Improved Exact Algorithms for Mildly Sparse Instances of Max SAT*

Takayuki Sakai[1], Kazuhisa Seto[2], Suguru Tamaki[3], and Junichi Teruyama[4]

1 **Oki Electric Industry Co., Ltd.**
2 **Seikei University, 3-3-1 Kichijoji-Kitamachi, Musashino-shi, Tokyo 180-8633, Japan**
`seto@st.seikei.ac.jp`
3 **Kyoto University, Yoshida Honmachi, Sakyo-ku, Kyoto 606-8501, Japan**
`tamak@kuis.kyoto-u.ac.jp`
4 **National Institute of Informatics, and JST, ERATO, Kawarabayashi Large Graph Project, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan**
`teruyama@nii.ac.jp`

Abstract

We present improved exponential time exact algorithms for Max SAT. Our algorithms run in time of the form $O(2^{(1-\mu(c))n})$ for instances with n variables and $m = cn$ clauses. In this setting, there are three incomparable currently best algorithms: a deterministic exponential space algorithm with $\mu(c) = \frac{1}{O(c \log c)}$ due to Dantsin and Wolpert [SAT 2006], a randomized polynomial space algorithm with $\mu(c) = \frac{1}{O(c \log^3 c)}$ and a deterministic polynomial space algorithm with $\mu(c) = \frac{1}{O(c^2 \log^2 c)}$ due to Sakai, Seto and Tamaki [Theory Comput. Syst., 2015]. Our first result is a deterministic polynomial space algorithm with $\mu(c) = \frac{1}{O(c \log c)}$ that achieves the previous best time complexity without exponential space or randomization. Furthermore, this algorithm can handle instances with exponentially large weights and hard constraints. The previous algorithms and our deterministic polynomial space algorithm run super-polynomially faster than 2^n only if $m = O(n^2)$. Our second results are deterministic exponential space algorithms for Max SAT with $\mu(c) = \frac{1}{O((c \log c)^{2/3})}$ and for Max 3-SAT with $\mu(c) = \frac{1}{O(c^{1/2})}$ that run super-polynomially faster than 2^n when $m = o(n^{5/2}/\log^{5/2} n)$ and $m = o(n^3/\log^2 n)$ respectively.

1998 ACM Subject Classification F.2.0 [Analysis of Algorithms and Problem Complexity]: General

Keywords and phrases maximum satisfiability, weighted, polynomial space, exponential space

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.90

1 Introduction

The *maximum satisfiability problem* (Max SAT) is, given a set of clauses, to find an assignment to Boolean variables that maximizes the number of satisfied clauses, where a *clause* is a disjunction of literals, a *literal* is a Boolean variable or its negation, and an assignment *satisfies* a clause if at least one literal in the clause becomes true under the assignment. Max SAT is one of the most fundamental problems in practice and theory. It is known to be

* This work was supported in part by MEXT KAKENHI (24106003); JSPS KAKENHI (25240002, 26330011, 26730007); JST ERATO Kawarabayashi Large Graph Project.

Table 1 A historical overview of upper bounds. k: an objective value, i.e., the number of constraints that must be satisfied. l: the length of an instance, i.e., the sum of arities of constraints. m: the number of constraints. n: the number of variables.

Running time	Problem	Space	Reference
$O(2^{0.4414k})$	Max SAT	polynomial	[3]
$O(2^{0.1000l})$	Max 2-SAT	polynomial	[13]
$O(2^{0.1450l})$	Max SAT	polynomial	[1]
$O(2^{0.1583m})$	Max 2-SAT	polynomial	[10]
$O(2^{0.1801m})$	Max 2-CSP	polynomial	[11]
$O(2^{0.4057m})$	Max SAT	polynomial	[5]
$O(2^{0.7909n})$	Max 2-CSP	exponential	[21, 35]
$O(2^{(1-\alpha(\frac{m}{n}))n}), \alpha(c) = \frac{1}{O(c/\log c)}$	Max 2-CSP	polynomial	[12]
$O(2^{(1-\beta(\frac{m}{n}))n}), \beta(c) = \frac{1}{O(c \log c)}$	Max SAT	exponential	[9]
$O(2^{(1-\gamma(\frac{m}{n}))n}), \gamma(c) = \frac{1}{O(2^{O(c)})}$	Max SAT	polynomial	[24]
$O(2^{(1-\delta(\frac{m}{n}))n}), \delta(c) = \frac{1}{O(c \log^3 c)}$	Max SAT	polynomial	[28] (randomized)
$O(2^{(1-\epsilon(\frac{m}{n}))n}), \epsilon(c) = \frac{1}{O(c^2 \log^2 c)}$	Max SAT	polynomial	[28] (deterministic)
$O(2^{(1-\zeta(\frac{m}{n}))n}), \zeta(c) = \frac{1}{O(c \log c)}$	Max SAT	polynomial	Theorem 1
$O(2^{(1-\eta(\frac{m}{n}))n}), \eta(c) = \frac{1}{O((c \log c)^{2/3})}$	Max SAT	exponential	Theorem 2
$O(2^{(1-\iota(\frac{m}{n}))n}), \iota(c) = \frac{1}{O(c^{1/2})}$	Max 3-SAT	exponential	Theorem 3

NP-hard. Max ℓ-SAT is a special case of Max SAT with the restriction that each clause contains at most ℓ literals. It is also NP-hard even when $\ell = 2$.

