Two-Dimensional Indirect Binary Search
for the Positive One-In-Three Satisfiability Problem

Shunichi Matsubara*
Aoyama Gakuin University,
5-10-1, Fuchinobe, Chuo-ku, Sagamihara, Kanagawa,
252-5258, Japan

Abstract
In this paper, we propose an algorithm for the positive-one-in-three satisfiability problem (Pos1In3Sat). The proposed algorithm decides the existence of a satisfying assignment in all assignments for a given formula by using a 2-dimensional binary search method without constructing an exponential number of assignments.

1 Introduction
In this paper, we propose an algorithm for the positive-one-in-three satisfiability problem (Pos1In3Sat). Pos1In3Sat is known to be NP-complete [3]. We prove that the proposed algorithm can run efficiently.

The proposed algorithm decides whether there is a satisfying assignment for a given positive 3CNF formula by using a 2-dimensional version of the binary search method. First, it constructs an equivalent positive 3CNF formula for the given formula as the preprocess for the binary search. Then, it encodes all partial assignments to single variables in the constructed positive 3CNF formula. As a result, we obtain a matrix whose components means a truth assignment in the constructed formula. The algorithm does the binary search for that matrix. Every row and column in that matrix are sorted in ascending order. Thus, the algorithm can expectedly do the binary search. Representing all components of the matrix requires an exponential space for the size of the input formula. However, we can use the matrix without constructing all the components.

In Section 2 we define some basic concepts and notation. In Section 3 we propose an algorithm for Pos1In3Sat. Then, we prove its validity and analyze its running time.

* matusbara@it.aoyama.ac.jp
2 Basic concepts and notation

In this section, we define basic concepts and notation that are used throughout the paper. We follow convention of literature in theoretical computer science or combinatorics.

We denote the empty string by $\varepsilon$. We denote by $\lceil p \rceil$ a characteristic function on a predicate $p$; i.e., $\lceil p \rceil$ is 1 if $p = 1$, and 0 otherwise.

We denote the sets of all nonnegative and positive integers by $\mathbb{N}$ and $\mathbb{N}_+$, respectively. Given $l, u \in \mathbb{N}$, we denote the interval $\{i \in \mathbb{N} : l \leq i \leq u\}$ by $[l, u]$. Given $l, u \in \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$, if $l > u$, then we consider $\sum_{x=l}^{u} f(x)$ to be 0.

Let $\Phi$ be a finite set. Let $n \in \mathbb{N}_+$. We denote $\bigcup_{i \in \mathbb{N}_+} \Phi^i$ by $\Phi^*$. Let $b \in \Phi^*$. Given $b$, for every $i \in [1, n]$, we represent the $i$th component of $b$ by the corresponding normal weight symbol $b_i$; i.e., $b = (b_1, \ldots, b_n)$. Conversely, given $n$ elements $b_1, \ldots, b_n$ of $\Phi$, $b$ denotes the vector $(b_1, \ldots, b_n)$. Given $l, u \in \mathbb{N}$, we denote the vector $(b_l, \ldots, b_u)$ by $b_{lu}$. Given $\delta \in \Phi$ and $k \in \mathbb{N}_+$, we denote the vector $(\delta, \ldots, \delta)$ by $\delta^k$. We omit the subscript “$(k)$” if no confusion arises. Let $a \in \Phi^n$. We denote $a \leq b$ if and only if $a_i \leq b_i$ for every $i \in [1, n]$. Given $a, b$, we denote the inner product $\sum_{i=1}^{n} a_i b_i$ by $a \cdot b$. For convenience, we identify a vector in $\Phi^n$ with a sequence of length $n$ over $\Phi$, a string of length $n$ over $\Phi$, or a mapping from $[1, n]$ to $\Phi$ if no confusion arises. For example, we identify the vector $(b_1, \ldots, b_n)$ with the sequence $b_1, \ldots, b_n$, the string $b_1 \cdots b_n$, or a mapping that maps $i$ to $b_i$ for every $i \in [1, n]$. We denote $b \in \Phi$ if $b$ contains $a$ as a component. We define a binary relation $\subseteq$ over $\Phi^*$ as follows. Given $a$ and $b$ in $\Phi^*$, $a \subseteq b$ if and only if $a_i = b_i$ for every $i \in [1, |a|]$. Given $b$, we denote its reverse $(b_n, \ldots, b_1)$ by $b^R$. Given $l, u \in [1, n]$, $b_{lu}^R$ denotes $(b_{lu})^R$.

We say that a finite set $A$ of integers can be 2-dimensionally-sorted in ascending (descending) order if there is a matrix $M$ such that every row and column are sorted in ascending (descending) order; and there is some one to one correspondence from $A$ to the set of all components of $M$.

2.1 Concepts and notation on integers

Let $b \in \mathbb{N}_+$. Let $n, k \in \mathbb{N}$ such that $\lceil \log_b n \rceil + 1 \leq k$. If $d_i$ is $\lceil n/b^{i-1} \rceil \mod b$ for every $i \in [1, k]$, then we call the string $d_k \cdots d_1$ over $[0, b-1]$ the base-$b$ representation of $n$ of length $k$. We omit the phrase “of length $k$” if no confusion arises. Given a base-$b$ representation $\alpha$ of some $m \in \mathbb{N}$, $(\alpha)_b$ denotes the integer $m$. We consider $\varepsilon$ to be the base-$b$ representation of 0 of length 0; i.e., we consider $(\varepsilon)_b$ to be 0. Given $l, u \in [1, k]$ with $l < u$, if $d_k \cdots d_1$ is the base-$b$ representation of $n$, then we call the substring $d_l \cdots d_u$ the base-$b$ $(l, u)$-zone of $n$. We omit the phrase “base-$b$” if no confusion arises.
2.2 Boolean formulae

In this subsection, we define notation and assumptions and review some concepts on Boolean formulae. We assume the reader to be familiar to basic concepts in Boolean satisfiability. The reader is referred to some books by some chapters in Arora and Barak [1], Creignou, Khanna, and Sudan [2], or Wegener [4] if necessary.

