Solving Sparse Instances of Max SAT via Width Reduction and Greedy Restriction

Takayuki Sakai[1], Kazuhisa Seto[2,⋆], and Suguru Tamaki[1,⋆⋆]

[1] Kyoto University, Yoshida Honmachi, Sakyo-ku, Kyoto 606-8501, Japan
{tsakai,tamak}@kuis.kyoto-u.ac.jp

[2] Seikei University, Musashino-shi, Tokyo 180-8633, Japan
seto@st.seikei.ac.jp

Abstract. We present a moderately exponential time polynomial space algorithm for sparse instances of Max SAT. Our algorithms run in time of the form $O(2^{(1-\mu(c))n})$ for instances with n variables and cn clauses. Our deterministic and randomized algorithm achieve $\mu(c) = \Omega(\frac{1}{c^2 \log^2 c})$ and $\mu(c) = \Omega(\frac{1}{c \log^3 c})$ respectively. Previously, an exponential space deterministic algorithm with $\mu(c) = \Omega(\frac{1}{c \log c})$ was shown by Dantsin and Wolpert [SAT 2006] and a polynomial space deterministic algorithm with $\mu(c) = \Omega(\frac{1}{2^{O(c)}})$ was shown by Kulikov and Kutzkov [CSR 2007].

Our algorithms have three new features. They can handle instances with (1) weights and (2) hard constraints, and also (3) they can solve counting versions of Max SAT. Our deterministic algorithm is based on the combination of two techniques, width reduction of Schuler and greedy restriction of Santhanam. Our randomized algorithm uses random restriction instead of greedy restriction.

Keywords: Exponential time algorithm, polynomial space, weight, hard constraint, counting.

1 Introduction

In the maximum satisfiability problem (Max SAT), the task is, given a set of clauses, to find an assignment that maximizes the number of satisfied clauses, where a clause is a disjunction of literals and a literal is a Boolean variable or its negation. Max SAT is one of the most fundamental NP-hard problems. In Max ℓ-SAT, we pose a restriction on input instances that each clause contains at most ℓ literals. Max ℓ-SAT is NP-hard even when $\ell = 2$.

Given an instance of Max SAT with n variables and $m = cn$ clauses, one can solve the problem in time $O(m2^n)$. The challenge is to reduce the running time to $O(\mathrm{poly}(m)2^{(1-\mu)n})$ for some absolute constant $\mu > 0$. This is

⋆ This work was supported in part by MEXT KAKENHI (24106001, 24106003); JSPS KAKENHI (26730007).

⋆⋆ This work was supported in part by MEXT KAKENHI (24106003); JSPS KAKENHI (25240002, 26330011).

a very difficult goal to achieve since the best running time upper bound is of the form $O(\mathrm{poly}(m)2^{(1-\frac{1}{O(\log c)})n})$ [4, 28] even for the satisfiability problem (SAT), where the task is to find an assignment that satisfies all the clauses. Therefore, it is natural to seek for an algorithm for Max SAT which runs in time $O(\mathrm{poly}(m)2^{(1-\mu(c))n})$ for some $\mu(c) > 0$. Previously, an exponential space deterministic algorithm with $\mu(c) = \Omega(\frac{1}{c \log c})$ was shown by Dantsin and Wolpert [8] and a polynomial space deterministic algorithm with $\mu(c) = \Omega(\frac{1}{2^{O(c)}})$ was shown by Kulikov and Kutzkov [24]. In this paper, we prove the following theorems.

Theorem 1. *For instances with n variables and $m = cn$ clauses, Max SAT can be solved deterministically in time $O(\mathrm{poly}(m)2^{(1-\mu(c))n})$ and polynomial space, where $\mu(c) = \Omega(\frac{1}{c^2 \log^2 c})$.*

Theorem 2. *For instances with n variables and $m = cn$ clauses, Max SAT can be solved probabilistically in expected time $O(\mathrm{poly}(m)2^{(1-\mu(c))n})$ and polynomial space, where $\mu(c) = \Omega(\frac{1}{c \log^3 c})$.*

Our algorithms have three new features which were not treated in [8, 24]. (1) Our algorithms can handle instances with weights. (2) Our algorithms can handle instances with hard constraints (instances of partial Max SAT), i.e., instances in which some clauses must be satisfied. (3) Our algorithms can count the number of optimal assignments. Furthermore, if we are allowed to use exponential space, our algorithms can be modified into the ones that can count the number of assignments achieving a given objective value. For more formal statements, see Theorems 4,5,7 and 8.

1.1 Related Work

The complexities of Max SAT and Max CSP (constraint satisfaction problem) has been studied with respect to several parameters [1–3, 6, 8–11, 14, 15, 19–26, 29, 30, 32]. 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.

We summarize the best upper bounds for each parametrization in Table 1. Here k is the objective value, i.e., the number of constraints that must be satisfied, l is the length of an instance, i.e., the sum of arities of constraints, m is the number of constraints, and n is the number of variables. We omit the factor in polynomial in k, l, m, n there. Note that for Max 2-SAT (and Max 2-CSP), the best upper bound is of the form $O(\mathrm{poly}(m)2^{(1-\mu)n})$ for some absolute constant $\mu > 0$. However, it is not known whether $O(\mathrm{poly}(m)2^{(1-\mu(\ell))n})$ time algorithms exist for Max ℓ-SAT when $\ell \geq 3$, where $\mu(\ell) > 0$ is a constant only depending on ℓ.

