathrm{var}(g)| - 2}{|\mathrm{var}(g)| - 1} \cdot (L(g) - L(g'')) + \frac{L(g)}{|\mathrm{var}(g)| - 1}\\
\geq &\frac{|\mathrm{var}(g)| - 2}{|\mathrm{var}(g)| - 1} \cdot \frac{L(g)}{|\mathrm{var}(g)|} + \frac{L(g)}{|\mathrm{var}(g)| - 1}\\
= &\frac{L(g)}{|\mathrm{var}(g)| - 1}\left(\frac{|\mathrm{var}(g)| - 2}{|\mathrm{var}(g)|} + 1\right) = \frac{2L(g)}{|\mathrm{var}(g)|}.
\end{aligned} \tag{3}$$

Thus, the first inequality holds. The rest of the proof is the case that g is a formula $h|_{z=0}$ or $h|_{z=1}$, where h is a formula that satisfies case 3(a), and z and x are variables picked up for restriction. Becasue EvalFormula does not apply Simplify to g, we have $|\mathrm{var}(h)| = |\mathrm{var}(g)| + 1$ and

$$L(g) \leq L(h) - \frac{L(h)}{|\mathrm{var}(h)|}.$$

Therefore,

$$\frac{L(h)}{|\mathrm{var}(h)|} \geq \frac{L(g)}{|\mathrm{var}(h)| - 1} = \frac{L(g)}{|\mathrm{var}(g)|}$$

holds. Since we have $\mathrm{freq}(g, x) = \mathrm{freq}(h, x) \geq L(h)/|\mathrm{var}(h)|$,

$$\mathrm{freq}(g, x) \geq \frac{L(g)}{|\mathrm{var}(g)|}$$

holds and this means Eq. (1) holds, too. Eq. (2) also holds and then Eq. (3) occurs the same way as the above case. Thus, the proof for the first inequality is complete.

To prove the second inequality holds with a probability of at least $1/2$, it suffices to show the following lemma.

Lemma 4. *EvalFormula reduces the size of the formula by at least* $2L(g)/|\mathrm{var}(g)| + 2/5$ *with a probability of at least 1/2 by restricting two variables.*

Lemma 4 implies that

$$\begin{aligned}
L(g') &\leq L(g) - \left(2\frac{L(g)}{|\mathrm{var}(g)|} + \frac{2}{5}\right)\\
&= L(g)\left[1 - \frac{2}{|\mathrm{var}(g)|}\left(1 + \frac{|\mathrm{var}(g)|}{5L(g)}\right)\right]\\
&\leq L(g)\left(1 - \frac{2}{|\mathrm{var}(g)|}\right)^{1+\frac{|\mathrm{var}(g)|}{5L(g)}}
\end{aligned}$$

holds with the probability at least 1/2. The last inequality is led by the fact that $1 - ax \leq (1 - x)^a$ holds for any $a \geq 1$ and $0 \leq x \leq 1$.

The rest of the proof is to prove Lemma 4. If formula g satisfies case 3(a), then the size of the formula is reduced by at least $2L(g)/|\mathrm{var}(g)| + 1$ with the probability of at least 1/2 by assigning two variables x and y. If case 2 or case 3(b) appears to assign y, then we can eliminate one additional literal with a probability of at least 1/2. Thus, in such a case, the size of the formula reduced by assigning two variables is at least $2L(g)/|\mathrm{var}(g)| + 1$.

If case 2 or case 3(b) appears for to assign x, then $L(g) - L(g'') \geq L(g)/|\mathrm{var}(g)| + 1$ holds with a probability of at least 1/2. In such a case, the size of the formula reduced by assigning two variables is at least

$$\begin{aligned}
&\frac{|\mathrm{var}(g)| - 2}{|\mathrm{var}(g)| - 1} \cdot (L(g) - L(g'')) + \frac{L(g)}{|\mathrm{var}(g)| - 1}\\
\geq &\frac{|\mathrm{var}(g)| - 2}{|\mathrm{var}(g)| - 1} \cdot \left(\frac{L(g)}{|\mathrm{var}(g)|} + 1\right) + \frac{L(g)}{|\mathrm{var}(g)| - 1}\\
= &\frac{2L(g)}{|\mathrm{var}(g)|} + 1 - \frac{1}{|\mathrm{var}(g)| - 1} \geq \frac{2L(g)}{|\mathrm{var}(g)|} + \frac{9}{10},
\end{aligned}$$

where the last inequality is obtained by $|\mathrm{var}(g)| \geq 11$.

