PDMI PREPRINT — 4/1997

# DECIDING SATISFIABILITY OF FORMULAS WITH $K$ CLAUSES IN LESS THAN $2^{0.30897K}$ STEPS

**EDWARD A. HIRSCH**

Russian Academy of Sciences
Steklov Mathematical Institute
St. Petersburg Department
Fontanka 27, St. Petersburg 191011, Russia
hirsch@pdmi.ras.ru

Febrary 1997

ABSTRACT

In 1980 B. Monien and E. Speckenmeyer proved the worst-case upper bound $2^{K/3}$ on the time complexity of the propositional satisfiability problem, where $K$ is the number of clauses in the input formula. The present paper improves this bound to $2^{0.30897K}$ by using a new method, called *the extended sign principle*.

It is well-known that SAT (the problem of satisfiability of a formula in CNF) is NP-complete (see, for example, [GJ]), all known decision algorithms for it have exponential upper bounds on time. Nevertheless, the trivial upper bound for this problem $2^N$ ($N$ is the number of variables in the input formula) was reduced for formulas in 3-CNF in the early 1980s by Monien and Speckenmeyer ([MS1], see also [MS3]) and independently by Dantsin ([D], see also [DK]). After that, many upper bounds for SAT and its subproblems were obtained ([L,Sch1,KL,K,Sch2]). These bounds were obtained with respect to three main parameters: the length $L$ of the input formula, the number $K$ of its clauses and the number $N$ of the variables occuring in it. The best upper bounds for SAT to this moment are:

- $p(L)2^{L/9}$ ([L], see also [KL]),
- $p(L)2^{K/3}$ ([MS2], see also [KL]),

where $p$ is a suitable polynomial. It is still unknown whether the trivial upper bound $2^N$ for the general SAT problem can be pushed down. Kullmann and Luckhardt in [KL] simplified the algorithm of Monien and Speckenmeyer from [MS2] and presented an improved algorithm which has the same complexity $p(L)2^{K/3}$ with respect to $K$, but works better when the ratio of the number of clauses and the number of variables is greater than 2.64557. In this paper the algorithm is given which decides SAT in time $p(L)2^{0.30897K}$.

Most of the algorithms for SAT and its NP-complete subproblems use *splitting*. This method goes back to the Davis-Putnam procedure ([DP]). In short, the main idea is that formula $F$ is satisfiable iff at least one of the formulas $F[a]$ and $F[\overline{a}]$ is satisfiable. In wider sense ([DLL]), any splitting algorithm builds some *branching tree* by reducing the satisfiability problem for formula $F$ to the satisfiability problem for several formulas $F[I_1]$, $F[I_2]$, ... ,$F[I_s]$ obtained from $F$ by assignments $I_1$, $I_2$, ... ,$I_s$ respectively. After this "exponential" reduction, the algorithm performs *polynomial reductions* for the formulas $F[I_1]$, $F[I_2]$, ... ,$F[I_s]$. These reductions take polynomial time on each of these formulas and their role is to reduce the appropriate parameter $L$, $K$ or $N$ of the formula and to simplify the formula by eliminating pure literals, 1-clauses and other "easy pieces" from it.

A general method for the estimation of the size of such a tree (and thus, the algorithm running time), was proposed by Kullmann and Luckhardt in [KL]. We associate each node of this tree with *a branching vector* of non-negative numbers and a polynomial associated with this vector. (For precise definitions, we refer the reader to §2.) We can estimate the number of leaves in the tree using the largest of the positive roots of the polynomials associated with the nodes of the branching tree. We present this method in §3.

The running time of an algorithm that uses splitting highly depends on the polynomial reductions which it performs. The first two reductions were proposed in the original paper of Davis and Putnam ([DP]). We can eliminate pure literals (a literal is pure if its negation does not occur in the formula) and 1-clauses. We also apply the original reduction by Davis and Putnam in the case it does not increase the number of clauses in the formula. We do it by eliminating from the formula all clauses that contain some variable $v$ and by adding to the formula all resolvents of these clauses (the notion of the resolvent was introduced in [R]). In fact, after applying these three reductions, the formula does not contain literals that occur only once (1-literals), and each variable occurs in it at least five times.

The next reduction is the key point of our algorithm. Let $P(l, F)$ be an arbitrary property of pairs of literals and formulas in CNF containing these literals. Kullmann and Luckhardt proposed in [KL] *the generalized sign principle*.

*The generalized sign principle.* If for all variables $v$ of the formula $F$ in CNF, the property $P(l, F)$ holds for at least one of the literals $l = v$ or $l = \overline{v}$, then there exist only two possibilities:

(1) there exists a clause $\{l_1, l_2, \ldots, l_s\}$ in $F$ such that $P(l_i, F)$ holds for each of the literals $l_i$,

(2) $F$ is satisfiable.