Table 1. A historical overview of upper bounds

Running time	Problem	Space	Reference
$O(2^{0.4414k})$	Max SAT	polynomial	[3]
$O(2^{0.1000l})$	Max 2-SAT	polynomial	[10]
$O(2^{0.1450l})$	Max SAT	polynomial	[1]
$O(2^{0.1583m})$	Max 2-SAT	polynomial	[9]
$O(2^{0.1901m})$	Max 2-CSP	polynomial	[9]
$O(2^{0.4057m})$	Max SAT	polynomial	[6]
$O(2^{0.7909n})$	Max 2-CSP	exponential	[21, 32]
$O(2^{(1-\alpha(\frac{m}{n}))n}), \alpha(c) = \frac{1}{O(c \log c)}$	Max SAT	exponential	[8]
$O(2^{(1-\beta(\frac{m}{n}))n}), \beta(c) = \frac{1}{O(2^{O(c)})}$	Max SAT	polynomial	[24]
$O(2^{(1-\gamma(\frac{m}{n}))n}), \gamma(c) = \frac{1}{O(c^2 \log^2 c)}$	Max SAT	polynomial	Theorem 1
$O(2^{(1-\delta(\frac{m}{n}))n}), \delta(c) = \frac{1}{O(c \log^3 c)}$	Max SAT	polynomial	Theorem 2

An alternative way to parametrize MAX SAT is to ask whether at least $\tilde{m}+k$ clauses can be satisfied, where $\tilde{m}$ is an expected number of satisfied constraints by a uniformly random assignment, see, e.g., [12].

1.2 Our Technique

Our deterministic algorithm is based on the combination of two techniques, width reduction of Schuler [4, 28] and greedy restriction of Santhanam [7, 27]. Our first observation is that we can solve Max ℓ-SAT in time $O(\text{poly}(m)2^{(1-\Omega(\frac{1}{\ell^2 c^2}))n})$ by slightly modifying Santhanam's greedy restriction algorithm for De Morgan formula SAT [7, 27]. To apply the Max ℓ-SAT algorithm for general Max SAT, we adopt the technique of Schuler's width reduction [4, 28]. Briefly, width reduction is, given an instance of SAT, to produce a collection of instances of ℓ-SAT such that the original instance is satisfiable if and only if at least one of the produced instances is satisfiable. We will see that width reduction can be also applied to Max SAT instances. Once we recognize the approach of combining greedy restriction and width reduction works, the analysis of running time basically follows from the existing analysis.

Our randomized algorithm uses a simple random restriction algorithm for Max ℓ-SAT instead of the greedy restriction algorithm.

1.3 Paper Organization

In section 2, we introduce definitions and notation needed in the paper. In section 3, we present a greedy restriction algorithm for Max formula SAT and its running time analysis. The algorithm is used to solve Max ℓ-SAT. In section 4, we show a deterministic algorithm for Max SAT and its analysis. In section 5, we present an exponential space deterministic algorithm for a counting version of Max SAT and its analysis. In section 6, we show a randomized algorithm for Max ℓ-SAT and its analysis.

2 Preliminaries

We denote by $\mathbb{Z}$ the set of integers. We define $-\infty$ as $-\infty + z = z + -\infty = -\infty$ and $-\infty < z$ for any $z \in \mathbb{Z}$. ∞ is defined analogously. Let $V = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. 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 of V. Note that a 0-constraint is either '0' or '1' (a constant function). An *instance* Φ of Max CSPs 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 a k_i-constraint and $w_i : \{0,1\} \to \{-\infty\} \cup \mathbb{Z}$. For a weight function w, we denote by $\widetilde{w}$ a weight function such that $\widetilde{w}(1) = w(1), \widetilde{w}(0) = -\infty$. Note that a constraint with a weight of the form $w(0) = -\infty$ must be satisfied, i.e., it is a *hard* constraint. The width of Φ is $\max_i k_i$. We use the notation as $\mathrm{Val}(\Phi, a) := \sum_{i=1}^m w_i(\phi_i(a))$ and $\mathrm{Opt}(\Phi) := \max_{a \in \{0,1\}^n} \mathrm{Val}(\Phi, a)$. For an integer K, we define

$$\#\mathrm{Val}_{\geq K}(\Phi) := |\{a \in \{0,1\}^n \mid \mathrm{Val}(\Phi, a) \geq K\}|,$$
$$\#\mathrm{Opt}(\Phi) := |\{a \in \{0,1\}^n \mid \mathrm{Val}(\Phi, a) = \mathrm{Opt}(\Phi)\}|.$$

We are interested in subclasses of Max CSPs defined by restricting the type of constraints. We denote by $y_1, \ldots, y_\ell$ arbitrary literals. In *Max SAT*, each constraint must be a *clause*, i.e., a disjunction of literals of the form $y_1 \vee \cdots \vee y_\ell$. We allow 0-constraints to appear in instances of Max SAT.

3 A Greedy Restriction Algorithm for Max formula SAT

In this section, we present an algorithm for Max CSPs in which each constraint is given as a De Morgan formula. The algorithm immediately yields an algorithm for Max ℓ-SAT.

3.1 De Morgan Formula and Its Simplification

A *De Morgan formula* is a rooted binary tree in which each leaf is labeled by a literal from the set $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ or a constant from $\{0,1\}$ and each internal node is labeled by $\wedge$ ("and") or $\vee$ ("or"). Given a De Morgan formula ϕ, a *subformula* of ϕ is a De Morgan formula which is a subtree in ϕ. Every De Morgan formula computes in a natural way a Boolean function from $\{0,1\}^n$ to $\{0,1\}$. The *size* of a De Morgan formula ϕ is defined to be the number of leaves in it, and it is denoted by $L(\phi)$. We denote by $\mathrm{var}(\phi)$ the set of variables which appear as literals in ϕ. The *frequency* of a variable x in ϕ is defined to be the number of leaves labeled by x or $\overline{x}$, and it is denoted by $\mathrm{freq}_\phi(x)$.

For any formula ϕ, 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 formula obtained from ϕ by assigning to each x_{i_j}, $\overline{x}_{i_j}$ the value a_j, $\overline{a}_j$ and applying the following procedure **Simplify**. We can similarly define $\phi[l_{i_1} = a_1, \ldots, l_{i_k} = a_k]$ for any set of literals $\{l_{i_1}, \ldots, l_{i_k}\}$. The procedure **Simplify** reduces the size of a formula by applying rules to eliminate constants and redundant literals. These are the same simplification rules used by [13, 27].