The time complexities of Max SAT and Max CSP (constraint satisfaction problem) have been studied with respect to several parameters such as an objective value k, i.e., the number of constraints that must be satisfied, the length of an instance l, i.e., the sum of arities of constraints, the number of constraints m, and the number of variables n, see, e.g., [1, 2, 3, 5, 9, 10, 11, 12, 13, 14, 16, 17, 19, 20, 21, 22, 23, 24, 26, 27, 28, 32, 33, 35]. We summarize the previous and our results in Table 1. We omit polynomial factors with respect to complexity parameters in the table. Recall that Max CSP is a generalization of Max SAT, where an instance consists of a set of arbitrary constraints instead of clauses. In Max ℓ-CSP, each constraint depends on at most ℓ variables.

An alternative way to parametrize MAX SAT is to ask whether at least $\tilde{m} + k$ clauses can be satisfied, where $\tilde{m}$ is the expected number of satisfied constraints by a uniformly random assignment, see, e.g., [15]. As for a special case of Max SAT, it is known that the satisfiability problem of CNF formulas with n variables and cn clauses can be solved in time $2^{(1-\mu(c))n}$, where $\mu(c) = 1/O(\log c)$, see [8]. As for more general problems than Max SAT, Impagliazzo, Paturi and Schneider [18] showed a satisfiability algorithm for depth-2 threshold circuits that runs in time $2^{(1-\mu(c))n}$ and exponential space for circuits with n variables and cn wires, where $\mu(c) = 1/c^{O(c^2)}$.

In this paper, we consider n and m as complexity parameters. This choice is appropriate for instances with $m = cn$ clauses, where $c > 0$ is unbounded. Given an instance with n variables and $m = cn$ clauses, Max SAT can be solved in time $\mathrm{poly}(m) \cdot 2^n$. Our goal is to design algorithms that run in time of the form $\mathrm{poly}(m) \cdot 2^{(1-\mu(c))n}$ with $\mu(c) > 0$ as large as possible. In this setting, there are three incomparable currently best algorithms: a deterministic exponential space algorithm with $\mu(c) = \frac{1}{O(c \log c)}$ due to Dantsin and Wolpert [9], a randomized polynomial space algorithm with $\mu(c) = \frac{1}{O(c \log^3 c)}$ and a deterministic polynomial space algorithm with $\mu(c) = \frac{1}{O(c^2 \log^2 c)}$ due to Sakai, Seto and Tamaki [28].

In this paper, we present an algorithm that achieves the previous best time complexity without exponential space or randomization as follows.

▶ **Theorem 1** (Polynomial Space). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max SAT can be solved deterministically in time* $\mathrm{poly}(m, \log W) \cdot 2^{(1-\mu(c))n}$ *and polynomial space, where* $\mu(c) = \frac{1}{O(c \log c)}$.

Furthermore, this algorithm can handle instances with hard constraints as [28]. The previous algorithms and our deterministic polynoamial space algorithm run super-polynomially faster than 2^n only if $m = O(n^2)$. In this paper, we present algorithms that run super-polynomially faster than 2^n for instances with $\omega(n^2)$ clauses with the help of exponential space as follows.

▶ **Theorem 2** (Exponential Space). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max SAT can be solved deterministically in time* $\mathrm{poly}(m, W) \cdot 2^{(1-\mu(c))n}$ *and exponential space, where* $\mu(c) = \frac{1}{O((c \log c)^{2/3})}$.

▶ **Theorem 3** (Exponential Space, Max 3-SAT). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max 3-SAT can be solved deterministically in time* $\mathrm{poly}(m, W) \cdot 2^{(1-\mu(c))n}$ *and exponential space, where* $\mu(c) = \frac{1}{O(c^{1/2})}$.

These algorithms run super-polynomially faster than 2^n for instances of Max SAT and Max 3-SAT with $m = o(n^{5/2}/\log^{5/2} n)$ and $m = o(n^3/\log^2 n)$ respectively. They can handle instances with hard constraints as well.

After this paper was submitted to this conference, the authors learned that Chen and Santhanam [7] independently obtained similar but better results than ours, i.e., a deterministic polynomial space algorithm with $\mu(c) = \frac{1}{O(c)}$ and a deterministic exponential space algorithm with $\mu(c) = \frac{1}{O(c^{1/2})}$.

1.1 Our technique

Our algorithms are based on the combination of width reduction and greedy restriction due to Sakai, Seto and Tamaki [28]. The basic idea is as follows:

1. Width reduction. Given an instance of Max SAT with n variables and m clauses, we produce a collection of instances of Max ℓ-SAT with $\ell = O(\log(m/n))$ such that the optimum of the original instance is equal to the maximum of the optima of the produced instances. Width reduction is originally invented by Schuler [31] to solve the CNF satisfiability problem, but we can modify it to handle Max SAT.

2. Greedy restriction. For each Max ℓ-SAT instance, we repeat the following until Max 1-SAT instances are obtained: Pick a variable x that appears most frequently in clauses that contain at least 2 literals, then produce two instances by setting $x = 0$ and $x = 1$. Greedy restriction is inspired by the satisfiability algorithms for Boolean formulas due to Santhanam and others [6, 30, 34].

3. Max 1-SAT. We solve Max 1-SAT instances by a polynomial time algorithm (majority voting).

In the previous analysis of greedy restriction [28], they regard each clause as a De Morgan formula and apply the so-called shrinkage lemma for De Morgan formulas due to Chen, Kabanets, Kolokolova, Shaltiel and Zuckerman [6]. In this paper, we treat each clause as

it is and improve the analysis of greedy restriction. Furthermore, if we are allowed to use exponential space, we can replace the base case of Max 1-SAT by Max 2-SAT and apply Williams' algorithm for Max 2-SAT [21, 35]. This further improves the efficiency of greedy restriction since we only have to consider clauses that contain at least 3 literals instead of at least 2 literals. Note that there is a difference between the polynomial time algorithm for Max 1-SAT and Williams' algorithm for Max 2-SAT with respect to the dependency on the maximum weight W of instances: The running time of the former and the latter involve $\mathrm{poly}(\log W)$ and $\mathrm{poly}(W)$ factors respectively as seen in Theorem 1 and Theorems 2 and 3. If $(1-\varepsilon)$-multiplicative approximation is allowed, the dependency is improved to $\mathrm{poly}(\log W)$ in Williams' algorithm, see Section 4.2 in [35].