2.2.1 Assumptions

In this subsubsection, we define assumptions on Boolean formulae, which we use throughout the paper. These assumptions are for technical reasons. By those assumption, we do not lose the generality of discussion on polynomial-time computability.

We fix $Z$ to be a countable set of Boolean variables. For every $i \in \mathbb{N}^+$, $z_i$ denotes a Boolean variable in $Z$. We assume that every Boolean formula in this paper is defined over $Z$. We fix $\phi$ and $\psi$ to be positive 3CNF formulae over $Z$.

We assume a clause in a Boolean formula to be a sequence of literals although a clause is often assumed to be a set of literals in other literature. For example, we distinguish $z_1 \vee z_2 \vee z_3$ from $z_3 \vee z_2 \vee z_2$. Similarly, we assume a CNF formula to be a sequence of clauses although a CNF formula is often assumed to be a set of clauses in other literature. For example, given clauses $C_1$ and $C_2$, we distinguish a conjunction $C_1 \land C_2$ from $C_2 \land C_1$. Needless to say, the satisfiability of a given formula do not depend on whether clauses or formulae are regarded as sets or sequences.

Let $\phi$ be a given. We assume that $\phi$ consists of 2 or more clauses. We assume that every clause in $\phi$ contains distinct variables. We assume that no two clauses in $\phi$ consist of the identical combination of variables. We assume that the indices of variables occurring in $\phi$ are successive integers from 1; i.e., the set of all variables occurring in $\phi$ can be represented as $\{z_i : i \in [1,k]\}$ for some $k \in \mathbb{N}$. We consider the size of a Boolean formula to be the number of variables in the formula.

2.2.2 Concepts and notation

Let $z \in Z$. Then, we call $z$ or $\neg z$ literals. In particular, we call $z$ a positive literal; and $\neg z$ a negative literal. We say that $\phi$ is positive if $\phi$ consists of only positive literals. We denote the set of all variables in $\phi$ by $V(\phi)$. We denote the set of all clauses in $\phi$ by $C_\phi$.

Suppose that $\phi$ is represented as $C_m \land \cdots \land C_1$; i.e., $m = |C_\phi|$. Let $k$ be the number of variables in $\phi$. We define a partial assignment $\sigma$ for $\phi$ as a mapping from $\{z_i : i \in [1,j]\}$ to $\{0,1\}$, where $j \in [1,|V_\phi|]$. We call $\sigma$ a truth assignment for $\phi$ if $|\sigma| = |V(\phi)|$. We often call a truth assignment for $\phi$ simply an assignment for $\phi$. Let $\sigma \in \{0,1\}^\nu$. We say that a partial
assignment \( \sigma \) is a 1-in-3-satisfying for \( \varphi \) if \( |\{z: \sigma(z) = 1, z \in C_j\}| \leq 1 \) for every \( j \in [1, m] \). We say that an assignment \( \sigma \) is 1-in-3-satisfying for \( \varphi \) if \( |\{z_i: \sigma(z_i) = 1\}| = 1 \) for every \( j \in [1, m] \). Given \( \varphi \), if there is a 1-in-3-satisfying assignment \( \sigma \), then we say that \( \varphi \) is 1-in-3-satisfiable. We consider \( \varepsilon \) to be a partial assignment that does not assign anything. Given a partial assignment \( \sigma \) for \( \varphi \) and a literal \( z \) in \( \varphi \), we call \( z \) a true literal in \( \sigma \) if \( \sigma(z) = 1 \). Let \( b \in \mathbb{N}_+ \). Given \( i \in [1, k] \), we denote the integer \( \sum_{j=1}^{m} b^{i-1} |\{z_i \in C_j\}| \) by \( \langle \varphi, z_i \rangle_b \). We denote the vector \( ((\varphi, z_1)_4, \ldots, (\varphi, z_k)_4) \) by \( \bar{\varphi} \).

**Problem 1 (Pos1In3Sat).**

**Instance.** A positive 3CNF formula \( \varphi \).

**Question.** Is \( \varphi \) 1-in-3-satisfiable?

### 2.3 Computational complexity

We assume the reader to be familiar to basic concepts and results in computational complexity theory. The reader is referred to Arora and Barak [1] if necessary. Basically, we estimate the running time of an algorithm by using a function in the bit length of a given input. On the other hand, this paper focuses on the polynomial-time computability of Pos1In3Sat. Thus, we analyze the running time of an algorithm roughly to some extent that we do not lose the correctness in favor of clarity of discussion. For example, as we described in Subsubsection 2.2.1, we adopt the number of variables as the size of a given 3CNF formula.

### 3 Algorithm

In this section, we propose a new algorithm for Pos1In3Sat. Subsection 3.1 describes the outline and key ideas of this algorithm informally. Subsection 3.2 describes the details of the algorithm formally. In Subsection 3.3, we prove the validity of the algorithm. Finally, in Subsection 3.4, we analyze the running time of the algorithm.

For preparation, we fix some symbols as follows. \( C_{m_1} \wedge \cdots \wedge C_1 \) denotes a positive 3CNF formula \( \psi \). Moreover, \( k_1 \) denotes the number of variables in \( \psi \). For every \( i \in [1, m_1] \) and \( j \in [1, 3] \), \( \langle i, j; \psi \rangle \) denotes an integer in \([1, k_1]\) such that \( C_i = z_{(i,3; \psi)} \lor z_{(i,2; \psi)} \lor z_{(i,1; \psi)} \). We denote \( \langle i, j; \psi \rangle \) by simply \( \langle i, j \rangle \) if no confusion arises.

#### 3.1 Ideas

In this subsection, we first outline the algorithm that we propose in this paper. After that, we describe some intuitive ideas of the algorithm by executing the algorithm for a 3CNF formula.
3.1.1 Outline

In the proposed algorithm, every Boolean variable $z_i$ in a given $\psi$ is encoded to $(\psi, z_i)_4$. Given $(\psi, z_i)_4$, we can observe the following meaning for its base-4 representation of length $m_1$. In that base-4 representation, the $j$th digit from the right end means whether the clause $C_j$ contains $z_i$, where $j \in [1, m_1]$. That is, $(\psi, z_i)_4$ simulates the assignment of 1 to $z_i$ in $\psi$. Thus, we can represent a total truth assignment for $\psi$ as the integer $\sum_{i \in I}(\psi, z_i)_4$ for some $I \subseteq [1, k_1]$. If $i \in I$, then we consider $z_i$ to be assigned 1 in $\psi$. Then, a satisfying assignment for $\psi$ corresponds to $\left(\underbrace{1\ldots1}_{m_1}\right)_4$; i.e., $\sum_{i=1}^{m_1}4^{i-1}$.