Simplify(ϕ: formula)
Repeat the following until there is no decrease in size of ϕ.

(a) If $0 \wedge \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by 0.
(b) If $0 \vee \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by ψ.
(c) If $1 \wedge \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by ψ.
(d) If $1 \vee \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by 1.
(e) If $y \vee \psi$ occurs as a subformula, where ψ is a formula and y is a literal, then replace all occurrences of y in ψ by 0 and all occurrence of $\overline{y}$ by 1.
(f) If $y \wedge \psi$ occurs as a subformula, where ψ is a formula and y is a literal, then replace all occurrences of y in ψ by 1 and all occurrence of $\overline{y}$ by 0.

Fig. 1. Simplification procedure

It is easy to see that **Simplify** runs in time polynomial in the size of ϕ and the resulting formula computes the same function as ϕ.

3.2 An Algorithm for Max Formula SAT

Max formula SAT is a subclass of Max CSPs where each constraint is given as a De Morgan formula, i.e., an instance is of the form $\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ where ϕ_i is a De Morgan formula and w_i is a weight function. We use the notation as $\mathrm{var}(\Phi) := \cup_{i=1}^{m} \mathrm{var}(\phi_i)$, $\widetilde{L}(\Phi) := \sum_{i:L(\phi_i)\geq 2} L(\phi_i)$ and $\widetilde{\mathrm{freq}}_{\Phi}(x) := \sum_{i:L(\phi_i)\geq 2} \mathrm{freq}_{\phi_i}(x)$. For any instance Φ, any set of variables $\{x_{i_1}, \ldots, x_{i_k}\}$ and any constants $a_1, \ldots, a_k \in \{0, 1\}$, we define as $\Phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k] := \{(\phi'_1, w_1), \ldots, (\phi'_m, w_m)\}$ where $\phi'_i = \phi_i[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]$.

We need the following lemma.

Lemma 1. *Given an instance Φ of Max formula SAT with m constraints,* $\mathrm{Opt}(\Phi)$ *and* $\#\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m)2^{\widetilde{L}(\Phi)}$.

Proof. We first show the following claim.

Claim. Given an instance Φ of Max formula SAT with m constraints and $\widetilde{L}(\Phi) = 0$, $\mathrm{Opt}(\Phi)$ and $\#\mathrm{Opt}(\Phi)$ can be computed in time $\mathrm{poly}(m)$.

Proof. $\widetilde{L}(\Phi) = 0$ means that 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

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

If $S_i(0) = S_i(1)$, then let $x_i = *$ (don't care), if $S_i(0) > S_i(1)$, then let $x_i = 0$, otherwise let $x_i = 1$. This assignment achieves $\mathrm{Opt}(\Phi)$ where we assign arbitrary values to don't care variables. If the number of don't care variables is d, then $\#\mathrm{Opt}(\Phi) = 2^d$. □

Let $V' = \{x_{i_1}, x_{i_2} \ldots\}$ be the set of Boolean variables that appear in constraints of size at least two. Note that $|V'| \le \widetilde{L}(\Phi)$. For each assignment $a_1, a_2, \ldots \in \{0,1\}$, we can compute

$$\mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots]), \#\mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots])$$

in time $\mathrm{poly}(m)$ since $\widetilde{L}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots]) = 0$. Let

$$K = \max_{a_1, a_2, \ldots \in \{0,1\}} \mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots]),$$
$$N = \sum \#\mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots]),$$

where the summation is over

$$\{a_1, a_2, \ldots \in \{0,1\} \mid \mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots]) = K\}.$$

We have $K = \mathrm{Opt}(\Phi)$ and $N = \#\mathrm{Opt}(\Phi)$ and can compute them in time $\mathrm{poly}(m) 2^{\widetilde{L}(\Phi)}$. □

Now we are ready to present our algorithm for Max formula SAT as shown in Figure 2.

This is a slight modification of Santhanam's greedy restriction algorithm for De Morgan formula SAT [27]. The difference is (1) we use a different definition of size and frequency, and (2) in the case of $\widetilde{L}(\Phi) < \frac{n}{2}$, we need an additional lemma. We have the following theorem.

Theorem 3. *Given an instance Φ of Max formula SAT with n variables and $\widetilde{L}(\Phi) = m = cn$,* **EvalFormula** *computes* $\mathrm{Opt}(\Phi)$ *and* $\#\mathrm{Opt}(\Phi)$ *in time* $\mathrm{poly}(m) \times 2^{(1 - \frac{1}{32c^2})n}$.

Since a disjunction of ℓ literals can be regarded as a De Morgan formula of size ℓ, the above theorem immediately implies the following.

Corollary 1. *Given an instance Φ of Max ℓ-SAT with n variables and $m = cn$ constraints,* **EvalFormula** *computes* $\mathrm{Opt}(\Phi)$ *and* $\#\mathrm{Opt}(\Phi)$ *in time* $\mathrm{poly}(m) \times 2^{(1 - \frac{1}{32\ell^2 c^2})n}$.

```
EvalFormula($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: instance, $n$: integer)
01: if $\widetilde{L}(\Phi) = cn < n/2$,
02:     compute $\mathrm{Opt}(\Phi), \#\mathrm{Opt}(\Phi)$ by Lemma 1 and return $(\mathrm{Opt}(\Phi), \#\mathrm{Opt}(\Phi))$.
03: else
04:     $x = \arg\max_{x \in \mathrm{var}(\Phi)} \widetilde{\mathrm{freq}}_\Phi(x)$.
05:     $(K_0, N_0) \leftarrow$ EvalFormula($\Phi[x = 0], n - 1$).
06:     $(K_1, N_1) \leftarrow$ EvalFormula($\Phi[x = 1], n - 1$).
07:     if $K_0 = K_1$.
08:         return $(K_0, N_0 + N_1)$
09:     else
10:         $i = \arg\max_{j \in \{0,1\}} \{K_j\}$ and return $(K_i, N_i)$.
```

Fig. 2. Max formula SAT algorithm

Proof (of Theorem 3). The proof of Theorem 3 is almost identical to the proof of Theorem 5.2 in Chen et al. [7], which gives a running time analysis of Santhanam's algorithm for De Morgan formula SAT based on a super-martingale approach.