In the rest of the proof is when case 1 appears twice. In this case, $L(g) - L(g'')$ and $L(g'') - L(g')$ should be at least $L(g)/|\mathrm{var}(g)| + 1/4$ and $L(g'')/(|\mathrm{var}(g)| - 1) + 1/4$. Therefore,

$$\begin{aligned}
&L(g) - L(g')\\
= &L(g) - L(g'') + L(g'') - L(g')\\
\geq &L(g) - L(g'') + \frac{L(g'')}{|\mathrm{var}(g)| - 1} + \frac{1}{4}\\
= &\frac{|\mathrm{var}(g)| - 2}{|\mathrm{var}(g)| - 1} \cdot (L(g) - L(g'')) + \frac{L(g)}{|\mathrm{var}(g)| - 1} + \frac{1}{4}
\end{aligned}$$

$$\geq \frac{|\mathrm{var}(g)|-2}{|\mathrm{var}(g)|-1}\cdot\left(\frac{L(g)}{|\mathrm{var}(g)|}+\frac{1}{4}\right)+\frac{L(g)}{|\mathrm{var}(g)|-1}+\frac{1}{4}$$
$$=\frac{2L(g)}{|\mathrm{var}(g)|}+\frac{1}{2}-\frac{1}{|\mathrm{var}(g)|-1}\geq\frac{2L(g)}{|\mathrm{var}(g)|}+\frac{2}{5},$$

where the last inequality is obtained by $|\mathrm{var}(g)|\geq 11$.

Therefore, we complete the proof of Lemma 4, and then the second inequality holds with probability at least 1/2. □

Since $|\mathrm{var}(g')|\leq|\mathrm{var}(g)|-2$, the following corollary is obtained from Lemma 3.

Corollary 1.

$$\frac{L(g')}{|\mathrm{var}(g')|}\leq\frac{L(g)}{|\mathrm{var}(g)|}.$$

Next, we prove Lemma 6 by using Lemma 3 and the following lemma.

Lemma 5 (Lemma 4.2 [6]). *Let $\{X_i\}_{i=0}^n$ and $\{R_i\}_{i=1}^{n-1}$ be sequences of random variables and $Y_i=X_i-X_{i-1}$. If $\mathbf{E}[X_i\mid R_{i-1},\ldots,R_1]\leq X_{i-1}$ for $1\leq i\leq n$, and for every $1\leq i\leq n$, the random variables Y_i (conditioned on $R_{i-1},\ldots,R_1$) assumes two values with equal probability, and there exists a constant $c_i\geq 0$ such that $Y_i\leq c_i$, then, for any λ, we have*

$$\Pr[X_n-X_0\geq\lambda]\leq\exp\left(-\frac{\lambda^2}{2\sum_{i=1}^n c_i^2}\right)$$

Lemma 6. *Let f_{n-k} be the formula after restricting $n-k$ variables.*

$$\Pr\left[L(f_{n-k})\geq\sqrt{2}\cdot L(f)\cdot\left(\frac{k}{n}\right)^{1+\frac{|\mathrm{var}(f)|}{10L(f)}}\right]\leq 2\cdot 2^{-k/2}.$$

Proof. Consider the node in the computation tree at depth $2j$ for $j=1,\ldots,(n-k)/2$. Let $R_1,R_2,\ldots,R_{2(j-1)}$ be the random value that takes the assignment to the restricted variable at each step in EvalFormula. We define a sequence of random variables $Z_1,Z_2,\ldots,Z_j$ as follows:

$$Z_j=\log L(f_{2j})-\log L(f_{2(j-1)})-\left(1+\frac{|\mathrm{var}(f)|}{10L(f)}\right)\log\left(\frac{n-2j}{n-2j+2}\right).$$

First, we show that

$$Z_j\leq-\frac{|\mathrm{var}(f)|}{10L(f)}\log\left(\frac{n-2j}{n-2j+2}\right).$$