2 Preliminaries

We denote by $\mathbb{Z}$ the set of integers. We define $-\infty$ as $-\infty + z = z + -\infty = -\infty$ and $-\infty < z$ for all $z \in \mathbb{Z}$.

Let V be a set of Boolean variables $\{x_1, \ldots, x_n\}$. We use the value 1 to indicate Boolean 'true', and 0 'false'. The *negation* of a variable $x \in V$ is denoted by $\overline{x}$. A *literal* is either a variable or its negation. An *ℓ-constraint* is a Boolean function $\phi : \{0,1\}^\ell \to \{0,1\}$, which depends on ℓ variables in V. The *width* of ϕ is ℓ. Note that a 0-constraint is either '0' or '1' (a constant function). An *ℓ-clause* is an ℓ-constraint represented as a disjunction of ℓ literals, i.e., $y_1 \vee \cdots \vee y_\ell$ for some literals $y_1, \ldots, y_\ell$.

An *instance* Φ of Max SAT consists of pairs of a constraint and a weight function, i.e., $\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$, where each ϕ_i is an ℓ_i-clause and $w_i : \{0,1\} \to \{-\infty\} \cup \mathbb{Z}$. We allow 0-constraints to appear in instances of Max SAT. The *width* of Φ is $\max_i \ell_i$. In Max ℓ-SAT, each instance has width at most ℓ. The *maximum weight* of Φ is $\max_{i,a:w_i(a)\neq-\infty} |w_i(a)|$. For a weight function w, we denote by $\widetilde{w}$ the weight function defined as $\widetilde{w}(1) := w(1), \widetilde{w}(0) := -\infty$. Note that a constraint with $w_i(0) = -\infty$ ($w_i(1) = -\infty$, resp.) must be satisfied (unsatisfied, resp.), i.e., it is a *hard constraint*. Without loss of generality, we do not consider instances with $w_i(0) = w_i(1) = -\infty$ for some i. We use the notation as $\mathrm{Val}(\Phi, a) := \sum_{i=1}^m w_i(\phi_i(a))$ and $\mathrm{Opt}(\Phi) := \max_{a\in\{0,1\}^n} \mathrm{Val}(\Phi, a)$.

The *length* of Φ, denoted by $L(\Phi)$, is defined as the sum of widths of clauses, i.e., $L(\Phi) := \sum_{i=1}^m \ell_i$. More generally, we define $L_k(\Phi) := \sum_{i:\ell_i \geq k} \ell_i$. We denote by $\mathrm{var}(\Phi)$ the set of variables that appear as literals in Φ. The *frequency* of a variable x with respect to Φ is the number of literals that appear in Φ as x or $\overline{x}$, and denoted by $\mathrm{freq}(\Phi, x)$. We define $\mathrm{var}_k(\Phi)$ and $\mathrm{freq}_k(\Phi, x)$ analogously to $L_k(\Phi)$.

For any instance Φ of Max SAT, any set of variables $\{x_{i_1}, \ldots, x_{i_k}\}$ and any constants $a_1, \ldots, a_k \in \{0,1\}$, we denote by $\Phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]$ the instance obtained from Φ by assigning to each x_{i_j}, $\overline{x}_{i_j}$ the value a_j, $\overline{a}_j$ respectively and applying the following simplification rules repeatedly:

1. If Φ contains a clause of the form $\phi = 0 \vee \psi$ where ψ is a disjunction of literals, replace ϕ by the clause ψ.
2. If Φ contains a clause of the form $\phi = 1 \vee \psi$ where ψ is a disjunction of literals, replace ϕ by 1 (as a 0-constraint).

We similarly define $\phi[l_{i_1} = a_1, \ldots, l_{i_k} = a_k]$ for any set of literals $\{l_{i_1}, \ldots, l_{i_k}\}$.

3 Algorithms for Max 1-SAT and Max 2-SAT

In this section, we introduce a polynomial time algorithm for Max 1-SAT and an exponential space algorithm for Max 2-SAT. These algorithms serve as the base cases respectively in the

polynomial and exponential space algorithms for Max ℓ-SAT in the next section.

Our polynomial time algorithm for Max 1-SAT is based on simple majority voting.

▶ **Lemma 4** (Max 1-SAT). *Given an instance of Max 1-SAT* Φ ($L_2(\Phi) = 0$) *with* m *clauses and the maximum weight* W*,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, \log W)$.

Proof. Let Φ be an instance $\{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ of Max 1-SAT. Since the width of Φ is at most 1, each ϕ_i is either '0', '1', 'x_j' or '$\overline{x}_j$' (for some j). For each x_i, define $S_i(0), S_i(1)$ as

$$\begin{aligned} S_i(0) &:= \sum_{j:\phi_j = x_i} w_j(0) + \sum_{j:\phi_j = \overline{x}_i} w_j(1), \\ S_i(1) &:= \sum_{j:\phi_j = x_i} w_j(1) + \sum_{j:\phi_j = \overline{x}_i} w_j(0). \end{aligned}$$

If $S_i(0) = S_i(1)$, then set $x_i = *$ (don't care), if $S_i(0) > S_i(1)$, then set $x_i = 0$, otherwise set $x_i = 1$. This assignment achieves $\mathrm{Opt}(\Phi)$, where we assign arbitrary values to don't care variables. Operations such as addition and comparison can be done in time $\mathrm{poly}(m, \log W)$. ◀

Our exponential space algorithm for Max 2-SAT is based on Williams' algorithm for Max 2-SAT [21, 35].