Here we present a further generalization of this principle, which we call *the extended sign principle*. The key idea is that the generalized sign principle can be applied to a part of the variables occuring in the formula $F$.

*The extended sign principle.* If for all variables $v$ of the formula $F$ in CNF, the property $P(l, F)$ holds for at most one of the literals $l = v$ or $l = \overline{v}$, then there exist only two possibilities:

(1′) there exists a clause $\{l_1, l_2, \ldots, l_s\}$ in $F$ such that $P(\overline{l}_i, F)$ holds for none of the literals $\overline{l}_i$ and $P(l_j, F)$ holds for at least one of the literals $l_j$,

(2′) if we eliminate from $F$ all clauses that contain literals $l$ such that $P(\overline{l}, F)$ holds, then we do not affect satisfiability of the formula.

Suppose that the formula $F$ does not satisfy (1′). Then each of its clauses that contains a literal satisfying the property $P$ contains also a negation of a literal satisfying $P$. Consider the assignment $I$ that consists of the negations of literals satisfying $P$. The formula $F[I]$ is a subset of the formula $F$. Now the correctness of this principle is a consequence of *the autarkness principle* suggested by Monien and Speckenmeyer ([MS1, MS3]) and independently by Luckhardt ([L], see also [KL]).

*The autarkness principle.* If substituting truth variables in some variables of the formula $F$ does not result in arising of new clauses (including subsets of old clauses), then satisfiability of the obtained formula is equivalent to satisfiability of $F$.

More precisely, consider an assignment $J$ that satisfies the formula $F$. If we change the values of the literals having the property $P$ to *False* in the assignment $J$, then we obtain a satisfying assignment again. The reverse is trivial. Thus, the extended sign principle is correct. It gives us a chance to do each time a "good" splitting by the clause which existence is guaranteed by (1′). Otherwise, we perform the reduction (2′). If it does not change the formula, then we have the formula of a very special kind. We shall use the property $P(l, F)$ =*the literal $l$ occurs in $F$ twice, and the literal $\overline{l}$ occurs in $F$ at least three times.* Thus, if (1′) does not hold and the reduction (2′) does not change the formula, then each literal occurs at least three times in it. The splitting by a literal that occurs more than three times in this case is "good". If each literal occurs in $F$ exactly three times, then we are forced to perform the splitting by such a literal, but such a formula is very "unstable", i.e. the formulas that are produced from it by splitting, must contain literals that occur in $F$ more or less than three times. For further details we refer the reader to §4 which contains the algorithm and the estimation of its running time.

## §2. Basic definitions

Let $V$ be a finite set of Boolean variables. The negation of a variable $v$ is denoted by $\overline{v}$. Given a set $U$, we denote $\overline{U} = \{\overline{u} \mid u \in U\}$. *Literals* are the members of the set $W = V \cup \overline{V}$. *Positive literals* are the members of the set $V$. *Negative literals* are their negations. If $w$ denotes a negative literal $\overline{v}$, then $\overline{w}$ denotes the variable $v$. A clause is a finite set of literals $\{w_1, w_2, \dots, w_s\} \subset W$. A clause does not contain simultaneously any variable together with its negation, i.e. $w_i \neq \overline{w}_j$ for $i \neq j$. The empty clause is interpreted as *False*. *A formula in CNF* (*CNF-formula*) is a finite set of clauses. The empty formula is interpreted as *True*. The *length of a clause* is its cardinality, the *length of a formula* is the sum of the lengths of all its clauses. A *$k$-clause* is a clause of the length $k$. If we say that *a literal $v$ occurs* in a clause or in a formula, we mean that this clause or this formula contains a literal $v$. However, if we say that *a variable $v$ occurs* in a clause or in a formula, we mean that this clause or this formula contains a literal $v$, or it contains a literal $\overline{v}$. In what follows $F$ is a formula in CNF, and all its variables are contained in $V$.

A literal $l$ is a $i$-literal if the formula $F$ contains exactly $i$ occurences of $l$. A literal $l$ is a *($i$+,$j$+)-literal* if the formula $F$ contains at least $i$ occurences of $l$ and at least $j$ occurences of $\overline{l}$. It is a *($i$,$j$+)-literal* if the formula $F$ contains exactly $i$ occurences of $l$ and at least $j$ occurences of $\overline{l}$. It is a *($i$+,$j$)-literal* if the formula $F$ contains at least $i$ occurences of $l$ and exactly $j$ occurences of $\overline{l}$. It is a *($i$,$j$)-literal* if the formula $F$ contains exactly $i$ occurences of $l$ and exactly $j$ occurences of $\overline{l}$.

