Записки научных
семинаров ПОМИ
Том 399, 2012 г.

I. A. Bliznets

A NEW UPPER BOUND FOR $(n,3)$-MAX-SAT

Abstract. It is still not known whether the satisfiability problem (SAT), and hence maximum satisfiability problem (MAX-SAT), can be solved in time poly$(|F|)c^n$, for $c < 2$, where c is a constant, n is the number of variables, and F is an input formula. However, such bounds are known for some special cases of these problems where the clause length, the maximal number of variable occurrences or the length of the formula is bounded. In this paper, we consider the $(n,3)$-MAX-SAT problem – a special case of MAX-SAT where each variable appears in a formula at most three times. We present a simple algorithm with running time $O^*(2^{n/3})$. As a byproduct we also obtain a polynomially solvable subclass that may be of independent interest.

§1. Introduction

It is still not known whether the satisfiability problem (SAT), and hence maximum satisfiability problem (MAX-SAT), can be solved in time poly$(|F|)c^n$, for $c < 2$, where c is a constant, n is the number of variables, and F is an input formula. Therefore, many restricted versions of these problems are studied:

- k-SAT and MAX-k-SAT are special cases of SAT and MAX-SAT, respectively, where each clause of an input formula contains at most k literals;
- ClLin-SAT and ClLin-MAX-SAT take as input formulas containing at most Δn clauses, where Δ is a constant;
- (n,k)-SAT and (n,k)-MAX-SAT take as input formulas where each variable appears in a formula at most k times.
- Unique k-SAT is a special case of k-SAT where an input formula has at most one satisfying assignment.

Table 1 summarizes some of the known results for the mentioned problems (as usual, $O^*(\cdot)$ suppresses polynomial factors).

In this note, we focus on $(n,3)$-MAX-SAT problem. $(n,3)$-MAX-SAT is known to be NP-hard [6]. We prove that it can be solved in time

Key words and phrases: algorithm, maximum satisfiability, satisfiability.

Table 1. Known results.

problem	time	reference
k-SAT	$O^*((\frac{2(k-1)}{k}+\epsilon)^n)$	Moser, Scheder [3]
Unique 3-SAT	$O^*(1.308^n)$	Hertli [2]
Unique 4-SAT	$O^*(1.469^n)$	Hertli [2]
MAX-2-SAT	$O^*(1.74^n)$	Williams [10]
ClLin-MAX-SAT	$O^*(c^n), c < 2$	Dantsin, Wolpert [12]; Kulikov, Kutzkov [11]
(n,k)-SAT	$O^*(a^n), a < 2$	Wahlström [13]
$(n,3)$-MAX-SAT	$O^*(1.273^n)$	Kulikov [5]

$O^*(2^{\frac{n}{3}}) = O^*(1.25993^n)$. This improves the previously known upper bound $O^*(1.27203^n)$ [5]. We prove an upper bound by the standard splitting technique. Informally, we show how to reduce any formula with n variables to two formulas with $(n-3)$ variables. This implies an upper bound $O^*(2^{\frac{n}{3}})$.

As a byproduct we also obtain a polynomially solvable subclass that may be of independent interest. Namely, by finding a maximum matching in a special graph we are able to solve $(n,3)$-MAX-SAT for the following formulas:

- each variable appears once negatively and twice positively;
- all negative literals occur in unit clauses only.

Example of such a formula is

$$(\overline{x}) \wedge (\overline{y}) \wedge (\overline{z}) \wedge (x \vee y \vee z) \wedge (x \vee y) \wedge (z).$$

§2. Definitions and Notation

For a Boolean variable x, we say that x is *a positive literal* and $\overline{x}$ is *a negative literal*. By l_x we denote either x or $\bar{x}$, i.e., l_x is a literal of the variable x. A *complementary literal* for a literal l is $\bar{l}$. We say that a literal l is *a pure literal in a formula F*, if the complementary literal does not occur in F (we omit F if it is clear from the context). A clause is a disjunction of literals. A formula is a conjunction of clauses. We usually refer to a clause and a formula just as a set of literals and a multiset of clauses, respectively. Through the rest of the paper we consider only formulas where each variable appears at most three times. For formulas F_1 and F_2, we say that F_1 is a *subformula* of F_2 if $F_1 \subseteq F_2$ as multisets of clauses. F_2 is called *a closed subformula* of F_1, if no variable appears in both $F_1 \backslash F_2$ and F_2.