For an instance Φ of Max formula SAT on n variables, define $\Phi_0 = \Phi$. For $1 \le i \le n$, we define Φ_i to be an instance of Max formula SAT obtained from Φ_{i-1} by uniformly at random assigning the most frequent variable of Φ_{i-1}. We need the following lemma.

Lemma 2 (Shrinkage Lemma with Respect to $\widetilde{L}(\cdot)$). *Let Φ be an instance of Max formula SAT on n variables. For any $k \ge 4$, we have*

$$\Pr\left[\widetilde{L}(\Phi_{n-k}) \ge 2 \cdot \widetilde{L}(\Phi) \cdot \left(\frac{k}{n}\right)^{3/2}\right] < 2^{-k}.$$

Let Φ be an instance of Max formula SAT with n variables and $\widetilde{L}(\Phi) = m = cn$ for some constant $c > 0$. Let $p = \left(\frac{1}{4c}\right)^2$ and $k = pn$. We construct the following computation tree.

The root is labeled Φ. At first, we pick up the most frequent variable x with respect to $\widetilde{L}(\Phi)$ and set the variable first to 0 then to 1. By simplification, Φ branches to the two Max formula SAT instances such as $\Phi[x = 0]$ and $\Phi[x = 1]$. The children of the current node are labeled by these instances. Repeating this procedure until we can make a full binary tree of depth exactly $n - k$. Note that this computation tree can be made in time $\mathrm{poly}(m)2^{n-k}$. Using Lemma 2, for all but at most 2^{-k} fraction of the leaves have the size $\widetilde{L}(\Phi_{n-k}) < 2\widetilde{L}(\Phi)\left(\frac{k}{n}\right)^{3/2} = 2cnp^{3/2} = 2cp^{1/2} \cdot pn = \frac{1}{2}pn = \frac{k}{2}$. For instances with $\widetilde{L}(\Phi_{n-k}) < k/2$, we can compute $\mathrm{Opt}(\Phi_{n-k}), \#\mathrm{Opt}(\Phi_{n-k})$ in time $\mathrm{poly}(m)2^{k/2}$ by Lemma 1. For instances with $\widetilde{L}(\Phi_{n-k}) \ge k/2$, we can compute $\mathrm{Opt}(\Phi_{n-k}), \#\mathrm{Opt}(\Phi_{n-k})$ in time $\mathrm{poly}(m)2^k$ by brute force search. The overall running time of **EvalFormula** is bounded by $\mathrm{poly}(m)2^{n-k}\{2^{k/2}(1 - 2^{-k}) + 2^k \cdot 2^{-k}\} = \mathrm{poly}(m)2^{(1-\frac{1}{32c^2})n}$. □

4 A Deterministic Algorithm for Max SAT

In this section, we present our deterministic algorithm for sparse instances of Max SAT based on the combination of the algorithm for Max formula SAT and width reduction.

Before presenting our algorithm for Max SAT, let us recall Schuler's width reduction technique for SAT [28]. Let Φ be an instance of SAT, i.e., $\Phi = \{\phi_1, \phi_2, \ldots, \phi_m\}$ where each ϕ_i is a disjunction of literals. Assume for some i and $\ell' > \ell$, $\phi_i = (l_1 \vee \cdots \vee l_\ell \vee l_{\ell+1} \vee \cdots \vee l_{\ell'})$ and define $\Phi_L := \{\Phi \backslash \{\phi_i\}\} \cup \{(l_1 \vee \cdots \vee l_\ell)\}$ and $\Phi_R := \Phi[l_1 = \cdots l_\ell = 0]$. Then we can observe that Φ is satisfiable if and only if at least one of Φ_L, Φ_R is satisfiable. Note that compared with Φ, the number of clauses with width more than ℓ decreases in Φ_L, and the number of variables decreases in Φ_R. The step of producing two instances Φ_L, Φ_R from Φ is the one step of width reduction and by applying the reduction recursively, we obtain a set of instances where the width of each constraint is at most ℓ. After obtaining a set of ℓ-SAT instances, we can apply an algorithm for ℓ-SAT which runs in time $\mathrm{poly}(m)2^{(1-\mu(\ell))n}$ for some $\mu(\ell) > 0$ and decide the satisfiability of Φ. Schuler makes use of this observation to design an algorithm for SAT.

We see that width reduction is possible for Max SAT instances. Our algorithm for Max SAT is shown in Figure 3.

```
MaxSAT($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: instance, $\ell, n$: integer)
01: if $\forall \phi_i \in \Phi, |\mathrm{var}(\phi_i)| \le \ell$,
02:     return EvalFormula($\Phi, n$).
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:     $(K_L, N_L) \leftarrow$ MaxSAT($\Phi_L, \ell, n$).
07:     $\Phi_R \leftarrow \Phi[l_1 = \cdots = l_\ell = 0]$.
08:     $(K_R, N_R) \leftarrow$ MaxSAT($\Phi_R, \ell, n - \ell$).
09:     if $K_L = K_R$,
10:         return $(K_L, N_L + N_R)$
11:     else
12:         $i = \arg\max_{j \in \{L,R\}} \{K_j\}$ and return $(K_i, N_i)$.
```

Fig. 3. Max SAT algorithm

The correctness of **MaxSAT** is guaranteed by the following claim.

Claim. In **MaxSAT**, if **else** condition of line 3 holds, then $\mathrm{Opt}(\Phi) = \max\{K_L, K_R\}$.