A basic strategy in the algorithm is a 2-dimensional version of binary search. First, the algorithm does a preprocess for the given $\psi$. By that procedure, we construct a positive 3CNF formula $\varphi$ of $m$ clauses and $k$ variables. Then, the algorithm searches the integer $\sum_{i=1}^{m}4^{i-1}$ in the set of integers $\sum_{i \in I}(\psi, z_i)_4, \ldots, \sum_{i \in I_a}(\psi, z_i)_4$, where $\alpha = 2^k - 1$ and $I_0, \ldots, I_a$ are distinct subsets of $[1, k]$. Needless to say, an exponential space is necessary to explicitly construct all the integers $\sum_{i \in I}(\psi, z_i)_4, \ldots, \sum_{i \in I_a}(\psi, z_i)_4$. Thus, we can do this search without explicitly constructing the overall sequence. Moreover, the sequence $\sum_{i \in I}(\psi, z_i)_4, \ldots, \sum_{i \in I_a}(\psi, z_i)_4$ is required to be sorted in an order. Sorting these integers in 1-dimension appears to be difficult. However, if we arrange those integers in 2-dimension, then we can sort them, as we will describe below.

We fix $\psi_1$ to be a positive 3CNF $\psi_1 = \left(\frac{z_1 \lor z_2 \lor z_3}{C_2} \land \frac{z_1 \lor z_2 \lor z_4}{C_1}\right)$. In the remaining part of this subsection, we will describe the details of the proposed algorithm for $\psi_1$. In the algorithm, for convenience, we replace $\psi_1$ by new 3CNF formulae some times. Thus, for every symbol, we often use a parenthesized superscript for distinguishing the phase when the symbol is used.

3.1.2 Preprocess

As a preparation for the main search, the proposed algorithm constructs a new 3CNF formula from $\psi_1$, and then encodes it to a set of integers. Let us describe it in more detail below. Let us represent $\psi_1$ in the earliest phase of the algorithm by $\left(\frac{z_1^{(0)} \lor z_2^{(0)} \lor z_3^{(0)}}{C_2^{(0)}} \land \frac{z_1^{(0)} \lor z_2^{(0)} \lor z_4^{(0)}}{C_1^{(0)}}\right)$. Then, for every $i \in [1, 4]$, the base-4 representation for $(\psi_1^{(0)}, z_i^{(0)})_4$ is illustrated in Table 1.

First, the algorithm replaces the indices of the variables so that $(\psi_1, z_1)_4 \leq (\psi_1, z_2)_4 \leq (\psi_1, z_3)_4 \leq (\psi_1, z_4)_4$. Let us represent $z_i$ as $z_i^{(1)}$ for every $i \in [1, 4]$ and $C_j$ as $C_j^{(1)}$ for every $j \in \{1, 2\}$ in the phase immediately after those re-
In $\psi$ variables following clauses. That is, in this phase, $\psi(1) = \left(\frac{z_4^{(1)} \lor z_3^{(1)} \lor z_2^{(1)}}{C_2^{(1)}}\right) \land \left(\frac{z_4^{(1)} \lor z_3^{(1)} \lor z_1^{(1)}}{C_1^{(1)}}\right)$.

In $\psi(1)$, every occurrence of every variable can be represented as Table 2.

In the next phase, the algorithm constructs three clauses for every clause $C_j^{(1)}$, where $j \in \{1, 2\}$, by using the variables $z_{(j,1)}$, $z_{(j,2)}$, and $z_{(j,3)}$ and new variables $z_{k_1+4(j-1)+1}$, $z_{k_1+4(j-1)+2}$, and $z_{k_1+4(j-1)+3}$. In this phase, let us use "(2)" as a superscript of every symbol. In more details, we construct the following clauses.

$C_{4j}^{(2)} = (z_{k_1+4(j-1)+3} \lor z_{k_1+4(j-1)+2} \lor z_{(j,3)})$

$C_{4j-1}^{(2)} = (z_{k_1+4(j-1)+3} \lor z_{k_1+4(j-1)+1} \lor z_{(j,2)})$

$C_{4j-2}^{(2)} = (z_{k_1+4(j-1)+3} \lor z_{(j,3)} \lor z_{(j,2)})$

Moreover, the algorithm renames the clause $C_j$ as $C_{4j-3}$; i.e., $C_{4j-3}^{(2)} = C_j^{(1)}$. That is, the algorithm constructs the following $\psi_1^{(2)}$.

\[
\begin{align*}
\left(\frac{z_{10} \lor z_9 \lor z_4}{C_8}\right) \land \left(\frac{z_{10} \lor z_8 \lor z_3}{C_7}\right) \land \left(\frac{z_{10} \lor z_4 \lor z_3}{C_6}\right) \land \left(\frac{z_4 \lor z_3 \lor z_2}{C_5}\right) \\
\land \left(\frac{z_7 \lor z_6 \lor z_4}{C_4}\right) \land \left(\frac{z_7 \lor z_5 \lor z_3}{C_3}\right) \land \left(\frac{z_7 \lor z_4 \lor z_3}{C_2}\right) \land \left(\frac{z_4 \lor z_3 \lor z_1}{C_1}\right)
\end{align*}
\]

### 3.1.3 Sorted matrix

After the preprocess in Subsection 3.1.2 we can find a $(2^4 \times 2^6)$-matrix $M_{\psi_1}$, each of whose rows and columns is sorted. In this subsection, we describe more details of $M_{\psi_1}$. In this subsection, we fix $\sigma$ to be an assignment for $\psi_1^{(2)}$. Every component in $M_{\psi_1}$ corresponds to an assignment.