Suppose we are given literal $l$ and clauses $C_1$ and $C_2$ such that $l$ is the only literal satisfying $l \in C_1$, $\overline{l} \in C_2$. In this case the clause $C_1 \cup C_2 \setminus \{l, \overline{l}\}$ is called the *$l$-resolvent* of the clauses $C_1$ and $C_2$.

For a given literal $l$, we construct the formula $DP_l(F)$ by

- adding all $l$-resolvents to $F$;
- eliminating from $F$ all clauses containing $l$ or $\overline{l}$.

*An assignment* is a finite subset of $W$ that does not contain simultaneously any variable together with its negation.

We write $g = f[I]$ for an assignment $I = \{w_1, \dots, w_s\}$ and CNF-formulas $f$ and $g$ if $g$ can be obtained from $f$ by

- eliminating all clauses that contain the literals $w_i$, and
- deleting the literals $\overline{w}_i$ from the other clauses.

We then say that *$g$ is obtained from $f$ by means of $I$*. For short we write $f[w_1, \dots, w_s]$ instead of $f[\{w_1, \dots, w_s\}]$.

*A branching tree* is a tree whose nodes are labelled with CNF-formulas such that if some node is labelled with CNF-formula $f$, then its sons are labelled with formulas $f[I_1]$, $f[I_2]$, ... , $f[I_s]$ for some assignments $I_1$, $I_2$, ... , $I_s$.

**Example** of a branching tree is shown in Figure 1.

Figure 1

In fact, our algorithm constructs some branching tree. Its nodes each can be labelled with some CNF-formula in a natural way. To estimate the number of nodes in this tree (and thus the time complexity of the algorithm) we use some auxiliary results concerning *branching tuples* introduced by Kullmann and Luckhardt in [KL]. These results are presented in §3, and here we define some relevant terminology.

Suppose we have a tree $T$ whose nodes are labelled with objects associated with some characteristic of their complexity: to each object $F$ we attach a non-negative integer $\mu(F)$ (*complexity of* $F$). Assume that this tree has the property that the complexity of an object labelling any node is greater than the complexity of each of the objects labelling its sons.

**Typical examples of $\mu$:**
1) objects are CNF-formulas, $\mu_1(F)$ is the number of variables in the formula $F$;
2) objects are CNF-formulas, $\mu_2(F)$ is the length of $F$;
3) objects are CNF-formulas, $\mu_3(F)$ is the number of clauses in $F$.

In this paper we use only $\mu_3$. We denote it by $\mathfrak{K}$. However, in §3 we do not fix any concrete measure of complexity. It is even immaterial for us now that our objects are CNF-formulas.

Figure 2

Let us consider an arbitrary node in our tree labelled with object $F_0$. Suppose its sons are labelled with $F_1$, $F_2$, ... , $F_m$ (see Figure 2). *A branching vector of a node* is an $m$-tuple $(t_1, \ldots, t_m)$ where $t_i$ are positive numbers not exceeding $\mu(F_0) - \mu(F_i)$.

*The branching number* $\tau(\vec{t})$ *of a node* with branching vector $\vec{t} = (t_1, t_2, \ldots, t_m)$ is the only positive root of *the characteristic polynomial of the branching vector* $\vec{t}$

$$\mathfrak{h}_{\vec{t}}(x) = 1 - \sum_{i=1}^{m} x^{-t_i}.$$

We suppose $\tau(\emptyset) = 1$ for leaves.

*Remark.* The characteristic polynomial $\mathfrak{h}_{\vec{t}}(x)$ is a monotone function of $x$ on the interval $(0, +\infty)$, and, $\mathfrak{h}_{\vec{t}}(0) = -\infty$, $\mathfrak{h}_{\vec{t}}(+\infty) = 1$. Hence, $\mathfrak{h}_{\vec{t}}(x) = 0$ has a unique solution on $(0, +\infty)$.

The largest of the branching numbers $\tau(\vec{t})$ of all nodes of the tree $T$ we call *the branching number of the tree* $T$. We denote it $\tau_{max,T}$.

## §3. The number of nodes in a branching tree

**Lemma 3.1** (Kullmann, Luckhardt, [KL]). *Let $T$ be a branching tree, let its root be labelled with the object $F_0$. Then the number of leaves in the tree $T$ does not exceed $(\tau_{max,T})^{\mu(F_0)}$.*

*Proof* by induction on the construction of the tree.

*Base.* The tree consisting of one node. We have $\tau_{max,T} = 1$, $(\tau_{max,T})^{\mu(F_0)} = 1^{\mu(F_0)} = 1$, that is not greater than the number of leaves in this tree.

*Step.* We consider the tree $T$ presented in Figure 3.