Since $L(f_{2j})\leq L(f_{2(j-1)})\left(1-\frac{2}{n-2j+2}\right)$ by Lemma 3, we have

$$\log L(f_{2j})-\log L(f_{2(j-1)})\leq\log\left(1-\frac{2}{n-2j+2}\right).$$

Therefore,

$$Z_j\leq\log\left(1-\frac{2}{n-2j+2}\right)-\left(1+\frac{|\mathrm{var}(f)|}{10L(f)}\right)\log\left(\frac{n-2j}{n-2j+2}\right)$$
$$=-\frac{|\mathrm{var}(f)|}{10L(f)}\log\left(1-\frac{2}{n-2j+2}\right)$$

holds. Let c_j be as follows:

$$c_j=-\frac{|\mathrm{var}(f)|}{10L(f)}\log\left(1-\frac{2}{n-2j+2}\right)\geq 0.$$

Thus, $Z_j\leq c_j$ holds. By Lemma 3, Corollary 1 and $|\mathrm{var}(f_{2(j-1)})|\leq n-2j+2$, conditioned on $R_1,R_2,\ldots,R_{2(j-1)}$, with a probability of at least 1/2, we have

$$\log L(f_{2j})-\log L(f_{2(j-1)})$$
$$\leq\left(1+\frac{|\mathrm{var}(f_{2(j-1)})|}{5L(f_{2(j-1)})}\right)\log\left(1-\frac{2}{|\mathrm{var}(f_{2(j-1)})|}\right)$$
$$\leq\left(1+\frac{|\mathrm{var}(f)|}{5L(f)}\right)\log\left(1-\frac{2}{n-2j+2}\right).$$

Then,

$$Z_j\leq\left(1+\frac{|\mathrm{var}(f)|}{5L(f)}\right)\log\left(1-\frac{2}{n-2j+2}\right)-\left(1+\frac{|\mathrm{var}(f)|}{10L(f)}\right)\log\left(1-\frac{2}{n-2j+2}\right)$$
$$=\frac{|\mathrm{var}(f)|}{10L(f)}\log\left(1-\frac{2}{n-2j+2}\right)=-c_j.$$

Thus, $Z_j\leq -c_j$ holds with a probability of at least 1/2.

Let $Y_1,Y_2,\ldots,Y_j$ be a sequence of random variables such that each Y_j takes $-c_j$ and c_j with equal probability. Since $Z_j\leq c_j$ always holds, and $Z_j\leq -c_j$ holds with a probability of at least 1/2, $\Pr[Z_j\geq\lambda]\leq\Pr[Y_j\geq\lambda]$ holds for any λ. Moreover, letting $X_0,X_1,\ldots,X_j$ be a sequence of random variables $X_0=0$ and $X_j=\sum_{\ell=1}^j Y_\ell$, we have $\mathbf{E}[X_j\mid R_{2(j-1)},\ldots,R_1]=X_{j-1}$. Thus, random variables X_i satisfy the conditions of Lemma 5. Applying Lemma 5, we have for any $\lambda>0$ and positive integer i,

$$\Pr[X_i-X_0\geq\lambda]=\Pr\left[\sum_{j=1}^i Y_j\geq\lambda\right]\leq\exp\left(-\frac{\lambda^2}{2\sum_{j=1}^i c_j}\right).$$

Let $i=(n-k)/2$. For simplicity, we assume $n-k$ can be divided by 2.

First, we estimate the sum of c_j^2. Since $|\mathrm{var}(f)|\leq L(f)$ holds and $\log(1+x)\leq x$ holds for any $x>-1$, we have

$$c_j=\frac{|\mathrm{var}(f)|}{10L(f)}\log\left(1+\frac{2}{n-2j}\right)\leq\frac{1}{2(n-2j)}.$$

Then, by elementary calculation, we have

$$\sum_{j=1}^{(n-k)/2}c_j^2\leq\frac{1}{10}\sum_{j=1}^{(n-k)/2}\left(\frac{1}{n-2j}\right)^2$$

$$\leq \frac{1}{10}\sum_{j=1}^{(n-k)/2}\frac{1}{(n-2j-2)(n-2j)}$$
$$\leq \frac{1}{10}\sum_{j=1}^{(n-k)/2}\frac{1}{2}\left(\frac{1}{n-2j-2}-\frac{1}{n-2j}\right)$$
$$= \frac{1}{20}\left(\frac{1}{k-2}-\frac{1}{n-2}\right) \leq \frac{1}{20(k-2)}.$$