Proof. First note that $\mathrm{Opt}(\Phi) \ge \max\{K_L, K_R\}$. Let a be an assignment that maximizes the total weight of satisfied constraints, i.e., $a = \arg\max_a \mathrm{Val}(\Phi, a)$. If $l_1 = \cdots = l_\ell = 0$ according to a, then $\mathrm{Opt}(\Phi) = K_R$. Otherwise, a satisfies $(l_1 \vee \cdots \vee l_\ell)$ and $\mathrm{Opt}(\Phi) = K_L$ holds. □

Now we describe our main theorem and its proof.

Theorem 4. *Given an instance Φ of Max SAT with n variables and $m = cn$ clauses,* **MaxSAT** *with appropriately chosen ℓ computes* $\mathrm{Opt}(\Phi)$ *and* $\#\mathrm{Opt}(\Phi)$ *in time $O(\mathrm{poly}(m)2^{(1-\mu(c))n})$, where $\mu(c)$ is $\Omega\left(\frac{1}{c^2 \log^2 c}\right)$ and $c > 4$.*

Proof. The overall structure of the proof is similar to the analysis of width reduction for SAT by 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 by an input instance Φ and for each node labeled with Ψ, its left (right) child is labeled with Ψ_L (Ψ_R, resp.). If Ψ is an instance of Max ℓ-SAT, i.e., every constraint ψ_i of Ψ satisfies $|\mathrm{var}(\psi_i)| \leq \ell$, 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 constraints 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 $T(n, m, \ell)$ denote the running time of **EvalFormula** on instances of Max ℓ-SAT with n variables and $m = cn$ clauses, then we can upper bound the running time of **MaxSAT** as:

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

We first upper bound the second summation above.

$$\mathrm{poly}(m)\left(\sum_{R=\frac{n}{2\ell}}^{\frac{n}{\ell}} \binom{m+R}{R} T(n - R\ell, m, \ell)\right)$$
$$\leq \mathrm{poly}(m)\left(\sum_{R=\frac{n}{2\ell}}^{\frac{n}{\ell}} \binom{m+R}{R} 2^{n-R\ell}\right)$$
$$\leq \mathrm{poly}(m)\binom{m+\frac{n}{2\ell}}{\frac{n}{2\ell}} 2^{n/2} \leq O(\mathrm{poly}(m)2^{(1-\mu(c))n}),$$

where the last two inequalities hold when $\ell = \alpha \log c$ for sufficiently large constant $\alpha > 0$.

We move on the analysis of the following summation.

$$\begin{aligned}
&\mathrm{poly}(m)\left(\sum_{R=0}^{\frac{n}{2\ell}-1}\binom{m+R}{R}T(n-R\ell,m,\ell)\right)\\
&=\mathrm{poly}(m)\left(\sum_{R=0}^{\frac{n}{2\ell}-1}\binom{m+R}{R}2^{\left(1-\frac{1}{32\ell^2(\frac{cn}{n-R\ell})^2}\right)(n-R\ell)}\right)\\
&\le\mathrm{poly}(m)\left(\sum_{R=0}^{\frac{n}{2\ell}-1}\binom{m+R}{R}2^{\left(1-\frac{1}{32\ell^2(2c)^2}\right)(n-R\ell)}\right)\\
&\le\mathrm{poly}(m)\left(\sum_{R=0}^{m+\frac{n}{2\ell}}\binom{m+\frac{n}{2\ell}}{R}2^{\left(1-\frac{1}{32\ell^2(2c)^2}\right)(n-R\ell)}\right)
\end{aligned}$$

$$\begin{aligned}
&=\mathrm{poly}(m)\left(2^{\left(1-\frac{1}{32\ell^2(2c)^2}\right)n}\sum_{R=0}^{m+\frac{n}{2\ell}}\binom{m+\frac{n}{2\ell}}{R}2^{-\left(1-\frac{1}{32\ell^2(2c)^2}\right)R\ell}\right)\\
&=\mathrm{poly}(m)\left(2^{\left(1-\frac{1}{32\ell^2(2c)^2}\right)n}\left(1+2^{-\left(1-\frac{1}{32\ell^2(2c)^2}\right)\ell}\right)^{m+\frac{n}{2\ell}}\right)\\
&\le\mathrm{poly}(m)\left(2^{\left(1-\frac{1}{32\ell^2(2c)^2}\right)n}\left(e^{2^{-\left(1-\frac{1}{32\ell^2(2c)^2}\right)\ell}}\right)^{m+\frac{n}{2\ell}}\right)\\
&\le\mathrm{poly}(m)\left(2^{\left(1-\frac{1}{32\ell^2(2c)^2}\right)n+\frac{2m}{2^{\left(1-\frac{1}{32\ell^2(2c)^2}\right)\ell}}}\right).
\end{aligned}$$

Now we set $\ell=\beta\log c$ for sufficiently large constant $\beta>0$, then the exponent is

$$\begin{aligned}
&\left(1-\frac{1}{32\ell^2(2c)^2}\right)n+\frac{2m}{2^{\left(1-\frac{1}{32\ell^2(2c)^2}\right)\ell}}\\
&=\left(1-\frac{1}{128\beta^2c^2\log^2 c}\right)n+\frac{2cn}{2^{\beta\log c-\frac{1}{128\beta c^2\log c}}}\\
&=\left(1-\frac{1}{128\beta^2c^2\log^2 c}+\frac{2^{\frac{1}{128\beta c^2\log c}+1}}{c^{\beta-1}}\right)n\\
&\le\left(1-\frac{1}{256\beta^2c^2\log^2 c}\right)n,
\end{aligned}$$

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

□

Remark 1. Since we are interested in the running time of **MaxSAT** when c goes to infinity, we do not give a precise upper bound for small c. However, we can show that the running time is of the form $\mathrm{poly}(m)2^{(1-\mu(c))n}$ for some $\mu(c) > 0$ even when c is small.