| Base-4 representation | Table 1: Base-4 representation for $(\psi_1^{(0)}, \tau_i^{(0)})_4$, where $i \in [1, 4]$. |
|-----------------------|------------------------------------------------------------------|
| $(\psi_1, z_1)_4$     | $(\psi_1, z_2)_4$                                               |
| $(\psi_1, z_3)_4$     | $(\psi_1, z_4)_4$                                               |
| 11                    | 11                                                              |
| 10                    | 01                                                              |

Table 2: Correspondence between the indices of variables in two ways.

| $z_{(2,3)}$ | $z_{(2,2)}$ | $z_{(2,1)}$ | $z_{(1,3)}$ | $z_{(1,2)}$ | $z_{(1,1)}$ |
|------------|------------|------------|------------|------------|------------|
| $z_4$      | $z_3$      | $z_2$      | $z_4$      | $z_3$      | $z_1$      |
for $\psi_1$; and conversely, given an assignment $\sigma$, there is a component in $M_{\psi_1}$ corresponding to $\sigma$. Given $\sigma$, we denote $((\sigma R_{1,4})_2 + 1, (\sigma R_{5,10})_2 + 1)$ by $f_{\psi_1}(\sigma)$. The pair $f_{\psi_1}(\sigma)$ means the position of a component in $M_{\psi_1}$. Then, $M_{\psi_1}$ is a matrix such that every row and column are sorted in ascending order.

Table 3: $(\psi_1^{(2)}, z_i^{(2)})_4$ for every $i \in [1, 4]$.

| $i$ | $\sigma^R$ | $f_{\psi_1}(\sigma)$ | $(\psi_1, z_i)_4$ |
|-----|------------|----------------------|-------------------|
| 1   | 0000000001 | ((0001)_2, 0)        | (00000001)_4      |
| 2   | 0000000010 | ((0010)_2, 0)        | (00100010)_4      |
| 3   | 0000000100 | ((0100)_2, 0)        | (01120112)_4      |
| 4   | 0000001000 | ((1000)_2, 0)        | (11231123)_4      |

By Tables 3 and 5, we can observe that the larger $i$ is, the larger $(\psi, z_i)_4$ is. Table 3 and Table 5 show the constructed integers that affect the ordering of the magnitudes in the column and row directions in $M_{\psi_1}$, respectively.

Table 4: Integer $\sigma \cdot \tilde{\psi}_1$, which is the value of the $f_{\psi_1}(\sigma)$-component in $M_{\psi_1}$, for $\sigma \in \{1\}^i \{0\}^{10-i}$, where $i \in [1, 4]$.

| $i$ | $\sigma^R$ | $f_{\psi_1}(\sigma)$ | $\sigma \cdot \tilde{\psi}_1$ |
|-----|------------|----------------------|-------------------------------|
| 1   | 0000000001 | ((0001)_2, 0)        | (00000001)_4                  |
| 2   | 0000000011 | ((0011)_2, 0)        | (00010000)_4                  |
| 3   | 0000000111 | ((0111)_2, 0)        | (01120112)_4                  |
| 4   | 0000001111 | ((1111)_2, 0)        | (11231123)_4                  |

Table 5: $(\psi_1^{(2)}, z_i^{(2)})_4$ for every $i \in [5, 10]$.

| $i$ | $\sigma^R$ | $f_{\psi_1}(\sigma)$ | $(\psi_1, z_i)_4$ |
|-----|------------|----------------------|-------------------|
| 5   | 0000010000 | (0, (000001)_2)     | (00000100)_4      |
| 6   | 0000100000 | (0, (000010)_2)     | (00001100)_4      |
| 7   | 0001000000 | (0, (000100)_2)     | (00022100)_4      |
| 8   | 0010000000 | (0, (010000)_2)     | (01002210)_4      |
| 9   | 0100000000 | (0, (010000)_2)     | (11002210)_4      |
| 10  | 1000000000 | (0, (100000)_2)     | (22102210)_4      |

Table 4 shows that the first row in $M_{\psi_1}$ is sorted. By that table, we can find that every row in $M_{\psi_1}$ is sorted. Table 4 shows that the first column
Table 6: Integer $\sigma \cdot \hat{\psi}_1$, which is the value of the $f_{\psi_1}(\sigma)$-component in $M_{\psi_1}$ for $\sigma \in \{1\}^{\mathbb{I}\{0\}^{10-i}}$, where $i \in [5, 10]$.

| $i$ | $\sigma^R$ | $f_{\psi_1}(\sigma)$ | $\sigma \cdot \hat{\psi}_1$ |
|-----|------------|----------------------|-----------------------------|
| 5   | 0000010000 | (0, (0000011111))   | (0000010000)                |
| 6   | 0000110000 | (0, (0000101111))   | (0000110000)                |
| 7   | 0011110000 | (0, (0001111111))   | (0011110000)                |
| 8   | 0111110000 | (0, (0111111111))   | (0111110000)                |
| 9   | 1111110000 | (0, (1111111111))   | (1111110000)                |
| 10  | 1111111000 | (0, (1111111111))   | (22102221010)               |

By Observation 2, the following holds.

$$2^4 \left( \sum_{i=5}^{u_1} (\psi_1^{(2)}, z_i^{(2)})_4 \right) + \sum_{i=1}^{l_1} (\psi_1^{(2)}, z_i^{(2)})_4 \leq 2^4 (\psi_1^{(2)}, z_{u_2}^{(2)})_4 + (\psi_1^{(2)}, z_{l_2}^{(2)})_4.$$

Observation 3. Let $\sigma$ and $\mu$ be vectors such that $f_{\psi_1}(\sigma) \leq f_{\psi_1}(\mu)$. Then,

$$\sigma \cdot \hat{\psi}_1 \leq \mu \cdot \hat{\psi}_1.$$

Consequently, we find the following observation.

Observation 4. Let $M_{\psi_1}$ be the matrix whose $(i, j)$-element is $f^{-1}(i-1, j-1) \cdot \hat{\psi}_1$, where $i \in \{1, 2^4\}$ and $j \in \{1, 2^6\}$. Then, $M_{\psi_1}$ is sorted in ascending order.