For simplicity, we denote from now on the Boolean value true by 1 and false by 0. We allow a formula to contain also a special clause (T) that is always satisfied and does not depend on any variable (this clause is needed for counting the number of currently satisfied clauses). For a literal l and a formula F, by $F[l]$ we denote a formula resulting from F by assigning the value 1 to l. To do this, we first replace each clause containing the literal l by (T), then remove the literal $\bar{l}$ from the remaining clauses. This definition extends naturally to several literals: e.g., $F[l_1, l_2] = F[l_1][l_2]$. By $F[l_1 = l_2]$ we denote a formula resulting from F by replacing all occurrences of l_2 and $\bar{l_2}$ by l_1 and $\bar{l_1}$, respectively. *The V-neighborhood of a literal* l, denoted by $VN(l)$, is the set of all variables that appear in the same clause with literal l.

Let $\mathrm{Opt}(F)$ be the maximum number of clauses of F that can be simultaneously satisfied. Denote by $V(F)$ the set of all variables appearing in F. *A truth assignment* to variables, $\tau \colon V(F) \to \{0, 1\}$, is a function that assigns every variable of F a Boolean value true or false. *An optimum assignment* for a formula is a truth assignment which satisfies the largest possible number of clauses of this formula.

§3. Algorithm

The algorithm is given below. We analyze its running time in the following section.

```
Algorithm N3MaxSat
Input: A formula $F$ in CNF
Output: The maximum number of simultaneously satisfied clauses
Normalizing rules:
N1 Remove all empty clauses
N2 Replace all clauses containing a pair of complementary literals with $(T)$
N3 If $(F = F_1 \wedge F_2)$, where $F_1, F_2$ are closed subformulas,
  then return N3MaxSat($F_1$) + N3MaxSat($F_2$)
N4 Rename all variables such that for any variable $x$, $\bar{x}$ appears only once
```

```
Simplification rules:
S1 If ($F$ contains a pure literal $l$), then return N3MaxSat($F[l]$)
S2 If ($F = (x \vee A) \wedge (\overline{x} \vee B) \wedge F_1$), where $A, B$ are clauses, $F_1$ is a
formula that does not contain $x$ and $\overline{x}$, then return N3MaxSat($(A \vee B) \wedge F_1) + 1$
S3 If ($F$ contains a clause $(x)$), then return N3MaxSat($F[x]$)
S4 If ($F = (x \vee y) \wedge (x \vee y) \wedge (\overline{x} \vee A) \wedge (\overline{y} \vee B) \wedge F_1$),
  then return N3MaxSat($(A \vee B) \wedge F_1) + 3$
S5 If ($F = (x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee A) \wedge (y \vee B) \wedge F_1$),
  then return N3MaxSat($F_1 \wedge A) + 3$
S6 If ($F = (\overline{x} \vee y) \wedge C \wedge F_1$ and ($l_x, l_y \in C$) ), where $C$ is a clause,
$F_1$ is a formula,
  then return N3MaxSat($F[x = y]$)
Branching rules:
B1 If (($F = C \wedge F_1$) and ($\overline{x}, \overline{y} \in C$)), where $C$ is a clause, $F_1$ is a
formula,
  then return max(N3MaxSat($F[x]$), N3MaxSat($F[\bar{x}]$))
B2 If (($F = C \wedge F_1$) and ($\overline{x}, y, z \in C$)), where $C$ is a clause, $F_1$ is a
formula,
  then return max(N3MaxSat($F[x]$), N3MaxSat($F[\bar{x}]$))
B3 If ($F = (y \vee \bar{x}) \wedge F_1$),
  then return max(N3MaxSat($F[x, y]$), N3MaxSat($[\bar{x}, \bar{y}]$))
Matching:
Comment: $F$ depends on $n$ variables and contains $k$ ($\mathcal{T}$) clauses
Construct the following graph $G_F$:
— introduce a vertex for every clause consisting of positive literals
only;
— introduce an edge between two vertices for every variable, if the
corresponding clauses share this variable ($G_F$ could be a multigraph).
Find a maximum matching $M$ in the graph $G_F$, return $k + n + |M|$
```