5 A Deterministic Algorithm for Counting Problems

In this section, we show how to compute $\#\mathrm{Val}_{\geq K}(\Phi)$ in moderately exponential time and exponential space.

Theorem 5. *Given an instance Φ of Max SAT with n variables and $m = cn$ clauses,* **MaxSAT2** *with appropriately chosen ℓ computes $\#\mathrm{Val}_{\geq K}(\Phi)$ in time $O(\mathrm{poly}(m)2^{(1-\mu(c))n})$ and exponential space, where $\mu(c)$ is $\Omega\left(\frac{1}{c^2 \log^2 c}\right)$.*

We need the following lemma.

Lemma 3. *Given an instance Φ of Max formula SAT with n variables, m constraints and $\widetilde{L}(\Phi) \leq n$, $\#\mathrm{Val}_{\geq K}(\Phi)$ can be computed in time $\mathrm{poly}(m)2^{\widetilde{L}(\Phi)} \cdot 2^{(n-\widetilde{L}(\Phi))/2}$ and exponential space.*

Proof. We first show the following claim.

Claim. Given an instance Φ of Max formula SAT with n variables and $\widetilde{L}(\Phi) = 0$, $\#\mathrm{Val}_{\geq K}(\Phi)$ can be computed in time $\mathrm{poly}(m)2^{n/2}$ and exponential space.

Proof. $\widetilde{L}(\Phi) = 0$ means that each ϕ_i is either '0', '1', 'x_j' or '$\overline{x}_j$' (for some j). Without loss of generality, we can assume that ϕ_i is either 'x_j'or '$\overline{x}_j$' by removing each ϕ_i of the form '0' or '1' from Φ and replacing K by $K - w_i(\phi_i)$.

We use a slight modification of the algorithm for the subset-sum problem due to [16]. For brevity, let us assume that n is even and let $V_1 := \{x_1, \ldots, x_{n/2}\}, V_2 := \{x_{n/2+1}, \ldots, x_n\}$. Then Φ can be represented as a disjoint union of Φ_1, Φ_2 where Φ_i only depends on V_i. Define $W_i := \{\mathrm{Val}(\Phi_i, a) \mid a \in \{0,1\}^{n/2}\}$. We do not think of W_i as a multiset and the size of W_i can be less than $2^{n/2}$.

We are to construct arrays $\{(s^i_j, t^i_j)\}_{j \in W_i}$ for $i = 1, 2$ such that (1) s^i_j is the jth largest value in W_i, (2) t^1_j is the number of assignments $a \in \{0,1\}^{n/2}$ such that $\mathrm{Val}(\Phi_1, a) = s^1_j$, and (3) t^2_j is the number of assignments $a \in \{0,1\}^{n/2}$ such that $\mathrm{Val}(\Phi_2, a) \geq s^2_j$.

First, we assume that we have constructed the arrays and show how to compute $\mathrm{Val}_{\geq K}(\Phi)$. Let s be an integer and set $s = 0$. For each s^1_j, we can find the smallest $s^2_{j'}$ such that $s^1_j + s^2_{j'} \geq K$ in time $\mathrm{poly}(n)$ using binary search. If such $s^2_{j'}$ can be found, then update s as $s + t^1_j \cdot t^2_{j'}$. Repeat this for every $s^1_j \in W_1$. Then, we have $s = \#\mathrm{Val}_{\geq K}(\Phi)$ in the end.

To construct $\{(s^i_j, t^i_j)\}_{j \in W_i}$, we first enumerate $\mathrm{Val}(\Phi_i, a)$ for every $a \in \{0,1\}^{n/2}$ and sort them in the decreasing order in time $\mathrm{poly}(n)2^{n/2}$. Then, we can compute $\{(s^i_j, t^i_j)\}_{j \in W_i}$ from $j = 1$ inductively in time $\mathrm{poly}(n)2^{n/2}$. □

Let $V' = \{x_{i_1}, x_{i_2} \ldots\}$ be the set of Boolean variables that appear in constraints of size at least two. Note that $|V'| \leq \widetilde{L}(\Phi)$. For each assignment $a_1, a_2, \ldots \in \{0,1\}$, we can compute

$$\#\mathrm{Val}_{\geq K}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots])$$

in time $\mathrm{poly}(m)2^{n/2}$ since $\widetilde{L}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots]) = 0$. Let

$$N = \sum_{a_1, a_2, \ldots \in \{0,1\}} \#\mathrm{Val}_{\geq K}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots]).$$

We have $N = \#\mathrm{Val}_{\geq K}(\Phi)$ and can compute it in time $\mathrm{poly}(m)2^{\widetilde{L}(\Phi)} \cdot 2^{(n-\widetilde{L}(\Phi))/2}$. $\square$

Now we are ready to present our algorithm for $\#\mathrm{Val}_{\geq K}(\Phi)$ as shown in Figures 4,5. The running time analysis is almost the same as that for Theorem 4 and we omit it.

```
EvalFormula2($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: instance, $n$: integer)
01: if $\widetilde{L}(\Phi) = cn < n/2$,
02:     compute $\#\mathrm{Val}_{\geq K}(\Phi)$ by Lemma 3 and return $\#\mathrm{Val}_{\geq K}(\Phi)$.
03: else
04:     $x = \arg\max_{x \in \mathrm{var}(\Phi)} \widetilde{\mathrm{freq}}_{\Phi}(x)$.
05:     $N_0 \leftarrow$ EvalFormula2($\Phi[x = 0], n - 1$).
06:     $N_1 \leftarrow$ EvalFormula2($\Phi[x = 1], n - 1$).
07:     return $N_0 + N_1$.
```

Fig. 4. Max formula SAT algorithm

```
MaxSAT2($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: instance, $\ell, n$: integer)
01: if $\forall \phi_i \in \Phi, |\mathrm{var}(\phi_i)| \leq \ell$,
02:     return EvalFormula2($\Phi, n$).
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:     $N_L \leftarrow$ MaxSAT2($\Phi_L, \ell, n$).
07:     $\Phi_R \leftarrow \Phi[l_1 = \cdots = l_\ell = 0]$.
08:     $N_R \leftarrow$ MaxSAT2($\Phi_R, \ell, n - \ell$).
09:     return $N_L + N_R$.
```

Fig. 5. Max SAT algorithm

6 A Randomized Algorithm for Max ℓ-SAT

In this section, we present our randomized algorithm for Max ℓ-SAT.