Figure 3

Let $\vec{t} = (t_1, t_2, \ldots, t_s)$ be the branching vector of its root.

$$
\begin{aligned}
&\text{The number of leaves of the tree } T = \\
&= \sum_{j=1}^{s} (\text{the number of leaves of the tree } T_j) \\
&\leqslant \sum_{j=1}^{s} (\tau_{max,T_j})^{\mu(F_j)} \quad \text{(by the induction hypothesis)} \\
&\leqslant \sum_{j=1}^{s} (\tau_{max,T_j})^{\mu(F_0)-t_j} \quad \text{(by the definition of the branching vector)} \\
&\leqslant \sum_{j=1}^{s} (\tau_{max,T})^{\mu(F_0)-t_j} \quad (\text{since } \tau_{max,T_j} \leqslant \tau_{max,T}) \\
&= (\tau_{max,T})^{\mu(F_0)} \cdot \sum_{j=1}^{s} (\tau_{max,T})^{-t_j} \\
&= (\tau_{max,T})^{\mu(F_0)} \cdot (1 - \mathfrak{h}_{\vec{t}}(\tau_{max,T})) \quad (\text{by the definition of } \mathfrak{h}_{\vec{t}}) \\
&\leqslant (\tau_{max,T})^{\mu(F_0)} \quad (\text{by monotonicity of } \mathfrak{h}_{\vec{t}})
\end{aligned}
$$

□

## §4. Algorithm

**Function** *Reduce***.**
*Input:* A formula $F$ in CNF.
*Output:* A (reduced) formula in CNF.
*Method:*

(R1) (Elimination of pure literals.) If there exists literal $l$ in the formula $F$ such that $\bar{l}$ does not occur in $F$, then $F := F[l]$.

(R2) (Elimination of 1-clauses.) If $F$ contains a 1-clause $C = \{l\}$, then $F := F[l]$.

(R3) (Reduction by Davis-Putnam.) Choose $l$ such that $\Delta = \mathfrak{K}(F) - \mathfrak{K}(DP_l(F))$ is maximal. If $\Delta \geqslant 0$, then $F := DP_l(F)$.

(R4) If the formula $F$ has been changed at steps (R1)–(R3), then go to step (R1), otherwise goto step (R5).

(R5) (Extended sign principle application.) If each clause $C$ in the formula $F$ that contains a (2,3+)-literal, contains also a (3+,2)-literal, then $F := F[\{l \,|\, l \text{ is a (3+,2)-literal}\}]$.

(R6) If the formula $F$ has been changed at step (R5), then go to step (R1), otherwise return $F$.

□

**Function** *Split***.**
*Input:* A formula $F$ in CNF.
*Output:* If the input formula $F$ is satisfiable, then $True$, otherwise $False$.
*Method:*

(S1) (The empty formula.) If $F = \emptyset$, then return $True$.

(S2) (Formula containing the empty clause.) If $\emptyset \in F$, then return $False$.

(S3) (Splitting by a (3+,4+)-literal.) If the formula $F$ contains a (3+,4+)-literal $l$, then construct two formulas

$$F_1 = Reduce(F[l]),$$
$$F_2 = Reduce(F[\bar{l}]).$$

If *Split* returns $True$ for at least one of these, then return $True$, otherwise return $False$.

(S4) (Splitting by a clause containing only 2-literals.) If the formula $F$ contains a clause $C = \{l_1, l_2, \ldots, l_s\}$ that contains only 2-literals, then choose literal $l$ in it such that $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[l]))$, $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\bar{l}])) \geqslant 3$ and $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[l])) + \mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\bar{l}])) \geqslant 7$. Construct for the chosen literal $l$ two formulas

$$F_1 = Reduce(F[l]),$$
$$F_2 = Reduce(F[\bar{l}]).$$

If *Split* returns $True$ for at least one of these, then return $True$, otherwise return $False$.

(S5) (Splitting by a (3,3)-literal occuring together with a 2-literal.) Suppose there exists a clause $C$ in the formula $F$ such that

- $C$ contains at least one (3,3)-literal $l$, $(\star)$
- $C$ contains at least one 2-literal $w$. $(\star\star)$

Construct two formulas

$$F_1 = Reduce(F[l]),$$
$$F_2 = Reduce(F[\bar{l}]).$$

If *Split* returns $True$ for at least one of these, then return $True$, otherwise return $False$.

(S6) (Splitting by a "good" (3,3)-literal.) Choose (3,3)-literal $l$ in the formula $F$ such that $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[l])) \geqslant 4$ or $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\bar{l}])) \geqslant 4$. Construct for the chosen literal $l$ two formulas

$$F_1 = Reduce(F[l]),$$
$$F_2 = Reduce(F[\bar{l}]).$$

If *Split* returns $True$ for at least one of these, then return $True$, otherwise return $False$.