Therefore, we have

$$\exp\left(-\frac{\lambda^2}{2\sum_{j=1}^{(n-k)/2}c_j^2}\right) \leq \exp(-10\lambda^2(k-2)).$$

By setting $\lambda = \frac{\log 2}{2} > 0.346\cdots > \frac{1}{10}$, we have $\lambda^2 > \frac{\log 2}{20}$ and then

$$\exp\left(-10\lambda^2(k-2)\right) < \exp\left(-\frac{\log 2}{2}(k-2)\right) = 2\cdot 2^{-k/2}.$$

Then, we have

$$\Pr\left[\sum_{j=1}^{i} Z_j \geq \frac{\log 2}{2}\right] \leq \Pr\left[\sum_{j=1}^{i} Y_j \geq \frac{\log 2}{2}\right] \leq 2\cdot 2^{-k/2}.$$

The rest of the proof shows that

$$\exp\left(\sum_{i=1}^{(n-k)/2} Z_j\right) = \frac{L(f_{n-k})}{L(f)\left(\frac{k}{n}\right)^{1+\frac{|\mathrm{var}(f)|}{10L(f)}}} \tag{4}$$

holds. Eq. (4) implies that

$$\Pr\left[\sum_{j=1}^{(n-k)/2} Z_j \geq \frac{\log 2}{2}\right] = \Pr\left[\exp\left(\sum_{j=1}^{(n-k)/2} Z_j\right) \geq \sqrt{2}\right]$$
$$= \Pr\left[L(f_{n-k}) \geq \sqrt{2}\cdot L(f)\left(\frac{k}{n}\right)^{1+\frac{|\mathrm{var}(f)|}{10L(f)}}\right] \leq 2\cdot 2^{-k/2}.$$

Recalling that

$$Z_j = \log L(f_{2j}) - \log L(f_{2(j-1)}) - \left(1+\frac{|\mathrm{var}(f)|}{10L(f)}\right)\log\left(\frac{n-2j}{n-2j+2}\right),$$

we have

$$\sum_{j=1}^{(n-k)/2} Z_j$$
$$= \sum_{j=1}^{(n-k)/2}\left[\log L(f_{2j}) - \log L(f_{2(j-1)})\right] - \left(1+\frac{|\mathrm{var}(f)|}{10L(f)}\right)\sum_{j=1}^{(n-k)/2}\log\left(\frac{n-2j}{n-2j+2}\right)$$
$$= \log L(f_{n-k}) - \log(L(f)) - \left(1+\frac{|\mathrm{var}(f)|}{10L(f)}\right)\log\left(\frac{n-2}{n}\cdot\frac{n-4}{n-2}\cdot\cdots\cdot\frac{k}{k+2}\right)$$
$$= \log L(f_{n-k}) - \log L(f) - \left(1+\frac{|\mathrm{var}(f)|}{10L(f)}\right)\log\left(\frac{k}{n}\right)$$
$$= \log L(f_{n-k}) - \log L(f) - \log\left(\frac{k}{n}\right)^{1+\frac{|\mathrm{var}(f)|}{10L(f)}}.$$

Hence, Eq. (4) holds and it completes the proof. □

Now, we estimate the running time of EvalFormula.

Lemma 7. *Let f be a formula with n variables and cn size satisfying Lemma 2. EvalFormula determines the satisfiability of f in time* $\mathrm{poly}(n)\cdot 2^{(1-\mu(c))n}$ *for* $\mu(c) = \frac{1}{2\cdot(2\sqrt{2}c)^{10c}}$.

Proof. Let $p = (2\sqrt{2}c)^{-10c}$ and $k = pn$. We build a computation tree based on adaptive restriction variables according to the cases in Lemma 2, and continue the process until there are at most k variables left.

We assume that neither constants nor formulas that satisfy cases 4 and 5 in Lemma 2 appear in this process. That is, let us consider the situation in which only cases 1–3 happen. (We will deal with the situation where case 4 or 5 happens later.) Since $c = L(f)/|\mathrm{var}(f)|$, we have

$$\left(\frac{k}{n}\right)^{1+\frac{|\mathrm{var}(f)|}{10L(f)}} = p^{1+\frac{1}{10c}} = p\cdot(2\sqrt{2}c)^{-10c\cdot\frac{1}{10c}} = \frac{p}{2\sqrt{2}c}.$$