▶ **Lemma 5** (Max 2-SAT). *Given an instance of Max 2-SAT* Φ ($L_3(\Phi) = 0$) *with* n *variables,* m *clauses and the maximum weight* W*,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, W) \cdot 3^{n/2}$ *and exponential space.*

Proof. We use the following result.

▶ **Theorem 6** ([21, 35]). *Given an instance of Max 2-SAT with* n *variables,* m *clauses and the maximum weight* W*, where each constraint is different from the others and each weight function* w_i *satisfies* $w_i(1) > 0$ *and* $w_i(0) = 0$*,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, W) \cdot 2^{\omega n/3}$ *and exponential space, where* $\omega < 2.3728639$ [25] denotes the exponent of the matrix multiplication.

Before applying Williams' algorithm, we need preprocessing on a given instance because our instances are more general than those considered in [21, 35]. That is, we consider instances that contain two or more identical clauses, arbitrary weighted functions and hard constraints. We transform instances to a set of instances that do not contain identical clauses and whose weight functions are of the form $w_i(1) > 0$ and $w_i(0) = 0$. We ensure that the optimum of the original instance is equal to the maximum of the optima of the resulting instances. The preprocessing increases the running time, i.e., $3^{n/2} > 2^{\omega n/3}$.

Let Φ be an instance $\{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ of Max 2-SAT. For simplicity, we treat only instances with $w_i(1) \geq w_i(0)$ for all i. The restriction on instances can be removed with additional transformation.

First, we replace each $w_i(a)$ with $w_i(0) \neq -\infty$ by $w_i(a) - w_i(0)$ to satisfy $w_i(1) > 0$ and $w_i(0) = 0$. This transformation increases the maximum weight of the instance by at most twice. We define $W_0 := \sum_{i:w_i(0) \neq -\infty} w_i(0)$ and use this later to reset the offset.

Next, we reduce the number of clauses that appear twice or more in the instance as follows: If the instance contains (ϕ_i, w_i) and (ϕ_j, w_j) such that $\phi_i = \phi_j$, then replace (ϕ_i, w_i) by $(\phi_i, w_i + w_j)$ and remove (ϕ_j, w_j). The maximum weight of the instance becomes at most $O(mW)$.

Finally, we remove hard constraints and apply Williams' algorithm. To do so, we consider a *maximal independent set* of hard constraints constructed as follows: Set $I = \emptyset$ and repeat the following: Pick arbitrary (ϕ_i, w_i) in Φ such that $w_i(0) = -\infty$ and add it to I if ϕ_i is *independent* of I, i.e., the literals in ϕ_i do not appear in any clause in I. When I becomes a maximal independent set, the cardinality of I, denoted by $|I|$, is at most $n/2$.

We try every assignment that satisfies all the clauses in I one by one. The number of such assignments is $3^{|I|}$. Note that if we assign values to all the literals that appear in some clause in I, then we obtain an instance whose hard constraints are 0 or 1-constraints. Since hard 1-constraints only determine the values of some variables and hard 0-constraints determine the feasibility of the instance, now we can apply Williams' algorithm. Note that we must add W_0 to the result of Williams' algorithm to obtain $\mathrm{Opt}(\Phi)$. The overall running time is $\mathrm{poly}(m, W) \cdot 3^{|I|} \cdot 2^{(\omega/3)(n-2|I|)} \leq \mathrm{poly}(m, W) \cdot 3^{n/2}$. ◀

4 Greedy Restriction Algorithms for Max ℓ-SAT

In this section, we present polynomial and exponential space algorithms for Max ℓ-SAT based on the combination of greedy restriction and the algorithms in the previous section. These serve as subroutines in our main algorithms for Max SAT. For Max 3-SAT, our exponential space algorithm for Max ℓ-SAT runs in time as stated in Theorem 3. First, we introduce shrinkage lemmas that are useful in the analysis of greedy restriction. Then, we describe our algorithms and their analyses.

4.1 Shrinkage Lemmas

In this section, we provide shrinkage lemmas that are useful in upper-bounding the length of instances with respect to $L_k(\cdot)$ after a sequence of greedy restriction. For an instance Φ of Max SAT with n variables, we define a sequence of random variables $\Phi_0, \Phi_1, \ldots, \Phi_n$ inductively as follows: $\Phi_0 := \Phi$. Given $\Phi_0, \Phi_1, \ldots, \Phi_{i-1}$, $\Phi_i := \Phi_{i-1}[x = a]$, where $x = \arg\max_{x \in \mathrm{var}(\Phi_{i-1})} \mathrm{freq}_k(\Phi_{i-1}, x)$ and a is a uniform random bit. We have the following lemma.

▶ **Lemma 7** (Shrinkage of Max SAT instances with respect to $L_k(\cdot)$, Lemma 4.2 in [29]). *For $n' \geq 4$, we have*

$$\Pr\left[L_k(\Phi_{n-n'}) \geq 2^k \cdot L_k(\Phi) \cdot \left(\frac{n'}{n}\right)^{\frac{k+2}{2}}\right] < 2^{-n'}.$$

Note that the shrinkage lemma used in [28] is only for $k = 2$ and provides the bound

$$\Pr\left[L_2(\Phi_{n-n'}) \geq 2 \cdot L_2(\Phi) \cdot \left(\frac{n'}{n}\right)^{\frac{3}{2}}\right] < 2^{-n'}.$$

Thus, Lemma 7 generalizes the above by introducing a parameter k and also improves the bound of it. For instances of Max 3-SAT with n variables and $k = 3$, we further improve the bound of Lemma 7 as follows.