(S7) (Splitting by any (3,3)-literal.) Choose any (3,3)-literal $w$ in the formula $F$. Choose a clause $C_1$ in the formula $F[w]$ such that

$$\exists l \in C_1(\text{l is a 2-literal}) \,\&\, \forall l' \in C_1(l' \text{ is not a (3,2)-literal}) \qquad (\star\star\star)$$

If there is such a clause that contains a (2,3)-literal, then choose this clause for $C_1$, otherwise choose any clause that satisfies the above condition. Choose a literal $l_1$ in $C_1$ such that

$$\mathfrak{K}(F[w]) - \mathfrak{K}(Reduce(F[w,l_1])) + \mathfrak{K}(F[w]) - \mathfrak{K}(Reduce(F[w,\overline{l}_1)]) \geqslant 7$$

and $(\star\star\star\star)$

$$\mathfrak{K}(F[w]) - \mathfrak{K}(Reduce(F[w,l_1])),\ \ \mathfrak{K}(F[w]) - \mathfrak{K}(Reduce(F[w,\overline{l}_1])) \geqslant 3.$$

In the same way choose a clause $C_2$ in the formula $F[\overline{w}]$ and a literal $l_2$ in it. Construct for the chosen literals $w$, $l_1$, $l_2$ four formulas

$$\begin{aligned}
F_{11} &= Reduce(F[w,l_1]),\\
F_{12} &= Reduce(F[w,\overline{l}_1]).\\
F_{21} &= Reduce(F[\overline{w},l_2]),\\
F_{22} &= Reduce(F[\overline{w},\overline{l}_2]).
\end{aligned}$$

If *Split* returns *True* for at least one of these, then return *True*, otherwise return *False*.

□

**Algorithm 4.1.**
*Input:* A formula $F$ in CNF.
*Output:* If the input formula $F$ is satisfiable, then *True*, otherwise *False*.
*Method:*

(M1) Return $Split(Reduce(F))$.

□

**Lemma 4.1.** *Function Reduce returns a formula that contains only (2,3+)- and (3+,3+)-literals.*

*Proof.* Function *Reduce* eliminates 1-literals and (2,2)-literals at step (R3). Pure literals are eliminated at step (R1).

□

**Lemma 4.2.** *Algorithm 4.1 always returns the correct answer.*

*Proof.* 1) First we should make sure that Algorithm 4.1 always returns an answer. For this, we should prove that at least one of the conditions of the steps (S1)–(S7) is always satisfied. The argument of Function *Split* is always a formula reduced by Function *Reduce*. Suppose that $F, G$ are formulas in CNF, and $F = Reduce(G)$. We now prove that $F$ satisfies at least one of the conditions of the steps (S1)–(S7). Suppose that the formula $F$ does not satisfy the conditions of the steps (S1)–(S3). Since it is reduced by Function *Reduce*, Lemma 4.1 says that in this case $F$ contains only (2,3+)-, (3+,2)- and (3,3)-literals. First let us prove that if the formula $F$ contains a clause $C = \{l_1, l_2, \ldots, l_s\}$ that contains only 2-literals, then we can choose literal $l$ in it such that $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[l]))$, $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\overline{l}]))\geqslant 3$ and

$\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[l])) + \mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\overline{l}])) \geqslant 7$, i.e. the condition of the step (S4) is satisfied. We now consider two cases.

CASE 1. $s = 2$.

Two clauses that contain $l_1$ dissapear in $F[l_1]$. At least one more clause is eliminated at step (R3) or at step (R1) since $l_2$ becomes a 1-literal or all its occurences dissapear. Thus, we have $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[l_1])) \geqslant 3$. On the other hand, three clauses that contain $\overline{l}_1$ dissapear in $F[l_1]$. Also the clause $C$ dissapears since it becomes a 1-clause. So $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\overline{l}_1])) \geqslant 4$.

CASE 2. $s \geqslant 3$.

For any literal $l \in C$, we have $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\overline{l}])) \geqslant 3$ since three clauses that contain $\overline{l}$ disappear in $F[l]$.

On the other hand, two clauses that contain $l_1$ dissapear in $F[l_1]$, and several clauses are eliminated at steps (R3) or (R1) since $l_2$ and $l_3$ become 1-literals or disappear. If at least two clauses are eliminated at these steps, then we have $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[l_1])) \geqslant 4$ that proves the statement. Otherwise all clauses that contain the literal $l_2$ contain also the the literal $l_3$. Thus, $\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[l_2])) \geqslant 5$ since after the elimination of clauses containing $l_2$, the literal $\overline{l}_3$ becomes pure and must be eliminated at step (R1).

Now, if the formula $F$ does not satisfy the conditions of the steps (S4)–(S5), then it satisfies the condition of the step (R5). But since the formula $F$ is reduced by Function *Reduce*, the step (R5) keeps it unchanged, i.e. it contains only (3,3)-literals.