This implies that

$$\sqrt{2}\cdot L(f)\cdot\left(\frac{k}{n}\right)^{1+\frac{|\mathrm{var}(f)|}{10L(f)}} = \sqrt{2}\cdot cn\cdot\frac{p}{2\sqrt{2}c} = \frac{pn}{2} = \frac{k}{2}$$

holds since $L(f) = cn$. Therefore, by Lemma 6, we have

$$\Pr\left[L(f_{n-k}) \geq \frac{k}{2}\right] \leq 2\cdot 2^{-k/2}.$$

This means that at most $2\cdot 2^{-k/2}$ fraction of the branches end with a formula size at least $k/2$ after assigning $n-k$ variables. We check the satisfiability of such formulas by the brute-force search for the remaining k variables. The running time for these branches is bounded by $O(2^{n-k}\cdot 2\cdot 2^{-k/2}\cdot 2^k) = O(2^{n-k/2})$.

For the other branches that are at least $1 - 2\cdot 2^{-k/2}$ fraction of the branches, the size of formulas of their end is less than $k/2$. This means that the number of the remaining variables is less than $k/2$, thus we can check the satisfiability of such formulas by the brute-force search for the remaining $k/2$ variables. For these branches, the running time is bounded by $O(2^{n-k}\cdot 2^{k/2}) = O(2^{n-k/2})$.

For the rest of analysis of the running time, let us consider all leaves at higher than the depth of $n-k$ and denote by S the set of these leaves. We show that the total of the running time over S is at most $\mathrm{poly}(n)\cdot 2^{n-k/2}$. By the definition of the computation tree, any formula f in the label of leaves

satisfies cases 4 or 5, or $|\mathrm{var}(f)| < k/2$. If case 5 happens, we determine the satisfiability of the formula in polynomial time by the Gaussian Elimination. If $|\mathrm{var}(f)| < k/2$, we check the satisfiability of f in time $O(2^{k/2})$ by the brute-force search for the remaining at most $k/2$ variables. Assume that case 4 happens at depth d, where $d < n-k$ holds. We determine the satisfiability of f in time $\mathrm{poly}(n) \cdot 2^{(n-d)/2}$ by a brute-force search and the Gaussian Elimination as described in Sect. 3.2 since $|\mathrm{var}(f)| \le n-d$ holds. Therefore, for any d with $d < n-k$, the running time for any leaf of depth d is at most $\mathrm{poly}(n) \cdot 2^{(n-d)/2}$. Moreover, we have $(n-d)/2 < n-d-k/2$ since $d < n-k$ holds. Thus, the total of the running time over S is at most

$$\sum_{s\in S} \mathrm{poly}(n) \cdot 2^{n-\mathrm{depth}(s)-\frac{k}{2}}$$
$$< \mathrm{poly}(n) \cdot 2^{n-\frac{k}{2}} \sum_{s\in S} 2^{-\mathrm{depth}(s)} \le \mathrm{poly}(n) \cdot 2^{n-\frac{k}{2}}.$$

The last inequality is due to the binary tree as shown by the Kraft–McMillan inequality[†].

Therefore, the overall running time is bounded by $\mathrm{poly}(n) \cdot 2^{n-k/2}$, and then its exponent is

$$n - \frac{k}{2} = \left(1 - \frac{p}{2}\right) n = \left(1 - \frac{1}{2 \cdot (2\sqrt{2}c)^{10c}}\right) n.$$

This completes the proof. □

Combining Lemmas 1 and 7, we give the following theorem that leads immediately to Theorem 1.

Theorem 2. *W2BP-Sat determines the satisfiability of a deterministic width-2 branching program with n variables and cn nodes, and it runs in time* $\mathrm{poly}(n) \cdot 2^{(1-\mu(c))n}$ *for* $\mu(c) = 1/2^{O(c \log c)}$.