The basic idea of our algorithm is as follows: If $\widetilde{L}(\Phi) < n/2$, then we use Lemma 1. Otherwise, we choose a subset U of the set of variables $V = \{x_1, x_2, \ldots, x_n\}$ uniformly at random with $|U| = (1-p)n$, where p is chosen appropriately according to $\widetilde{L}(\Phi)$. If the number of constraints which depend on at least two variables in $V \setminus U$ is 'small,' i.e., $|\{\phi_i \in \Phi \mid |\mathrm{var}(\phi_i) \setminus U| \geq 2\}| < pn/(2\ell)$ holds, then we say U is *good*. Assume U is good and $U = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{|U|}}\}$. Then, for every $a \in \{0,1\}^{|U|}$, $\widetilde{L}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots]) < \ell \cdot pn/(2\ell) = pn/2$ and $\mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots])$ can be computed in time $\mathrm{poly}(m)2^{pn/2}$. Therefore, we can compute $\mathrm{Opt}(\Phi)$ in time $\mathrm{poly}(m)2^{pn/2} \cdot 2^{(1-p)n} = \mathrm{poly}(m)2^{(1-p/2)n}$.

Our randomized algorithm for Max ℓ-SAT is shown in Figure 6.

```
**Max$\ell$SAT($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: instance, $n$: integer)**
01: if $\widetilde{L}(\Phi) = cn < n/2$,
02:     compute Opt($\Phi$) by Lemma 1 and return Opt($\Phi$).
03: else
04:     Pick $U \subseteq V, |U| = (1-p)n = \left(1 - \frac{1}{c\ell^3}\right) n$ uniformly at random.
05:     if $|\{\phi_i \in \Phi \mid |\mathrm{var}(\phi_i) \setminus U| \geq 2\}| \geq pn/(2\ell)$,
06:       return $\perp$.
07:     else /* we assume $U = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{|U|}}\}$ */
08:       for each $a \in \{0,1\}^{|U|}$,
09:         $\Phi_{U,a} \leftarrow \Phi[x_{i_1} = a, x_{i_2} = a_2, \ldots])$.
10:         compute Opt($\Phi_{U,a}$) by Lemma 1.
11:       return $\max_{a \in \{0,1\}^{|U|}}$ Opt($\Phi_{U,a}$).
```

Fig. 6. Max ℓ-SAT algorithm

Note that if **MaxℓSAT** does not return $\perp$, then it returns $\mathrm{Opt}(\Phi)$. We show that U is good with constant probability. Similar calculation can be found in, e.g., [5].

Lemma 4. *In* **MaxℓSAT***, if* ***else*** *condition of line 3 holds, then* **MaxℓSAT** *returns $\perp$ with probability at most* $1/2$.

Proof. Let X_i be a random variable such that $X_i = 1$ if $|\mathrm{var}(\phi_i) \setminus U| \geq 2$, otherwise $X_i = 0$.
Since $\mathbf{Pr}_U[X_i = 1] \leq \binom{\ell}{2} p^2$ by the union bound, we have

$$\mathop{\mathbf{E}}_{U}\left[\sum_{i \in [m]} X_i\right] \leq \binom{\ell}{2} p^2 m = \binom{\ell}{2} p^2 cn.$$

By Markov's inequality,

$$\Pr_U\left[\sum_{i\in[m]} X_i \geq \frac{pn}{2\ell}\right] \leq \frac{\binom{\ell}{2}p^2cn}{\frac{pn}{2\ell}} = c\ell^2(\ell-1)p.$$

Setting p yields the consequence of the lemma. □

By combining the preceding argument and the above lemma, we have the following theorem.

Theorem 6. *Given an instance Φ of Max ℓ-SAT with n variables and m=cn constraints, with the probability at least $\frac{1}{2}$,* **MaxℓSAT** *computes* $\mathrm{Opt}(\Phi)$ *in time* $\mathrm{poly}(m)2^{\left(1-\frac{1}{2c\ell^3}\right)n}$.

MaxℓSAT can be easily modified to compute $\#\mathrm{Opt}(\Phi)$ and $\#\mathrm{Val}_{\geq K}(\Phi)$ as our deterministic algorithms. If we use **MaxℓSAT** instead of **EvalFormula** in **MaxSAT** and **MaxSAT2**, we have the following theorems.

Theorem 7. *Given an instance Φ of Max SAT with n variables and $m = cn$ clauses,* $\mathrm{Opt}(\Phi)$ *and* $\#\mathrm{Opt}(\Phi)$ *can be computed in expected time* $O(\mathrm{poly}(m) \times 2^{(1-\mu(c))n})$*, where $\mu(c)$ is* $\Omega\left(\frac{1}{c\log^3 c}\right)$.

Theorem 8. *Given an instance Φ of Max SAT with n variables and $m = cn$ clauses,* $\#\mathrm{Val}_{\geq K}(\Phi)$ *can be computed in expected time* $O(\mathrm{poly}(m)2^{(1-\mu(c))n})$ *and exponential space, where $\mu(c)$ is* $\Omega\left(\frac{1}{c\log^3 c}\right)$.