▶ **Lemma 8** (Shrinkage of Max 3-SAT instances with respect to $L_3(\cdot)$). *For $n' \geq 1$, we have*

$$L_3(\Phi_{n-n'}) \leq L_3(\Phi) \cdot \left(\frac{n'}{n}\right)^3.$$

MaxℓSAT($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: instance, n, n': integer)
01: **if** $n > n'$, **then**
02: $x \leftarrow \arg\max_{x \in \mathrm{var}(\Phi)} \mathrm{freq}_2(\Phi, x)$.
03: $K_0 \leftarrow$ **MaxℓSAT**$(\Phi[x = 0], n - 1, n')$.
04: $K_1 \leftarrow$ **MaxℓSAT**$(\Phi[x = 1], n - 1, n')$.
05: **return** $\max\{K_0, K_1\}$.
06: **else**
07: **return** $\mathrm{Opt}(\Phi)$ by Lemma 9.

Figure 1 Algorithm for Max ℓ-SAT.

Proof. Let $x = \arg\max_{x \in \mathrm{var}(\Phi_i)} \mathrm{freq}_3(\Phi_i, x)$. For all $a \in \{0, 1\}$, we have $L_3(\Phi_i[x = a]) = L_3(\Phi_i) - 3 \cdot \mathrm{freq}_3(\Phi_i, x)$ because if a clause of width 3 contains x as a literal, it becomes a clause of width either 0 or 2 by assigning a to x. Since $\mathrm{freq}_3(\Phi_i, x) \geq \frac{L_3(\Phi_i)}{n-i}$, for all $a \in \{0, 1\}$, we have

$$L_3(\Phi_i[x = a]) = L_3(\Phi_i) - 3 \cdot \mathrm{freq}(\Phi_i, x) \leq \left(1 - \frac{3}{n-i}\right) \cdot L_3(\Phi_i) \leq L_3(\Phi_i) \cdot \left(1 - \frac{1}{n-i}\right)^3.$$

We complete the proof by induction. ◀

4.2 A Polynomial Space Algorithm

Before presenting our polynomial space algorithm for Max ℓ-SAT, we show a simple algorithm for Max SAT that is efficient for instances with a small number of clauses of width at least 2.

▶ **Lemma 9** (Algorithm for "almost" Max 1-SAT instances). *Given an instance Φ of Max ℓ-SAT with m clauses and the maximum weight W, $\mathrm{Opt}(\Phi)$ can be computed in time $\mathrm{poly}(m, \log W) \cdot 2^{L_2(\Phi)}$.*

Proof. Recall that $\mathrm{var}_2(\Phi)$ is the set of variables that appear in clauses of width at least two. We assume $\mathrm{var}_2(\Phi) = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{|\mathrm{var}_2(\Phi)|}}\}$. For each assignment $a_1, a_2, \ldots, a_{|\mathrm{var}_2(\Phi)|} \in \{0, 1\}$ to $\mathrm{var}_2(\Phi)$, we can compute $\mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_2(\Phi)|}} = a_{|\mathrm{var}_2(\Phi)|}])$ in time $\mathrm{poly}(m, \log W)$ by Lemma 4 since $L_2(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_2(\Phi)|}} = a_{|\mathrm{var}_2(\Phi)|}]) = 0$. Define

$$K := \max_{a_1, a_2, \ldots, a_{|\mathrm{var}_2(\Phi)|} \in \{0,1\}} \mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_2(\Phi)|}} = a_{|\mathrm{var}_2(\Phi)|}]),$$

then $\mathrm{Opt}(\Phi) = K$ holds. The value K can be obtained in time $\mathrm{poly}(m, \log W) \cdot 2^{L_2(\Phi)}$ since $|\mathrm{var}_2(\Phi)| \leq L_2(\Phi)$. ◀

Our polynomial space algorithm for Max ℓ-SAT is shown in Fig. 1. The same algorithm and its analysis for the so-called Max De Morgan formula SAT was given by [28]. We analyze the running time of the algorithm only for instances of Max ℓ-SAT, which is a special case of Max De Morgan formula SAT, and obtain a better upper bound than that of [28]. Note that we may break ties arbitrarily in arg max in Line 02. In particular, when $\mathrm{var}_2(\Phi) = \emptyset$, we may choose any variable x. In what follows, we give the running time analysis of the algorithm.

▶ **Lemma 10** (Polynomial Space Algorithm for Max ℓ-SAT). *Given an instance Φ of Max ℓ-SAT with n variables, $m = cn$ clauses and the maximum weight W, $\mathrm{Opt}(\Phi)$ can be computed in time $\mathrm{poly}(m, \log W) \cdot 2^{\left(1 - \frac{1}{16c\ell}\right)n}$ and polynomial space.*

Proof. Let Φ be an instance $\{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ of Max ℓ-SAT with n variables, $m = cn$ clauses and the maximum weight W. Note that $L_2(\Phi) \leq \ell m = c\ell n$. We set the parameter n' in the algorithm as $n' = \frac{n}{8c\ell}$. For simplicity, we assume n' is an integer.

Let us regard the computation of the algorithm as a complete binary tree of depth $n - n'$ inductively defined as follows. The root is labeled with Φ. The left and the right children of the root are labeled with $\Phi[x = 0]$ and $\Phi[x = 1]$ respectively, where $x = \arg\max_{x \in \mathrm{var}(\Phi)} \mathrm{freq}_2(\Phi, x)$. We define the children of $\Phi[x = 0]$ and $\Phi[x = 1]$ in the similar way and so on until depth $n - n'$ is reached.