3.1.4 Indirect search for an implicit matrix

In this subsection, we fix $\psi_1$ to be $\psi_1^{(2)}$, and for every $i \in [1, 10]$, fix $z_i$ to be $z_i^{(2)}$. After constructing $\psi_1$, the algorithm constructs the set $\{ (\psi_1, z_i) : i \in [1, 10] \}$. Then, it searches the integer $\sum_{i=1}^8 4^{i-1}$ among the matrix $M_{\psi_1}$. $\sum_{i=1}^8 4^{i-1}$ corresponds to a satisfying assignment for $\psi_1$. An assignment $\sigma$ satisfies $\psi_1$ if and only $\sigma \cdot \hat{\psi}_1$ is (1111111111)$_4$; i.e., $\sum_{i=1}^8 4^{i-1}$. Needless to say, representing all components in $M_{\psi_1}$ requires an exponential space for the input size. However, we can simultaneously do binary searches in column and row directions in $M_{\psi_1}$ without explicitly constructing all the integers.
Figure 1: Values of components of the matrix $M_{\psi_1}$, whose $(i,j)$-element is $f^{-1}(i-1, j-1) \cdot \psi_1$, where $i \in [1,2^4]$ and $j \in [1,2^6]$.

Let us describe more details of that search below. Figure 2 illustrates the matrix $M_{\psi_1}$ in the first phase of the search. We first set the assignment 011111111 to the first candidate. In Figure 2, the left and right squares represent $M_{\psi_1}$ in case when the value of the component corresponding to the assignment 11111111 are smaller and larger than the one corresponding to the first candidate 01111111, respectively.
3.2 Formal details

In this subsection, we describe the details of our algorithms for Pos1In3Sat.
Algorithm 1: IsSat

Input: A 3CNF formula $\psi$

Output: A Boolean value, which means whether $\psi$ is satisfiable.

1 IsSat($\psi$)
2 $k_1 \leftarrow$ (the number of variables in $\psi$)
3 $m_1 \leftarrow$ (the number of clauses in $\psi$)
4 Rename variables in $\psi$ such that $(\psi, z_1) \leq \cdots \leq (\psi, z_k)$
5 foreach $i \in [1, m_1]$ Sort literals in $C_i$ so that $C_i = z_{(i, 3)} \lor z_{(i, 2)} \lor z_{(i, 1)}$ and $(i, 3) > (i, 2) > (i, 1)$.
6 foreach $i \in [1, m_1]$
7 $C_{4i} \leftarrow (z_{k_1 + 3i} \lor z_{k_1 + 3i - 1} \lor z_{(i, 3)})$ // a new clause
8 $C_{4i - 1} \leftarrow (z_{k_1 + 3i} \lor z_{k_1 + 3i - 2} \lor z_{(i, 2)})$ // a new clause
9 $C_{4i - 2} \leftarrow (z_{k_1 + 3i} \lor z_{(i, 3)} \lor z_{(i, 2)})$ // a new clause
10 $C_{4i - 3} \leftarrow C_i$ // a clause in $\psi$
11 $m \leftarrow 4m_1$ // the number of clauses in $\varphi$
12 $\varphi \leftarrow C_m \land \cdots \land C_1$
13 $k_2 \leftarrow 3m_1$ // the number of new variables
14 Compute $\bar{\varphi}$.
15 $t \leftarrow \sum_{i=1}^{m} 4^{i - 1}$. // a target integer
16 $p \leftarrow (0, 0)$ // the smallest corner in the search area
17 $q \leftarrow (2^{k_1} - 1, 2^{k_2} - 1)$ // the largest corner in the search area
18 return 2DIBSearch($\bar{\varphi}, p, q, k_1, t$)
Algorithm 2: 2DIBSearch

Input: A 5-tuple \((\Phi, p, q, k_1, t)\), where \(\Phi\) is in \(\mathbb{N}^*\); \(p \in \mathbb{N}^*_k\) and \(q \in \mathbb{N}^*_2\) mean the smallest and largest points in the area to search; \(k_1\) means a bias for distinguishing coordinates; and \(t \in \mathbb{N}_+\) means a target value.

Output: A Boolean value, which means whether \(t\) can be represented as \(b \cdot \Phi\) for some \(b \in \{0, 1\}^k\).

1. if \(q_1 - p_1 \leq 1\) and \(q_2 - p_2 \leq 1\)
   return 0
2. \(k \leftarrow (\text{the number of components of } \Phi)\)
3. \(r \leftarrow 2k_1 \cdot \left[\frac{p_1 + q_1}{2}\right] + \left[\frac{p_2 + q_2}{2}\right]\) // \(r\) means an indicator for \(\Phi\).
4. \(s \leftarrow \sum_{j=1}^k (\varphi_j, z_j) \cdot (\lvert r/2^{j-1} \rvert \mod 2)\) // \(s\) means a candidate.
5. if \(s = t\) // (the target) = (the current candidate)
   return 1
6. else if \(t < s\) // (the target) < (the current candidate)
   if \(q_1 - p_1 \geq 2\) and \(q_2 - p_2 \geq 2\)
     if 2DIBSearch(\(\Phi, (p_1, \left[\frac{p_1 + q_1}{2}\right]), (p_2, \left[\frac{p_2 + q_2}{2}\right]), k_1, t\) = 1
     return 1
   else
     if 2DIBSearch(\(\Phi, (p_1, \left[\frac{p_1 + q_1}{2}\right]), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\) = 1
     return 1
     else
     return 2DIBSearch(\(\Phi, (p_1, q_1), (p_2, \left[\frac{p_2 + q_2}{2}\right]), k_1, t\)
9. else
10. else if \(q_1 - p_1 \leq 1\)
   return 2DIBSearch(\(\Phi, (p_1, q_1), (p_2, \left[\frac{p_2 + q_2}{2}\right]), k_1, t\)
11. else
12. return 2DIBSearch(\(\Phi, (p_1, \left[\frac{p_1 + q_1}{2}\right]), (p_2, \left[\frac{p_2 + q_2}{2}\right]), k_1, t\)
13. return 2DIBSearch(\(\Phi, (p_1, \left[\frac{p_1 + q_1}{2}\right]), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\)
14. return 2DIBSearch(\(\Phi, (p_1, q_1), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\)
15. else
16. if \(q_1 - p_1 \geq 2\) and \(q_2 - p_2 \geq 2\)
   if 2DIBSearch(\(\Phi, (p_1, \left[\frac{p_1 + q_1}{2}\right]), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\) = 1
   return 1
   else
   if 2DIBSearch(\(\Phi, (\left\lfloor\frac{p_1 + q_1}{2}\right\rfloor), q_1), (p_2, \left[\frac{p_2 + q_2}{2}\right]), k_1, t\) = 1
   return 1
   else
   return 2DIBSearch(\(\Phi, (p_1, q_1), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\)
19. else
20. if \(q_1 - p_1 \leq 1\)
   return 2DIBSearch(\(\Phi, (p_1, q_1), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\)
21. else
22. if 2DIBSearch(\(\Phi, (\left\lfloor\frac{p_1 + q_1}{2}\right\rfloor), q_1), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\) = 1
   return 1
   else
   return 2DIBSearch(\(\Phi, (p_1, q_1), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\)
25. return 2DIBSearch(\(\Phi, (\left\lfloor\frac{p_1 + q_1}{2}\right\rfloor), q_1), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\)
26. else
27. return 2DIBSearch(\(\Phi, (p_1, q_1), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\)
28. else
29. return 2DIBSearch(\(\Phi, (p_1, q_1), (\left\lfloor\frac{p_2 + q_2}{2}\right\rfloor), k_1, t\)
3.3 Validity