7 Concluding Remarks

In this paper, we present moderately exponential time polynomial space algorithms for sparse instances of Max SAT. There are several possible directions for future work. First, since our algorithm is partly inspired by the recent development in the study of exact algorithms for the circuit satisfiability problem [7, 17, 18, 27, 31, 33], we may hope that it is possible to make use of such algorithms and their analysis to improve existing algorithms or design new algorithms for Max CSPs or other problems related to SAT. Second, our algorithm can count the number of assignments that achieve total weights greater than a given objective value K in exponential space, however, it seems not obvious to obtain the same result by polynomial space algorithms. Finally, to improve $\mu(c)$ to subpolynomial in $1/c$, say $\mu(c) = \Omega(1/\mathrm{poly}(\log c))$ is a challenging goal since it implies non-trivial algorithms for arbitrary size of Max 3-SAT instances.

References

1. Bansal, N., Raman, V.: Upper bounds for MaxSAT: Further improved. In: Aggarwal, A.K., Pandu Rangan, C. (eds.) ISAAC 1999. LNCS, vol. 1741, pp. 247–258. Springer, Heidelberg (1999)
2. Binkele-Raible, D., Fernau, H.: A new upper bound for Max-2-SAT: A graph-theoretic approach. J. Discrete Algorithms 8(4), 388–401 (2010)

3. Bliznets, I., Golovnev, A.: A new algorithm for parameterized MAX-SAT. In: Thilikos, D.M., Woeginger, G.J. (eds.) IPEC 2012. LNCS, vol. 7535, pp. 37–48. Springer, Heidelberg (2012)
4. Calabro, C., Impagliazzo, R., Paturi, R.: A duality between clause width and clause density for SAT. In: Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC), pp. 252–260 (2006)
5. Calabro, C., Impagliazzo, R., Paturi, R.: The complexity of satisfiability of small depth circuits. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 75–85. Springer, Heidelberg (2009)
6. Chen, J., Kanj, I.A.: Improved exact algorithms for MAX-SAT. Discrete Applied Mathematics 142(1-3), 17–27 (2004)
7. Chen, R., Kabanets, V., Kolokolova, A., Shaltiel, R., Zuckerman, D.: Mining circuit lower bound proofs for meta-algorithms. Electronic Colloquium on Computational Complexity (ECCC) TR13-057 (2013)
8. Dantsin, E., Wolpert, A.: MAX-SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 266–276. Springer, Heidelberg (2006)
9. Gaspers, S., Sorkin, G.B.: A universally fastest algorithm for Max 2-SAT, Max 2-CSP, and everything in between. J. Comput. Syst. Sci. 78(1), 305–335 (2012)
10. Gramm, J., Hirsch, E.A., Niedermeier, R., Rossmanith, P.: Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. Discrete Applied Mathematics 130(2), 139–155 (2003)
11. Gramm, J., Niedermeier, R.: Faster exact solutions for MAX2SAT. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) CIAC 2000. LNCS, vol. 1767, pp. 174–186. Springer, Heidelberg (2000)
12. Gutin, G., Yeo, A.: Constraint satisfaction problems parameterized above or below tight bounds: A survey. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) Fellows Festschrift 2012. LNCS, vol. 7370, pp. 257–286. Springer, Heidelberg (2012)
13. Håstad, J.: The shrinkage exponent of De Morgan formulas is 2. SIAM J. Comput. 27(1), 48–64 (1998)
14. Hirsch, E.A.: A new algorithm for MAX-2-SAT. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 65–73. Springer, Heidelberg (2000)
15. Hirsch, E.A.: Worst-case study of local search for MAX-k-SAT. Discrete Applied Mathematics 130(2), 173–184 (2003)
16. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM 21(2), 277–292 (1974)
17. Impagliazzo, R., Matthews, W., Paturi, R.: A satisfiability algorithm for AC^0. In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 961–972 (2012)
18. Impagliazzo, R., Paturi, R., Schneider, S.: A satisfiability algorithm for sparse depth two threshold circuits. In: Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 479–488 (2013)
19. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: On the parameterized complexity of exact satisfiability problems. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 568–579. Springer, Heidelberg (2005)
20. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: A bound on the pathwidth of sparse graphs with applications to exact algorithms. SIAM J. Discrete Math. 23(1), 407–427 (2009)
21. Koivisto, M.: Optimal 2-constraint satisfaction via sum-product algorithms. Inf. Process. Lett. 98(1), 24–28 (2006)

22. Kojevnikov, A., Kulikov, A.S.: A new approach to proving upper bounds for MAX-2-SAT. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 11–17 (2006)
23. Kulikov, A.S.: Automated generation of simplification rules for SAT and MAXSAT. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 430–436. Springer, Heidelberg (2005)
24. Kulikov, A.S., Kutzkov, K.: New bounds for MAX-SAT by clause learning. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 194–204. Springer, Heidelberg (2007)
25. Mahajan, M., Raman, V.: Parameterizing above guaranteed values: MaxSAT and MaxCUT. J. Algorithms 31(2), 335–354 (1999)
26. Niedermeier, R., Rossmanith, P.: New upper bounds for maximum satisfiability. J. Algorithms 36(1), 63–88 (2000)
27. Santhanam, R.: 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), pp. 183–192 (2010)
28. Schuler, R.: An algorithm for the satisfiability problem of formulas in conjunctive normal form. J. Algorithms 54(1), 40–44 (2005)
29. Scott, A.D., Sorkin, G.B.: Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 382–395. Springer, Heidelberg (2003)
30. Scott, A.D., Sorkin, G.B.: Linear-programming design and analysis of fast algorithms for Max 2-CSP. Discrete Optimization 4(3-4), 260–287 (2007)
31. Seto, K., Tamaki, S.: A satisfiability algorithm and average-case hardness for formulas over the full binary basis. Computational Complexity 22(2), 245–274 (2013)
32. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theor. Comput. Sci. 348(2-3), 357–365 (2005)
33. Williams, R.: Non-uniform ACC circuit lower bounds. In: Proceedings of the 26th Annual IEEE Conference on Computational Complexity (CCC), pp. 115–125 (2011)