§4. Analysis

In this section, we first prove that the algorithm is correct and then prove an upper bound $O^*(2^{n/3})$ on its running time.

4.1. Correctness of normalizing, simplification and branching rules. Correctness of the normalizing rules as well as the first two branching rules is obvious. Below we prove that all the remaining rules are correct.

Throughout all the subsection we assume that the formula under consideration is normalized. Recall that in normalized formula negations of all the variables appear exactly once.

Lemma 4.1 (correctness of **S2**). *If clauses A, B and formula F_1 do not depend on a variable x, then*

$$\mathrm{Opt}((x \vee A) \wedge (\overline{x} \vee B) \wedge F_1) = \mathrm{Opt}((A \vee B) \wedge F_1) + 1.$$

Proof. First we prove that $\mathrm{Opt}((x \vee A) \wedge (\overline{x} \vee B) \wedge F_1) \geqslant \mathrm{Opt}((A \vee B) \wedge F_1) + 1$. For any truth assignment τ that satisfies k clauses in the formula $(A \vee B) \wedge F_1$ we give a truth assignment τ_{new} that satisfies $k+1$ clauses in the formula $(x \vee A) \wedge (\overline{x} \vee B) \wedge F_1$. τ_{new} depends on τ and the values of A and B under τ.

$A = 0, B = 0$	$\tau_{\text{new}} = \tau \cup \{x = 1\}$
$A = 0, B = 1$	$\tau_{\text{new}} = \tau \cup \{x = 1\}$
$A = 1, B = 0$	$\tau_{\text{new}} = \tau \cup \{x = 0\}$
$A = 1, B = 1$	$\tau_{\text{new}} = \tau \cup \{x = 1\}$

The reverse inequality $\mathrm{Opt}((x \vee A) \wedge (\overline{x} \vee B) \wedge F_1) \leqslant \mathrm{Opt}((A \vee B) \wedge F_1) + 1$ is easier to show: τ satisfies k clauses of $(x \vee A) \wedge (\overline{x} \vee B) \wedge F_1$, so τ satisfies at least $k-1$ of $(A \vee B) \wedge F_1$. □

Lemma 4.2 (correctness of **S3**). *If $F = (x) \wedge G$, then* $\mathrm{Opt}(F[x]) \geqslant \mathrm{Opt}(F[\bar{x}])$.

Proof. It is easy to see that by changing the value of x from 0 to 1 in any truth assignment, we can only increase the number of satisfied clauses, as we gain the clause (x) for sure and we may lose the clause containing $\bar{x}$. □

Lemma 4.3 (correctness of **S4**). *Let*

$$F = (x \vee y) \wedge (x \vee y) \wedge (\overline{x} \vee A) \wedge (\overline{y} \vee B) \wedge F_1,$$

where formula F_1 and disjunctions of literals A, B do not depend on variables x, y. Then $\mathrm{Opt}(F) = \mathrm{Opt}(F_1 \wedge (A \vee B)) + 3$.

Proof. It is obvious that $\mathrm{Opt}(F) \leqslant \mathrm{Opt}(F_1 \wedge (A \vee B)) + 3$. Like in Lemma 4.1, it is enough to consider the same truth assignment for both formulas.