We now prove that the formula $F$ reduced by Function *Reduce* that contains only (3,3)-literals and does not satisfy the condition of the step (S6), must satisfy the condition of the step (S7). For each literal $w$ of this formula,

$$\mathfrak{K}(F[w]) = \mathfrak{K}(Reduce(F[w])). \qquad (\star\star\star\star\star)$$

Hence, there are no pure literals, 1-literals and 1-clauses in $F[w]$. There are several 2-literals in the formula $F[w]$, these are all literals that occur in the clauses of the formula $F$ that contain $w$. If there is no clause $C_1$ in the formula $F[w]$ that satisfies the condition $(\star\star\star)$, then the formula $F[w]$ satisfies the condition of the step (R5). This contradicts $(\star\star\star\star\star)$. Thus, there exists clause $C_1$ in $F[w]$ that satisfies $(\star\star\star)$. We should prove that there exists a literal $l_1$ in $C_1$ that satisfies $(\star\star\star\star)$. Let us denote $\Delta_1 = \mathfrak{K}(F[w]) - \mathfrak{K}(F[w, l_1])$, $\Delta_2 = \mathfrak{K}(F[w]) - \mathfrak{K}([w, \overline{l}_1])$. Let $l$ be the literal guaranteed by the condition $(\star\star\star)$. We now consider several cases.

CASE 1. THE CLAUSE $C_1$ CONTAINS A (3,3)-LITERAL.

We can choose this literal for $l_1$. We have $\Delta_2 \geqslant 3$ since $l_1$ is a (3,3)-literal. On the other hand, at least three clauses containing $l_1$ dissapear in $F[w, l_1]$, and one more clause will be eliminated at step (R3) or at step (R1) since the literal $l$ becomes a 1-literal or disappears. Thus, $\Delta_1 \geqslant 4$, and the condition $(\star\star\star\star)$ is satisfied.

CASE 2. THE CLAUSE $C_1$ DOES NOT CONTAIN A (3,3)-LITERAL AND CONTAINS A (2,3)-LITERAL.

Let $C_1 = \{w_1, w_2, \ldots, w_s\}$, $w_1$ is a (2,3)-literal, where $w_2, \ldots, w_s$ are 2-literals. We choose the literal $w_1$ for $l_1$.

CASE 2.1. $s = 2$.

Three clauses of the formula $F[w]$ that contain $\overline{l}_1$ dissapear in $F[w, \overline{l}_1]$ since $l_1$ is a (2,3)-literal. In addition, the clause $C_1$ dissapears since it becomes a 1-clause. So we have $\Delta_2 \geqslant 4$. On the other hand, two clauses containing $l_1$ dissapear in $F[w, l_1]$, and one more clause will be eliminated at step (R3) or at step (R1) since the literal $w_2$ becomes a 1-literal or disappears. Thus, $\Delta_1 \geqslant 3$, and the condition $(\star\star\star\star)$ is satisfied.

CASE 2.2. $s \geqslant 3$.

We have $\Delta_2 \geqslant 3$ since $l_1$ is a (2,3)-literal. On the other hand, two clauses containing $l_1$ dissapear in $F[w, l_1]$, and two clauses will be eliminated at step (R3) or at step (R1) since the literals $w_2$ and $w_3$ become 1-literals or disappear. We note that the literals $w_2$ and $w_3$ cannot occur together in two clauses since $F$ does not satisfy the condition of the step (S6). Thus, $\Delta_1 \geqslant 4$, and the condition $(\star\star\star\star)$ is satisfied.

CASE 3. THE CLAUSE $C_1$ CONSISTS OF (2,2)-LITERALS.

Since the condition $(\star\star\star\star\star)$ is satisfied, in this case the formula $F[w]$ contains only clauses of two types: clauses that consist of (2,2)-literals and clauses that consist of (3,3)-literals. Let $C_1 = \{w_1, w_2, \ldots, w_s\}$, where $w_1, w_2, \ldots, w_s$ are 2-literals.

CASE 3.1. $s = 2$.

Two clauses of the formula $F[w]$ that contain $\overline{w}_1$ dissapear in $F[w, \overline{w}_1]$ since $w_1$ is a (2,2)-literal. In addition, the clause $C_1$ dissapears since it becomes a 1-clause. The second clause $C_1'$ that contains the literal $w_2$ dissapears together with $C_1$. If all these clauses are distinct, we choose $w_1$ for $l_1$ and have that $\Delta_2 \geqslant 4$. If they are not, then $C_1'$ contains $\overline{w}_1$. In this case we choose the literal $w_2$ for $l_1$. If we have the same problem with it, then there exists clause $C_1''$ in the formula $F[w]$ that contains together the literals $\overline{w}_2$ and $w_1$. Since the clauses $C_1'$ and $C_1''$ have no resolvent, we have that the number of clauses in $F[w]$ can be reduced at step (R3). This fact contradicts the condition $(\star\star\star\star\star)$. Thus, we can choose one of the literals $w_1$ and $w_2$ for $l_1$ and have that $\Delta_2 \geqslant 4$.