To upper-bound the running time of the algorithm, we are interested in $L_2(\cdot)$ of instances that appear as leaves. Pick any leaf and assume its label is Ψ. Then, the algorithm executes Lines 06–07 on Ψ and it takes $\mathrm{poly}(m, \log W) \cdot 2^{L_2(\Phi)}$ time.

To estimate the average of $L_2(\Psi)$, we can apply Lemma 7 because the random sequence $\{\Phi_i\}$ is exactly associated with the complete binary tree defined above. In particular, if we pick a leaf of the tree uniformly at random, then the distribution of its label is exactly the same as that of $\Phi_{n-n'}$.

By Lemma 7 with $k = 2$, we can bound the fraction of leaves labeled with Ψ, $L_2(\Psi) \geq n'/2$, from above by $2^{-n'}$ due to the choice of n'. Thus, the overall running time of the algorithm is at most

$$\mathrm{poly}(m, \log W) \cdot 2^{n-n'} \cdot \{2^{-n'} \cdot 2^{n'} + (1 - 2^{-n'}) \cdot 2^{n'/2}\} = \mathrm{poly}(m, \log W) \cdot 2^{\left(1 - \frac{1}{16c\ell}\right)n}. \quad \blacktriangleleft$$

4.3 An Exponential Space Algorithm

Our exponential space algorithm for Max ℓ-SAT is almost the same as our polynomial space algorithm except that we use $L_3(\cdot)$ and the following lemma instead of $L_2(\cdot)$ and Lemma 9.

▶ **Lemma 11** (Algorithm for "almost" Max 2-SAT instances). *Given an instance Φ of Max ℓ-SAT with n variables, m clauses and the maximum weight W,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, W) \cdot 2^{L_3(\Phi)} \cdot 3^{(n - L_3(\Phi))/2}$ *and exponential space.*

Proof. Assume $\mathrm{var}_3(\Phi) = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{|\mathrm{var}_3(\Phi)|}}\}$. For each assignment $a_1, a_2, \ldots, a_{|\mathrm{var}_3(\Phi)|} \in \{0, 1\}$ to $\mathrm{var}_3(\Phi)$, we can compute $\mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_3(\Phi)|}} = a_{|\mathrm{var}_3(\Phi)|}])$ in time $\mathrm{poly}(m, W) \cdot 3^{(n - |\mathrm{var}_3(\Phi)|)/2}$ and exponentiail space by Lemma 5 since $L_3(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_3(\Phi)|}} = a_{|\mathrm{var}_3(\Phi)|}]) = 0$. Define

$$K := \max_{a_1, a_2, \ldots, a_{|\mathrm{var}_3(\Phi)|} \in \{0,1\}} \mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_3(\Phi)|}} = a_{|\mathrm{var}_3(\Phi)|}]),$$

then $\mathrm{Opt}(\Phi) = K$ holds. The value K can be obtained in time $\mathrm{poly}(m, W) \cdot 2^{L_3(\Phi)} \cdot 3^{(n - L_3(\Phi))/2}$ and exponential space since $|\mathrm{var}_3(\Phi)| \leq L_3(\Phi)$. $\blacktriangleleft$

We modify **MaxℓSAT** in Fig. 1 as follows: (1) In Line 02, replace freq_2 by freq_3, (2) in Line 07, replace Lemma 9 by Lemma 11. The resulting algorithm yields the following.

▶ **Lemma 12** (Exponential Space Algorithm for Max ℓ-SAT). *Given an instance of Max ℓ-SAT with n variables, $m = cn$ constrains and the maximum weight W,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, W) \cdot 2^{(1 - \mu(c))n}$ *and exponential space, where* $\mu(c) = \frac{1}{O((c\ell)^{2/3})}$.

▶ **Theorem 13** (Restatement of Theorem 3). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max 3-SAT can be solved deterministically in time* $\mathrm{poly}(m, W) \cdot 2^{(1 - \mu(c))n}$ *and exponential space, where* $\mu(c) = \frac{1}{O(c^{1/2})}$.

The proofs of the above lemma and theorem are almost as the same as that of Lemma 10 except that we apply Lemma 7 with $k = 3, n' = \frac{n}{(16c\ell)^{2/3}}$ and Lemma 8 with $n' = \frac{n}{\sqrt{6c}}$ respectively instead of Lemma 7 with $k = 2, n' = \frac{n}{8c\ell}$.

MaxSAT($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: **instance,** n, ℓ: **integer**)
01: **if** $L_{\ell+1}(\Phi) = 0$, **then**
02: **return Max**ℓ**SAT**(Φ, n, n'). /* $n' = \frac{n}{8(m/n)\ell}$ */
03: **else**
04: Pick arbitrary $\phi_i = (l_1 \vee \cdots \vee l_{\ell'})$ such that $\ell' > \ell$.
05: $\Phi_L \leftarrow \{\Phi \setminus \{(\phi_i, w_i)\}\} \cup \{(l_1 \vee \cdots \vee l_\ell, \widetilde{w_i})\}$.
06: $K_L \leftarrow$ **MaxSAT**(Φ_L, n, ℓ).
07: $\Phi_R \leftarrow \Phi[l_1 = \cdots = l_\ell = 0]$.
08: $K_R \leftarrow$ **MaxSAT**$(\Phi_R, n - \ell, \ell)$.
09: return $\max\{K_L, K_R\}$.

Figure 2 Max SAT algorithm.

5 Width Reduction Algorithms for Max SAT

In this section, we present polynomial and exponential space algorithms for Max SAT based on the combination of width reduction and the algorithms in the previous section, completing the proof of Theorems 1 and 2.