Now, we show that $\mathrm{Opt}(F) \geqslant \mathrm{Opt}(F_1 \wedge (A \vee B)) + 3$. For any truth assignment of formula $F_1 \wedge (A \vee B)$ we construct a truth assignment τ_{new} that satisfies at least three clauses more in the formula F. τ_{new} depends on the values of A, B under τ.

$A=0, B=0$	$\tau_{\text{new}} = \tau \cup \{x=1, y=0\}$
$A=0, B=1$	$\tau_{\text{new}} = \tau \cup \{x=0, y=1\}$
$A=1, B=0$	$\tau_{\text{new}} = \tau \cup \{x=1, y=0\}$
$A=1, B=1$	$\tau_{\text{new}} = \tau \cup \{x=1, y=0\}$

□

Lemma 4.4 (correctness of **S5**). *Let*

$$F = (x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee A) \wedge (y \vee B) \wedge F_1,$$

where F_1 does not depend on variables x and y. Then

$$\text{Opt}(F) = \text{Opt}(F_1 \wedge A) + 3.$$

Proof. If τ is a truth assignment satisfying k clauses of F_1, then the truth assignment $\tau \cup \{x=1, y=1\}$ satisfies $k+3$ clauses of F. So,

$$\text{Opt}(F) \geqslant \text{Opt}(F_1 \wedge A) + 3.$$

If $\text{Opt}(F) > \text{Opt}(F_1 \wedge A) + 3$, then there is a truth assignment that satisfies in F four clauses more than in $F_1 \wedge A$. Hence, under this assignment $A = 0$ and $x = 0$. Now, it is impossible to satisfy $(x \vee y)$, $(x \vee \overline{y})$ simultaneously. A contradiction. □

Lemma 4.5. *If $F = (\bar{x} \vee y) \wedge F_1$, then there exists an optimal assignment τ such that $\tau(x) = \tau(y)$.*

Proof. It is enough to prove that

$$\text{Opt}(F[x,y]) \geqslant \text{Opt}(F[x,\bar{y}]) \quad \text{and} \quad \text{Opt}(F[x,y]) \geqslant \text{Opt}(F[y,\bar{x}]).$$

The first claim follows from the the following fact if $x = 1$, then we can assign $y = 1$ due to Lemma 4.2, so $\text{Opt}(F[x,y]) \geqslant \text{Opt}(F[x,\bar{y}])$. The second claim follows from the following if $y = 1$, then due to **S1** we can assign $x = 1$, so $\text{Opt}(F[x,y]) \geqslant \text{Opt}(F[\bar{x},y])$. □

Correctness of **B3** follows from the just proved lemma.

Lemma 4.6 (correctness of **S6**). *Let $F = (\overline{x} \vee y) \wedge C \wedge F_1$ and $l_x, l_y \in C$. Then* $\text{Opt}(F[x=y]) = \text{Opt}(F)$. *After normalizing, the formula $F[x=y]$ contains at most 3 occurrences of the variable x.*

Proof. Due to Lemma 4.5, it is enough to consider assignments in which $x = y$. Replace y with x in the formula F. In the new formula variable x occurs 6 times. But we can eliminate the clause $(x \vee \overline{x})$ as well as the clause C or at least one literal from it. □

4.2. Correctness of the last step. If normalizing, simplifying, and branching rules are not applicable to F, then any clause containing a negative literal has length exactly one. Given such a formula F, the algorithm constructs the graph G_F and finds a maximum matching in it. Note that the number of variables in F is equal to the number of clauses with negative literals. Moreover, this number equals also the number of edges in G_F.

Theorem 4.7. *If each variable of a formula F appears once negatively and twice positively and all negative literals occur in unit clauses, then*

$$\mathrm{Opt}(F) = V(G_F) + E(G_F) - \rho(G_F) = E(G_F) + \nu(G_F),$$

where $E(G_F)$, $V(G_F)$, $\nu(G_F)$, $\rho(G_F)$ are correspondingly the number of edges, the number of vertices, the size of a maximum matching, the size of a minimum edge cover in the graph G_F.