On the other hand, two clauses containing $l_1$ dissapear in $F[w, l_1]$, and one more clause will be eliminated at step (R3) or at step (R1) since the literal $l_2$ becomes a 1-literal or disappears. Thus, $\Delta_1 \geqslant 3$, and the condition $(\star\star\star\star)$ is satisfied.

CASE 3.2. $s \geqslant 3$.

We choose the literal $w_1$ for $l_1$. Two clauses of the formula $F[w]$ that contain $\overline{l}_1$ dissapear in $F[w, \overline{l}_1]$. In addition, these clauses contain only (2,2)-literals which become 1-literals or dissapear. These literals must be eliminated at step (R3) or at step (R1). Thus, $\Delta_2 \geqslant 3$. On the other hand, two clauses containing $l_1$ dissapear in $F[w, l_1]$, and two more clauses will be eliminated at step (R3) or at step (R1) since the literals $w_2$ and $w_3$ become 1-literals or disappear. We note that the literals $w_2$ and $w_3$ cannot occur together in two clauses since $F$ does not satisfy the condition of the step (S6). Thus, $\Delta_1 \geqslant 4$, and the condition $(\star\star\star\star)$ is satisfied.

We can choose the literal $l_2$ in the formula $F[\overline{w}]$ in the same way.

2) Let us prove that Function *Reduce* does not affect satisfiability of the formula $F$. This is evident for the steps (R1)–(R3). Hence, it is sufficient to prove this for the step (R5). Suppose $F$ satisfies the condition of the step (R5). Then each of its clauses is of one of the two types:

(1) it contains at least one (3+,2)-literal $l$;
(2) it contains only (3+,3+)-literals.

Let us consider arbitrary assignment $I$ that satisfies the formula $G = F[\{l \mid l \text{ is a } (3+,2)\text{-literal}\}]$. Since $G$ contains only (3+,3+)-literals, w.l.o.g. (2,3+)-literals of the formula $F$ does not occur in $I$. The assignment $I \cup \{l \mid \overline{l} \text{ is a 2-literal}\}$ satisfies $F$ since all clauses of $F \setminus G$ are of the type (1).

So we have proved that if $G$ is satisfiable, then $F$ is satisfiable too. The reverse is also true since $G \subset F$.

3) Now it is evident that the Algorithm 4.1 returns the correct answer since it always reduces satisfiability of the input formula to satisfiability of two formulas $Reduce(F[w])$ and $Reduce(F[\overline{w}])$ for some literal $w$, and these are satisfiable iff the corresponding formulas $F[w]$ and $F[\overline{w}]$ are satisfiable.
□

**Theorem 4.1.** *The worst case running time of Algorithm 4.1 is $p(\mathfrak{L}(F))\, 2^{0.30897\mathfrak{K}(F)}$, where $F$ is the input formula and $p$ is a suitable polynomial.*

*Proof.* We now construct a tree which reflects the behaviour of Algorithm 4.1. Its internal nodes are labelled with formulas that Algorithm 4.1 splits at steps (S3)–(S7). Its leaves are labelled with formulas that satisfy the conditions of the steps (S1)–(S2). Each internal node labelled with $F$ has two sons labelled with $F_1$ and $F_2$ or four sons labelled with $F_{11}$, $F_{12}$, $F_{21}$ and $F_{22}$. The running time of the algorithm has a polynomial dependence on the number of leaves in this tree. By Lemma 3.1 it is sufficient to prove that the branching number of this tree is not greater than $2^{0.30897}$. We now prove that all branching vectors of the nodes of this tree take the form $(3,4)$ or $(6,7,6,7)$. Since $\max(\tau((3,4)), \tau((6,7,6,7))) < \max(2^{0.2878}, 2^{0.30897}) < 2^{0.30897}$, this completes the proof. We denote $\Delta_1 = \mathfrak{K}(F) - \mathfrak{K}(F_1)$, $\Delta_2 = \mathfrak{K}(F) - \mathfrak{K}(F_2)$.

CASE 1. $F$ SATISFIES THE CONDITION OF THE STEP (S3).

Since $l$ is a (3+,4+)-literal, $\Delta_2$ is at least four, and $\Delta_1$ at least three. So the branching vector is $(3,4)$.

CASE 2. $F$ SATISFIES THE CONDITION OF THE STEP (S4).

The condition of this step guarantees that the branching vector is $(3,4)$.

CASE 3. $F$ SATISFIES THE CONDITION OF THE STEP (S5).