Our polynomial space algorithm for Max SAT is shown in Fig. 2. Again, the same algorithm and its analysis was given by [28] but we obtain a better upper bound on the running time because we use a faster Max ℓ-SAT algorithm than that of [28] as a subroutine. In what follows, we show the running time analysis of the algorithm.

▶ **Theorem 14** (Restatement of Theorem 1). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max SAT can be solved deterministically in time* $\mathrm{poly}(m, \log W) \cdot 2^{(1-\mu(c))n}$ *and polynomial space, where* $\mu(c) = \frac{1}{O(c \log c)}$.

Proof. The overall structure of the proof is similar to the analysis of width reduction for the CNF satisfiability problem due to Calabro et al. [4]. We think of the execution of **MaxSAT** as a rooted binary tree T, i.e., the root of T is labeled with an input instance Φ and for each node labeled with Ψ, its left (right, resp.) child is labeled with Ψ_L (Ψ_R, resp.) as defined in Line 05 (Line 07, resp.) in the algorithm. If Ψ is an instance of Max ℓ-SAT, i.e., every clause ψ_i in Ψ has width at most ℓ, then the node labeled with Ψ is a leaf.

Let us consider a path p from the root to a leaf v labeled with Ψ. We denote by L and R the number of left and right children p selects to reach v. It is easy to see that (1) $L \leq m$ since the number of clauses is m, (2) $R \leq n/\ell$ since a right branch eliminates ℓ variables at a time, and (3) Ψ is defined over at most $n - R\ell$ variables. Furthermore, the number of leaves which are reachable by exactly R times of right branches is at most $\binom{m+R}{R}$. Let $T(n, m, W, \ell)$ denote the running time of **Max**ℓ**SAT** on instances of Max ℓ-SAT with n variables, $m = cn$ clauses and the maximum weight W due to Lemma 10. We can upper bound the running time of **MaxSAT** as:

$$\mathrm{poly}(m, \log W) \left(\sum_{R=0}^{\frac{n}{2\ell}-1} \binom{m+R}{R} T(n - R\ell, m, W, \ell) + \sum_{R=\frac{n}{2\ell}}^{\frac{n}{\ell}} \binom{m+R}{R} T(n - R\ell, m, W, \ell) \right).$$

In what follows, we omit $\mathrm{poly}(m, \log W)$ factors due to space limitations. That is, the following inequalities hold if we ignore $\mathrm{poly}(m, \log W)$ factors. We first upper bound the second summation above.

$$\left(\sum_{R=\frac{n}{2\ell}}^{\frac{n}{\ell}}\binom{m+R}{R}T(n-R\ell,m,W,\ell)\right)\leq\left(\sum_{R=\frac{n}{2\ell}}^{\frac{n}{\ell}}\binom{m+R}{R}2^{n-R\ell}\right)$$
$$\leq\binom{m+\frac{n}{2\ell}}{\frac{n}{2\ell}}2^{n/2}\leq\binom{2m}{\frac{n}{2\ell}}2^{n/2}\leq 2^{\frac{n\log(4c\ell)}{\ell}}\cdot 2^{n/2}\leq 2^{(1-\mu(c))n},$$

where we set $\ell=\alpha\log c$ for sufficiently large constant $\alpha>0$ and the last inequality follows from $\binom{n}{\beta n}\leq\mathrm{poly}(n)\cdot 2^{2\beta n\log(1/\beta)}$ for $\beta\leq 1/2$.

We move on to the analysis of the first summation.

$$\left(\sum_{R=0}^{\frac{n}{2\ell}-1}\binom{m+R}{R}T(n-R\ell,m,W,\ell)\right)\leq\left(\sum_{R=0}^{\frac{n}{2\ell}-1}\binom{m+R}{R}2^{\left(1-\frac{1}{16\ell(\frac{cn}{n-R\ell})}\right)(n-R\ell)}\right)$$
$$\leq\left(\sum_{R=0}^{m+\frac{n}{2\ell}}\binom{m+\frac{n}{2\ell}}{R}2^{\left(1-\frac{1}{32\ell c}\right)(n-R\ell)}\right)=\left(2^{\left(1-\frac{1}{32\ell c}\right)n}\left(1+2^{-\left(1-\frac{1}{32\ell c}\right)\ell}\right)^{m+\frac{n}{2\ell}}\right)$$
$$\leq\left(2^{\left(1-\frac{1}{32\ell c}\right)n}\left(e^{2^{-\left(1-\frac{1}{32\ell c}\right)\ell}}\right)^{m+\frac{n}{2\ell}}\right)\leq\left(2^{\left(1-\frac{1}{32\ell c}\right)n+\frac{2m}{2^{\left(1-\frac{1}{32\ell c}\right)\ell}}}\right).$$

Since we set $\ell=\alpha\log c$ for sufficiently large constant $\alpha>0$, the exponent is

$$\left(1-\frac{1}{32\ell c}\right)n+\frac{2m}{2^{\left(1-\frac{1}{32\ell c}\right)\ell}}=\left(1-\frac{1}{32\alpha c\log c}\right)n+\frac{2cn}{2^{\alpha\log c-\frac{1}{32c}}}\leq\left(1-\frac{1}{64\alpha c\log c}\right)n,$$

where the last inequality is by the choice of α and $c>4$. This completes the proof. ◀

If we use the modified algorithm for Max ℓ-SAT due to Lemma 12 in Line 02 in Fig. 2 with $n'=\frac{n}{(16c\ell)^{2/3}}$, we have:

▶ **Theorem 15** (Restatement of Theorem 2). *Given an instance with n variables, $m=cn$ clauses and the maximum weight W, Max SAT can be solved deterministically in time $\mathrm{poly}(m,W)\cdot 2^{(1-\mu(c))n}$ and exponential space, where $\mu(c)=\frac{1}{O((c\log c)^{2/3})}$.*

The proof of the above theorem is almost identical to that of Theorem 14 and we omit it.