Proof. Consider an optimum truth assignment τ for F. Without loss of generality suppose that τ satisfies all clauses with positive literals (otherwise we can flip the value of a variable in an unsatisfied clause with positive literals without decreasing the number of satisfied clauses). Note that all the edges labeled by the variables that are set to true by τ form an edge cover of G_F. Thus, $\mathrm{Opt}(F) = V(G_F) + E(G_F) - \rho(G_F)$. To conclude the proof we use the following well-known fact: $\rho(G_F) + \nu(G_F) = V(G_F)$ (see [7]). $\square$

4.3. Running time. In this subsection, we prove an upper bound on the running time of the presented algorithm by estimating the number of variables in two formulas that the algorithm gets after branching. In all the analysis below we assume that the considered formula is already normalized and simplified.

Lemma 4.8. *In a formula $F[x]$ we can eliminate at least two variables.*

Proof. Consider V-neighborhood of x. Note that $|VN(x)| \geqslant 2$, as otherwise one of the **S3**, **S4**, **S5**, **N3** rules would be applicable to F. Then in a formula $F[x]$, there are at least two variables that appear at most two times. They will be eliminated by normalizing and simplifying rules. $\square$

Lemma 4.9. *If $F = C \wedge F_1$ and $\bar{x}, \bar{y} \in C$, where C is a clause, F_1 is a formula, then in formulas $F[x]$, $F[\bar{x}]$ we can eliminate (using normalizing or simplification rules) two variables.*

Proof. The statement for $F[x]$ follows immediately from Lemma 4.8. Now, consider $F[\bar{x}]$. If we assign the value 0 to x, then y will be assigned 1 due to simplifying rules. So, we again get a special case of Lemma 4.8, but instead of x we are using y. □

Below in this subsection we assume that **B1** is not applicable to a formula F.

Lemma 4.10. *If $F = C \wedge F_1$ and $\bar{x}, y, z \in C$, where C is a clause, F_1 is a formula, then in formulas $F[x]$, $F[\bar{x}]$ we can eliminate (using normalizing or simplification rules) at least two variables.*

Proof. To get the desired result it is enough to see that

$$2 \leqslant \min\{|VN(x)|, |VN(\overline{x})|\}.$$

Then the proof is identical to the proof of Lemma 4.8. □

Below in this subsection we assume that rules **B1** and **B2** are not applicable to a formula F.

Lemma 4.11. *Let $F = (\bar{x} \vee y) \wedge F_1$. Then either at least two variables are eliminated in $F[x]$ and $F[\bar{x}]$ or one variable is eliminated in $F[\bar{x}]$ and four variables in $F[x]$.*

Proof. Lemma 4.5 states that there exists an optimum assignment τ such that $\tau(x) = \tau(y)$

Consider the following cases:

- All clauses containing literals x, y , except $(\overline{x} \vee y)$, have length at least three. In this case $|VN(x) \cup VN(y)| \geqslant 5$, because these clauses contain only positive literals and the total length of these three clauses is at least 9. So, if $x = 1$, then $y = 1$ and we can eliminate 3 additional variables. If $x = 0$, then $y = 0$, so we eliminate one variable.
- In the formula there is a clause $(x \vee \overline{z})$ or $(y \vee \overline{z})$ and variable z differs from x, y then it is enough to consider two cases $x = y = z = 0$ or $x = y = z = 1$ (like it was mentioned before). So, in this case we eliminate 2 variables in formulas $F[x], F[\bar{x}]$.
- The only remaining case is when in addition to the clause $(y \vee \overline{x})$ we have a clause $x \vee z$ or $y \vee z$. If $x = 1$, we eliminate 2 variables in $F[x]$ due to Lemma 4.8. If $x = 0$, then $y = 0$. Hence, due to simplification rule **S3**, we have $z = 1$.

□

Theorem 4.12. *The running time of the algorithm N3MaxSat is* $O^*(2^{\frac{n}{3}})$.