We fix $\varphi$ to be as in Algorithm 1. We fix $f$ to be a mapping such that $f(\sigma)$ is the pair $((\sigma_{1:k})_2^R + 1, (\sigma_{k+1:k})_2^R + 1)$ for a given assignment $\sigma$. We fix $M$ to be the $(2^k, 2^k)$-matrix whose $(i, j)$-element is $f^{-1}(i, j) \cdot \varphi$, where $i \in [1, 2^k]$ and $j \in [1, 2^k]$. Lemma 5 is necessary for 2DIBSEARCH to execute its procedure expectedly. Note that we do not compute all parts of the matrix $M$ in Algorithms 1.

**Lemma 5** (2-dimensional sortability). Let $x$ and $y$ be integers in $[2, 2^k]$ and $[2, 2^k]$, respectively. Then, the following holds.

1. $m_{x-1,y} < m_{x,y}$.
2. $m_{x,y-1} < m_{x,y}$.

**Proof of Lemma 5** We fix $\xi$ to be the vector $f^{-1}(x, y)$.

Let us first prove the inequality (5.1). Let $\theta$ be the vector $f^{-1}(x - 1, y)$. By definition, $(\xi^R)_2 = 2^k y + x$ and $(\theta^R)_2 = 2^k y + x - 1$. Then, there is an integer $l_0 \in [1, k_1]$ such that $\theta_i = \xi_i$ for every $i \in [l_0 + 1, k_1 + k_2]$; $\theta_{l_0} = 0$, and $\xi_{l_0} = 1$. Then, the following claim implies that $\xi \cdot \varphi > \theta \cdot \varphi$; i.e., $m_{x,y} > m_{x-1,y}$.

**Claim 6**. $(\varphi, z_{l_0})_4 > \sum_{i=1}^{l_0 - 1} (\varphi, z_i)_4$.

**Proof of Claim 6** The proof is by induction on $l_0$. Let $l_c = \max\{j: z_{l_c} \in C_j, j \in [1, m_1]\}$. Let $l_b$ be an integer in $[1, 3]$ such that $z_{l_b} \in C_{l_b}$. First, suppose that $l_b = 1$. By lines 4-6 Algorithm 1 $(\varphi, z_{l_b-1}) = (\varphi, z_{l_b-3})$. By lines 8-11 Algorithm 1 $2(\varphi, z_{l_b-1})_4 < (\varphi, z_{l_b})_4$. Thus, $2(\varphi, z_{l_b-1})_4 < (\varphi, z_{l_b})_4$. Next, suppose that $l_b \in \{2, 3\}$. By lines 4-6 Algorithm 1 $(\varphi, z_{l_b-1}) = (\varphi, z_{l_b-3})$. By lines 8-11 Algorithm 1 $2(\varphi, z_{l_b-1})_4 < (\varphi, z_{l_b})_4$. By induction hypothesis, $\sum_{i=1}^{l_b-2} (\varphi, z_i)_4 < (\varphi, z_{l_b})_4$. Thus, $\sum_{i=1}^{l_b-1} (\varphi, z_i)_4 < 2(\varphi, z_{l_b})_4$. Consequently, $\sum_{i=1}^{l_b-1} (\varphi, z_i)_4 < (\varphi, z_{l_b})_4$. □(Claim)

Let us next prove the inequality (5.2). Let $\eta$ be the vector $f^{-1}(x, y - 1)$. By definition, $(\eta^R)_2 = 2^k y + x$ and $(\eta^R)_2 = 2^k (y - 1) + x$. By line 14 Algorithm 1 $k_2 = 3m_1$. There are integers $l_0 \in [1, m_1]$ and $l_1 \in [0, 2]$ such that $\eta_i = \xi_i$ for every $i \in [k_1 + 3l_0 - l_1 - 1, k_1 + k_2]$; $\theta_{k_1 + 3l_0 - l_1} = 0$; and $\xi_{k_1 + 3l_0 - l_1} = 1$. Let $\lambda = k_1 + 3l_0 - l_1$. Then, the following claim implies that $\xi \cdot \varphi > \eta \cdot \varphi$; i.e., $\lambda_{x,y} \geq \lambda_{x-1,y}$.