We have $\Delta_2 \geqslant 3$ since $l$ is a (3,3)-literal. On the other hand, at least three clauses containing $l$ dissapear in $F[l]$, and one more clause will be eliminated at step (R3) or at step (R1) since the literal $w$ becomes a 1-literal or disappears. Thus, $\Delta_1 \geqslant 4$, and the branching vector is $(3,4)$.

CASE 4. $F$ SATISFIES THE CONDITION OF THE STEP (S6).

Since $l$ is a (3,3)-literal, we have $\Delta_1, \Delta_2 \geqslant 3$. Thus, the condition of this step guarantees that the branching vector is $(3,4)$.

CASE 5. $F$ SATISFIES THE CONDITION OF THE STEP (S7).

Since $w$ is a (3,3)-literal, we have that $\mathfrak{K}(F) - \mathfrak{K}(F[w]) \geqslant 3$. The condition of the step (S7) guarantees that

$$\mathfrak{K}(F[w]) - \mathfrak{K}(Reduce(F[w,l_1])),\ \mathfrak{K}(F[w]) - \mathfrak{K}(Reduce(F[w,\overline{l}_1])) \geqslant 3$$

and

$$\mathfrak{K}(F[w]) - \mathfrak{K}(Reduce(F[w,l_1])) + \mathfrak{K}(F[w]) - \mathfrak{K}(Reduce(F[w,\overline{l}_1])) \geqslant 7.$$

Thus,

$$\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[w,l_1])),\ \mathfrak{K}(F) - \mathfrak{K}(Reduce(F[w,\overline{l}_1])) \geqslant 6$$

and

$$\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[w,l_1])) + \mathfrak{K}(F) - \mathfrak{K}(Reduce(F[w,\overline{l}_1])) \geqslant 13.$$

In the same way we have that

$$\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\overline{w},l_1])),\ \mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\overline{w},\overline{l}_1])) \geqslant 6$$

and

$$\mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\overline{w}, l_1])) + \mathfrak{K}(F) - \mathfrak{K}(Reduce(F[\overline{w}, \bar{l}_1])) \geqslant 13.$$

Thus, the branching vector is $(6, 7, 6, 7)$.

□

## References


[GJ] M. R. Garey, D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[D] E. Dantsin, *Tautology proof systems based on the splitting method*, Leningrad Division of Steklov Institute of Mathematics (LOMI) (1982), PhD Dissertation, Leningrad. (in Russian)

[DK] E. Ya. Dantsin, V. Ya. Kreinovich, *Exponential upper bounds for the satisfiability problem*, Proceedings of the IX USSR conference on mathematical logics (1988), Leningrad. (in Russian)

[DLL] M. Davis, G. Logemann, D. Loveland, *A machine program for theorem-proving*, Comm. ACM **5** (1962), 394–397.

[DP] M. Davis, H. Putnam, *A computing procedure for quantification theory*, J. Assoc. Comp. Mach. **7** (1960), 201–215.

[K] O. Kullmann, *Worst-case analysis, 3-SAT decision and lower bounds: approaches for improved SAT algorithms*, DIMACS Proceedings SAT Workshop 1996, American Mathematical Society, 1996.

[L] H. Luckhardt, *Obere Komplexitätsschranken für TAUT-Entscheidungen*, Proc. Frege Conference 1984, Schwerine, Akademie-Verlag Berline, 331–337.

[KL] O. Kullmann and H. Luckhardt, *Deciding propositional tautologies: Algorithms and their complexity*, submitted to Information and Computation, 1997.

[MS1] B. Monien, E. Speckenmeyer, *3-satisfiability is testable in* $O(1.62^r)$ *steps*, Bericht Nr. 3/1979, Reihe Theoretische Informatik, Universität-Gesamthochschule-Paderborn.

[MS2] B. Monien, E. Speckenmeyer, *Upper bounds for covering problems*, Bericht Nr. 7/1980, Reihe Theoretische Informatik, Universität-Gesamthochschule-Paderborn.

[MS3] B. Monien, E. Speckenmeyer, *Solving satisfiability in less then* $2^n$ *steps*, Discrete Applied Mathematics **10** (1985), 287–295.

[R] J. A. Robinson, *Generalized resolution principle*, Machine Intelligence **3** (1968), N.Y., 77–94.

[Sch1] I. Schiermeyer, *Solving 3-satisfiability in less than* $1.579^n$ *steps*, LNCS **702** (1993), 379–394.

[Sch2] I. Schiermeyer, *Pure literal look ahead: An* $O(1.497^n)$ *3-Satisfiability algorithm*, Workshop on the Satisfiability Problem Technical Report, Siena, April, 29 – May, 3, **1996**. (University Köln, Report No. **96-230**)