Proof. Denote by $T(n)$ the worst-case running time of the algorithm for formulas with n variables. Note that normalizing, simplifying, branching rules take time polynomial in n (clearly, the length of the formula is linear). Moreover, if we apply any case of normalizing, simplification rules, then the number of variables in the resulting formula does not increase and the length of the formula decreases.

We now consider branching rules. When we apply any branching rule we get two smaller problems. From Lemmas 4.9, 4.10, and 4.11, we get the following recurrence relations on the running time

$$T(n) \leqslant T(n-2) + T(n-5) + \text{poly}(n),$$
$$T(n) \leqslant 2T(n-3) + \text{poly}(n).$$

And final stage of our algorithm, construction of a graph G_F and obtaining a maximum matching, takes only polynomial time [7].

It is well known that in this case $T(n) = O^*(\lambda^n)$, where λ is the largest root of the equations $x^3 - 2 = 0$, $x^5 - x^3 - 1 = 0$. The largest root of the first equation is $1.25993\ldots$ and it is bigger than $1.23652\ldots$ which is the largest root of the second equation. This concludes the proof.

□

Acknowledgments

I would like to thank Alexander S. Kulikov for help in writing this paper and the anonymous referees for valuable comments.

References

1. J. Chen, I. Kanj, *Improved exact algorithms for* Max-Sat. — Lect. Notes Computer Sci. **2286** (2002), 98–119.
2. H. Timon, 3-SAT *Faster and Simpler – Unique*-SAT *Bounds for* PPSZ *Hold in General.* — CoRR **abs/1103.2165** (2011), `http://arxiv.org/abs/1103.2165`.
3. R. A. Moser, D. Scheder, *A full derandomization of Schöning's k*-SAT *algorithm.* — In: Proceedings of the 43rd annual ACM Symposium on Theory of Computing, San Jose, California, USA (2011), pp. 245–252.
4. S. S. Fedin, A. S. Kulikov, *Automated proofs of upper bounds on the running time of splitting algorithms.* — Lect Notes Computer Sci., Springer **3162** (2004), 248–259.
5. A. Kulikov, *Automated generation of simplification rules for* SAT *and* MAXSAT. — Lect Notes Computer Sci., Springer **3569** (2005), 43–59.

6. Venkatesh Raman, B. Ravikumar, S. Srinivasa Rao, *A simplified* NP*-complete* MAXSAT *problem.* — Information Processing Lett. **65**, No. 1 (1998), 1–6.
7. J. Edmonds, Paths, Trees, and Flowers. — In: Modern Birkhäuser Classics, Birkhüser Boston (1987), pp. 361–379.
8. E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, U. Schяning, *A deterministic* $(2-2/(k+1))^n$ *algorithm for* k-SAT *based on local search.* — Theor. Computer Sci. **289**, No. 1 (2002) 69–83.
9. D. Scheder, *Guided search and a faster deterministic algorithm for 3-SAT.* — Lect. Notes Computer Sci. Springer Berlin–Heidelberg **4957** (2008), 60–71.
10. W. Ryan, *A new algorithm for optimal* 2*-constraint satisfaction and its implications.* — Theor. Computer Sci. **348**, No. 2 (2005), 357–365.
11. A. Kulikov, K. Kutzkov, *New bounds for* MAX-SAT *by Clause Learning.* — Lect. Notes Computer Sci. Springer Berlin–Heidelberg **4649** (2007), 194–204.
12. E. Dantsin, A. Wolpert, MAX-SAT *for formulas with constant Clause density can be solved faster than in* $O^*(2^n)$ *time.* — Lect. Notes Computer Sci. Springer Berlin–Heidelberg **4121** (2006), 266–276.
13. W. Magnus, *An algorithm for the* SAT *problem for formulae of linear length.* — Lect. Notes Computer Sci. Springer Berlin–Heidelberg **3669** (2005), 107–118.

St.Petersburg University of the
Russian Academy of Sciences
St.Petersburg 198504, Russia
E-mail: ivanbliznets@tut.by

Поступило 15 января 2012 г.