**Claim 7**. $(\varphi, z_{\lambda})_4 > \sum_{i=k_1+1}^{l-1} (\varphi, z_i)_4$.

**Proof of Claim 7**. The proof is by induction on $\lambda$. In Algorithm 1 operations for $z_{k_1}$ are only in the loop of lines 7-11. Moreover, those operations are only in time when $i = l_0$ during all iterations of that loop. Thus, by lines 8-11 Algorithm 1 $(\varphi, z_{k_1+3l_0-l_1})_4 > \sum_{i=1}^{l_1+1} (\varphi, z_{k_1+3l_0-l_1})_4 + 2(\varphi, z_{k_1+3l_0-l_1})_4$. The proof concludes.
That is, \((\varphi, z_\lambda)_4 > \sum_{i=k_1+3(l_0-1)}^{\lambda-1} (\varphi, z_i)_4 + 2(\varphi, z_{k_1+3(l_0-1)})_4\). By induction hypothesis, \((\varphi, z_{k_1+3(l_0-1)})_4 > \sum_{i=1}^{k_1+3(l_0-1)-1} (\varphi, z_i)_4\). Thus, \(2(\varphi, z_{k_1+3(l_0-1)})_4 > \sum_{i=1}^{k_1+3(l_0-1)} (\varphi, z_i)_4\). Consequently, \((\varphi, z_\lambda)_4 > \sum_{i=1}^{\lambda-1} (\varphi, z_i)_4\). \(\square\) (Claim 7)

**Lemma 8 (Equivalence of formulae).** Let \(\psi\) and \(\varphi\) be as in Algorithm \([4]\). Then, \(\psi\) is satisfiable if and only if \(\varphi\) is satisfiable.

**Proof of Lemma 8.** By line 11 in Algorithm \([4]\) \(\varphi\) contains all clauses in \(\psi\). Thus, the “if” part is trivial. We will prove the “only if” part below. Suppose that \(\psi\) is satisfiable. Let \(\sigma\) be a satisfying assignment for \(\psi\). It suffices to show that there is a satisfying assignment \(\mu\) for \(\varphi\) such that \(\mu_i = \sigma_i\) for every \(i \in [1,k_1]\). Let \(j \in [1,m_1]\). By line 11 in Algorithm \([4]\) \(C_{4j-3}\) is equal to \(C_j\) in \(\psi\). Thus, \(\mu\) satisfies \(C_{4j-3}\) in \(\varphi\). Let us then consider the conjunction of the clauses \(C_{4j} = (z_{k_1+3,j} \lor z_{k_1+3,j-1} \lor z_{j,3})\), \(C_{4j-1} = (z_{k_1+3,j} \lor z_{k_1+3,j-1} \lor z_{j,2})\), and \(C_{4j-2} = (z_{k_1+3,j} \lor z_{j,3} \lor z_{j,2})\) in \(\varphi\). In Algorithm \([4]\) for every \([0,2]\), \(z_{k_1+3,j-1}\) occurs only in \(C_{4j} \land C_{4j-1} \land C_{4j-2}\) in \(\varphi\). Thus, we can assign \(z_{k_1+3,j}\), \(z_{k_1+3,j-1}\), and \(z_{k_1+3,j-2}\) to values without affecting the values of all clauses except for \(C_{4j}, C_{4j-1}\), and \(C_{4j-2}\). First, suppose that \(z_{j,1}\), \(z_{j,2}\), and \(z_{j,3}\) are assigned 1, 0, and 0 respectively. If \(\mu\) assigns \(z_{k_1+3,j}\), \(z_{k_1+3,j-1}\), and \(z_{k_1+3,j-2}\) to 1, 0, and 0 respectively, then \(\mu\) satisfies \(C_{4j}, C_{4j-1}\), and \(C_{4j-2}\). Next, suppose that \(z_{j,1}\), \(z_{j,2}\), and \(z_{j,3}\) are assigned 0, 1, and 0 respectively. If \(\mu\) assigns \(z_{k_1+3,j}\), \(z_{k_1+3,j-1}\), and \(z_{k_1+3,j-2}\) to 0, 1, and 0 respectively, then \(\mu\) satisfies \(C_{4j}, C_{4j-1}\), and \(C_{4j-2}\). Next, suppose that \(z_{j,1}\), \(z_{j,2}\), and \(z_{j,3}\) are assigned 0, 0, and 1 respectively. If \(\mu\) assigns \(z_{k_1+3,j}\), \(z_{k_1+3,j-1}\), and \(z_{k_1+3,j-2}\) to 0, 0, and 1 respectively, then \(\mu\) satisfies \(C_{4j}, C_{4j-1}\), and \(C_{4j-2}\). Consequently, \(\varphi\) is satisfiable. \(\square\)

### 3.4 Running time

In this subsection, we analyze the running time of the proposed algorithm.

**Lemma 9 (Polynomial-time computability).** Given \(\psi\), \(\text{ISSAT}(\psi)\) runs in time polynomial in \(k_1\).

**Proof of Lemma 9.** Each operation of addition, subtraction, multiplication, division, mod, floor, and ceiling can be computed in time polynomial in \(k_1\). Those operations are executed \(H(k,m)\) times, where \(H\) is a linear function of \(k\) and \(m\). In lines 2-18 in Algorithm \([4]\) we spend time as follows. Note that \(m_1 = \Theta(k_1)\).

In line 1, we can count the variables in \(\psi\) in time linear in \(k_1\). In line 2 in Algorithm \([4]\) we count the clause in \(\psi\) in time linear in \(m_1\). For every \(j \in [1,k_1]\), we can compute \((\psi, z_j)_4\) in time polynomial in \(k_1\). Moreover, we can sort the sequence \((\psi, z_1)_4, \ldots, (\psi, z_{k_1})_4\) in ascending order in time polynomial in \(k_1\). After that procedure, we can rename variables in \(\psi\) \((\psi, z_1)_4 < \cdots <\)
(ψ, z_k1) in time linear in k1. That is, we can do the step in line 4 in time polynomial in k1. For every j ∈ [1, m1], we can do the step in line 6 in time linear in k1. Thus, we can do the loop in lines 5-6 in time polynomial in k1. For every j ∈ [1, m1], we can do the step in lines 8-11 in time linear in k1. Thus, we can do the loop in lines 7-11 in time polynomial in k1. In line 15, we compute (φ, z_j) for every j ∈ [1, k1 + 3m1]. That procedure are done in time polynomial in k1. By line 12, m = 4m1. Thus, m = Θ(k1).