6 Concluding Remarks

There are several open questions. First, can we show a (randomized) polynomial space algorithm that runs super-polynomially faster than 2^n for instances with $\omega(n^2)$ clauses? One possible way is to design a non-trivial polynomial space algorithm for instances of Max 2-SAT with arbitrary number of clauses. Second, can we modify our exponential space algorithms to handle instances with exponentially large weights? To do so, we might have to show a non-trivial algorithm for instances of Max 2-SAT with arbitrary number of clauses and exponentially large weights.

Finally, we remark that the recent work of the authors [29] shows a deterministic $2^{n-n^{1/O(\ell)}}$ time and exponential space algorithm for instances of Max SAT with $m=O(n^\ell)$ clauses.

Acknowledgements. We are grateful to Osamu Watanabe, who asked us whether [28] can be improved if we treat each constraint as a clause instead of a De Morgan formula. Without his question, this work would not exist. We would like to thank anonymous reviewers for their helpful comments on our paper.

References

1 Nikhil Bansal and Venkatesh Raman. Upper bounds for MaxSAT: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC)*, volume 1741 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 1999.

2 Daniel Binkele-Raible and Henning Fernau. A new upper bound for Max-2-SAT: A graph-theoretic approach. *J. Discrete Algorithms*, 8(4):388–401, 2010.

3 Ivan Bliznets and Alexander Golovnev. A new algorithm for parameterized MAX-SAT. In *Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 7535 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2012.

4 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC)*, pages 252–260, 2006.

5 Jianer Chen and Iyad A. Kanj. Improved exact algorithms for MAX-SAT. *Discrete Applied Mathematics*, 142(1-3):17–27, 2004.

6 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015.

7 Ruiwen Chen and Rahul Santhanam. Improved algorithms for sparse max-sat and max-k-csp. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2015. To appear.

8 Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 403–424. IOS Press, 2009.

9 Evgeny Dantsin and Alexander Wolpert. MAX-SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4121 of *Lecture Notes in Computer Science*, pages 266–276. Springer, 2006.

10 Serge Gaspers and Gregory B. Sorkin. A universally fastest algorithm for Max 2-SAT, Max 2-CSP, and everything in between. *J. Comput. Syst. Sci.*, 78(1):305–335, 2012.

11 Serge Gaspers and Gregory B. Sorkin. Separate, measure and conquer: Faster polynomial-space algorithms for Max 2-CSP and counting dominating sets. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 567–579, 2015.

12 Alexander Golovnev and Konstantin Kutzkov. New exact algorithms for the 2-constraint satisfaction problem. *Theor. Comput. Sci.*, 526:18–27, 2014.

13 Jens Gramm, Edward A. Hirsch, Rolf Niedermeier, and Peter Rossmanith. Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. *Discrete Applied Mathematics*, 130(2):139–155, 2003.

14 Jens Gramm and Rolf Niedermeier. Faster exact solutions for MAX2SAT. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC)*, volume 1767 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2000.

15 Gregory Gutin and Anders Yeo. Constraint satisfaction problems parameterized above or below tight bounds: A survey. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 257–286. Springer, 2012.

16 Edward A. Hirsch. A new algorithm for MAX-2-SAT. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1770 of *Lecture Notes in Computer Science*, pages 65–73. Springer, 2000.

17 Edward A. Hirsch. Worst-case study of local search for MAX-k-SAT. *Discrete Applied Mathematics*, 130(2):173–184, 2003.

18 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 479–488, 2013.

19 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. On the parameterized complexity of exact satisfiability problems. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 3618 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2005.

20 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. A bound on the pathwidth of sparse graphs with applications to exact algorithms. *SIAM J. Discrete Math.*, 23(1):407–427, 2009.

21 Mikko Koivisto. Optimal 2-constraint satisfaction via sum-product algorithms. *Inf. Process. Lett.*, 98(1):24–28, 2006.

22 Arist Kojevnikov and Alexander S. Kulikov. A new approach to proving upper bounds for MAX-2-SAT. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–17, 2006.

23 Alexander S. Kulikov. Automated generation of simplification rules for SAT and MAXSAT. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *Lecture Notes in Computer Science*, pages 430–436. Springer, 2005.

24 Alexander S. Kulikov and Konstantin Kutzkov. New bounds for MAX-SAT by clause learning. In *Proceedings of the 8th International Computer Science Symposium in Russia (CSR)*, volume 4649 of *Lecture Notes in Computer Science*, pages 194–204. Springer, 2007.

25 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

26 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSAT and MaxCUT. *J. Algorithms*, 31(2):335–354, 1999.

27 Rolf Niedermeier and Peter Rossmanith. New upper bounds for maximum satisfiability. *J. Algorithms*, 36(1):63–88, 2000.

28 Takayuki Sakai, Kazuhisa Seto, and Suguru Tamaki. Solving sparse instances of max SAT via width reduction and greedy restriction. *Theory Comput. Syst.*, 57(2):426–443, 2015.

29 Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. A satisfiability algorithm for depth-2 circuits with a symmetric gate at the top and AND gates at the bottom. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-136, 2015.

30 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 183–192, 2010.

31 Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.

32 Alex D. Scott and Gregory B. Sorkin. Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances. In *Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and the 7th International Workshop on Randomization and Computation (RANDOM)*, volume 2764 of *Lecture Notes in Computer Science*, pages 382–395. Springer, 2003.

33 Alexander D. Scott and Gregory B. Sorkin. Linear-programming design and analysis of fast algorithms for Max 2-CSP. *Discrete Optimization*, 4(3-4):260–287, 2007.

34 Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2):245–274, 2013.

35 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.