Proof. Let B be a width-2 branching program with n variables and cn nodes. By Lemma 1, Transformation transforms B to formula f with n variables and at most $1.5cn$ leaves in time $O(n)$ and $|F|$ is bounded by $O(n)$. Then, Lemma 7 implies that EvalFormula determines the satisfiability of each $f_i \in F$ in time

$$\mathrm{poly}(n) \cdot 2^{\left(1 - \frac{1}{2\cdot(2\sqrt{2}\cdot(1.5c))^{10\cdot(1.5c)}}\right)n}$$
$$= \mathrm{poly}(n) \cdot 2^{\left(1 - \frac{1}{2\cdot(3\sqrt{2}c)^{15c}}\right)n}.$$

Thus, the time complexity of W2BP-Sat is

$$O(n) + O(n) \cdot \mathrm{poly}(n) \cdot 2^{\left(1 - \frac{1}{2\cdot(3\sqrt{2}c)^{15c}}\right)n}$$
$$= \mathrm{poly}(n) \cdot 2^{\left(1 - \frac{1}{2^{15c\log(3\sqrt{2}c)+1}}\right)n}.$$

□

[†]For any binary tree and any set $\mathcal{L}$ of leaves in the tree, $\sum_{s\in\mathcal{L}} 2^{-\mathrm{depth(s)}} \le 1$ holds.

Acknowledgments

This research was partially supported by JSPS KAKENHI Grant Numbers JP18K11170, JP18K18003, JP20K19741, JP20H05794, and JP21K11743.

References

[1] A. Abboud, T.D. Hansen, V.V. Williams, and R. Williams, "Simulating branching programs with edit distance and friends: or: A polylog shaved is a lower bound made," Proc. 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC), pp.375–388, 2016.
[2] D.A. Barrington, "Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1," J. Comput. Syst. Sci., vol.38, no.1, pp.150–164, 1989.
[3] A. Bogdanov, Z. Dvir, E. Verbin, and A. Yehudayoff," "Pseudorandomness for width-2 branching programs," Theory of Comput., vol.9, pp.283–293, 2013.
[4] B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener, "Hierarchy theorems for kOBDDs and kIBDDs," Theoretical Computer Science, vol.205, no.1-2, pp.45–60, 1998.
[5] A. Borodin, D. Dolev, F.E. Fich, and W.J. Paul, "Bounds for width two branching programs," SIAM J. Comput., vol.15, no.2, pp.549–560, 1986.
[6] R. Chen, V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman, "Mining circuit lower bound proofs for meta-algorithms," Proc. 21st Annual IEEE Conference on Computational Complexity (CCC), pp.262–273, 2014.
[7] R. Meka, O. Reingold, and A. Tal, "Pseudorandom generators for width-3 branching programs," Proc. 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC), pp.626–637, 2019.
[8] A. Nagao, K. Seto, and J. Teruyama, "A moderately exponential time algorithm for k-IBDD satisfiability," Algorithmica, vol.80, no.10, pp.2725–2741, 2018.
[9] A. Nagao, K. Seto, and J. Teruyama, "Satisfiability algorithm for syntactic read-k-times branching programs," Theory of Computing Systems, vol.64, no.8, pp.1392–1407, 2020.
[10] R. Santhanam, "Fighting perebor: New and improved algorithms for formula and QBF satisfiability," 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp.183–192, 2010.
[11] K. Seto and S. Tamaki, "A satisfiability algorithm and average-case hardness for formulas over the full binary basis," Comput. Complex., vol.22, no.2, pp.245–274, 2013.
[12] R. Williams, "Improving exhaustive search implies superpolynomial lower bounds," SIAM J. Comput., vol.42, no.3, pp.1218–1244, 2013.
[13] A.C.-C. Yao, "Lower bounds by probabilistic arguments (extended abstract)," Proc. 24th Annual Symposium on Foundations of Computer Science (STOC), pp.420–428, 1983.

Tomu Makita received B.Sci. degree from Seikei University in 2020.

Atsuki Nagao received B.Eng., M.Info., and Ph.D. degrees in Informatics in 2010, 2012, and 2015, respectively. He is now an assistant professor in Faculty of Core Research Natural Science Division Ochanomizu University. He has majored in computational complexity, log-spaced algorithms, combinatorial games, and puzzles.

Tatsuki Okada received B.Sci. and M.Sci. degrees from Seikei University in 2019 and 2021, respectively.

Kazuhisa Seto received B.Eng., M.Info., and Ph.D. degrees in Informatics from Kyoto University in 2008, 2010, and 2013, respectively. He is now an associate professor at Hokkaido University. His research interests include satisfiability problems and circuit complexity.

Junichi Teruyama received B.Eng., M.Info., and Ph.D. degrees in Informatics in 2008, 2010, and 2013, respectively. He is now an assistant professor at University of Hyogo. His research interests include satisfiability problems, sorting algorithms, and query complexity.