In line 16, we can compute \( \sum_{i=1}^{m} 4^{i-1} \) in time polynomial in k1. By line 14, \( k_2 = 3m_1 \). Thus, \( k_2 = \Theta(k_1) \). In line 18, we can compute \( 2^{k_1} \) and \( 2^{k_2} \) in time polynomial in k1. Moreover, by Claim 10 below, we can do the step in line 19 in time polynomial in k1. Consequently, the total running time of IsSat(ψ) is \( O(k_1) \).

**Claim 10.** 2DIBSearch(\( \phi, p, q, k_1, t \)) runs in time polynomial in k1.

**Proof of Claim 10.** Let us first analyze the running time of 2DIBSearch for all steps except for recursive calls. By the above discussion, the number k of components in \( \phi \) is \( O(k_1) \). In line 4, we can count the number of components in \( \phi \) in time polynomial in k1. We can compute the expression in the righthand side in line 5. r takes at most \( O(k_1) \) bits. The variable s has the largest bit length in all variables in Algorithm 2. Its bit length is at most \( O(k_1) \). Thus, all steps in lines 2-30 except for recursive calls can be executed in time polynomial in k1.

In every recursive call, the bit lengths of the first, fourth, and fifth arguments are the same as the one in the calling procedure; and moreover, the ones of the second and third arguments are about the halves of the one in the calling procedure. We denote the sum of the bit lengths for representing \( \phi, k_1, \) and t by \( \lambda_0 \). We denote the sum of the bit lengths for representing \( p \) and \( q \) by \( \lambda \). In the first call for 2DIBSearch, \( \lambda = 2k_1 + 2k_2 \); i.e., 2k. The depth of recursion depends on \( \lambda \), but independent of \( \lambda_0 \). We define \( T_0(\lambda_0) \) as an upper bound for the time of all steps except for recursive calls in 2DIBSearch(\( \phi, p, q, k_1, t \)). We define \( T(\lambda_0, \lambda) \) as an upper bound for the total running time of 2DIBSearch(\( \phi, p, q, k_1, t \)).

In a call of 2DIBSearch, there are the following six cases. Let \( s_{(1)} \) be the value of \( s \) after line 6. \( \square[I] \) \( q_1 - p_1 \leq 1 \) and \( q_2 - p_2 \leq 1 \); i.e., the condition in line 2 is true. \( \square[II] \) \( t = s_{(1)}, q_1 - p_1 \geq 2, \) and \( q_2 - p_2 \geq 2 \); i.e., the condition in line 7 is true. \( \square[III] \) \( t < s_{(1)}, q_1 - p_1 \geq 2, \) and \( q_2 - p_2 \geq 2 \); i.e., the conditions in lines 9 and 10 are true. \( \square[IV] \) \( t < s_{(1)} \) and \( (q_1 - p_1 \leq 1 \) or \( q_2 - p_2 \leq 1 \); i.e., the condition in lines 9 and 10 are true and false, respectively. \( \square[V] \) \( t > s_{(1)}, q_1 - p_1 \geq 2, \) and \( q_2 - p_2 \geq 2 \); i.e., the conditions in lines 20 and 21 are true. \( \square[VI] \) \( t > s_{(1)} \) and \( (q_1 - p_1 \leq 1 \) or \( q_2 - p_2 \leq 1 \); i.e., the condition in lines 20 and 21 are true and false, respectively. Then, we obtain the
following recurrence.

\[
T(\lambda_0, \lambda) = \begin{cases} 
  B(\lambda_0) & \text{in cases (8-I) and (8-II)} \\
  3T\left(\lambda_0, \frac{\lambda}{2}\right) + T_0(\lambda_0) & \text{in cases (8-III) or (8-V)} \\
  T\left(\lambda_0, \frac{\lambda}{2}\right) + T_0(\lambda_0) & \text{in cases (8-IV) or (8-VI)},
\end{cases}
\]

where \(B\) is a polynomial. Let \(T_1(\lambda_0) = B(\lambda_0) + T_0(\lambda_0)\). Then, by Claim 11 below, \(T(\lambda_0, \lambda)\) is of polynomial order in \(\lambda\) and \(\lambda_0\). By the above discussion for Algorithm 1, \(\lambda_0\) and \(\lambda\) are of polynomial order in \(k_1\). Consequently, \(2\text{DIBSEARCH}(\bar{\phi}, p, q, k_1, t)\) runs in time polynomial in \(k_1\). \(\square\) (Claim 10)

Claim 11. \(T(\lambda_0, \lambda) \leq \lambda^u T_1(\lambda_0)\), where \(u = \frac{1}{\log_3 2}\).

Proof of Claim 11. The proof is by induction on \(\lambda\). By the definition of \(T_1\), \(B(\lambda_0) < T_1(\lambda_0)\). Thus, in cases (8-I) and (8-II), \(T(\lambda_0, \lambda) \leq \lambda^u T_1(\lambda_0)\). In cases (8-III) to (8-VI), \(T(\lambda_0, \lambda) \leq 3T(\lambda_0, \lambda/2) + T_0(\lambda_0)\). By induction hypothesis, \(T(\lambda_0, \lambda) \leq 3(\lambda/4)^u T_1(\lambda_0) + T_0(\lambda_0)\). By a rearrangement, \(3(\lambda/4)^u T_1(\lambda_0) + T_0(\lambda_0) = \lambda^u ((3/4)^u T_1(\lambda_0) + (1/\lambda^u) T_0(\lambda_0))\). By the definition of \(T_1\), \(T_0(\lambda_0) < T_1(\lambda_0)\). By definition, \(\lambda \geq 4\). Thus, \((3/4)^u + (1/\lambda^u) \leq 1\). It follows that \(3(\lambda/4)^u T_1(\lambda_0) + T_0(\lambda_0) \leq \lambda^u T_1(\lambda_0)\). That is, \(T(\lambda_0, \lambda) \leq \lambda^u T_1(\lambda_0)\). \(\square\) (Claim 11)

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: a Modern Approach*. Cambridge University Press, New York, NY, 2009.

[2] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. SIAM, 2001.

[3] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC ’78, pages 216–226, New York, NY, USA, 1978. ACM.

[4] Ingo Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, Inc., New York, NY, USA, 1987.