Discrete Applied Mathematics 10 (1985) 287–295
North-Holland

SOLVING SATISFIABILITY IN LESS THAN 2^n STEPS

B. MONIEN and E. SPECKENMEYER
Fachbereich Mathematik & Informatik, Universität Paderborn, Fed. Rep. Germany

Received 2 May 1983
Revised 12 April 1984

In this paper we describe and analyse an algorithm for solving the satisfiability problem. If E is a boolean formula in conjunctive normal form with n variables and r clauses, then we will show that this algorithm solves the satisfiability problem for formulas with at most k literals per clause in time $O(|F| \cdot \alpha_k^n)$, where α_k is the greatest number satisfying $\alpha_k = 2 - 1/\alpha_k^{k-1}$ (in the case of 3-satisfiability $\alpha_3 = 1{,}6181$).

1. Introduction

The problem whether a boolean formula F in conjunctive normal form (CNF) is satisfiable is known as the satisfiability problem and will be denoted by SAT. If the number of literals in each clause of a formula is restricted by k, the satisfiability problem for such formulas is denoted by k-SAT.

SAT and 3-SAT were the first problems, which have been shown to be NP-complete (see [1]). The related problem 2-SAT can be solved in linear time (see [5]). The NP-completeness of SAT and k-SAT, for $k \geq 3$, is a strong argument that there don't exist algorithms, which solve these problems in polynomial time. There has been done a lot of work in the past studying the behaviour of algorithms to solve SAT and k-SAT. Most of this work deals with the average case and with the probabilistic analysis of different versions of the Davis–Putnam procedure (DPP) under several instance distributions. In [3] and in [4] polynomial average case results for SAT are shown, where in [2] for some class of distributions a weaker form of DPP is shown to require exponential time with probability 1.

We will present in this paper an algorithm for SAT and analyse its worst case complexity. The algorithm is based upon branching techniques.

It is clear that SAT can be solved within time $O(|F| \cdot 2^n)$, where $|F|$ is the length of the encoding of the formula F and n the number of its variables. Our algorithm has a worst case complexity which is strictly better than 2^n. It solves k-SAT for $k \geq 3$ in time $O(|F| \cdot \alpha_k^n)$, where α_k is the greatest number satisfying $\alpha_k = 2 - 1/\alpha_k^{k-1}$. We get for example

$$\alpha_3 \approx 1{,}6181, \quad \alpha_4 \approx 1{,}8393, \quad \alpha_5 \approx 1{,}928, \quad \alpha_6 \approx 1{,}966.$$

We will use the following notions in this paper.

Let $V=\{a_1,\dots,a_n\}$ be a set of boolean variables and $L=\{a_1,\bar{a}_1,\dots,a_n,\bar{a}_n\}$ be the corresponding set of literals. If $L'\subseteq L$, then we denote

$$\mathrm{lit}(L'):=\{a,\bar{a}\,|\,a\in L' \text{ or } \bar{a}\in L'\}.$$

A disjunction $c=(x_1\vee\cdots\vee x_l)$ of literals $x_i\in L$ is called a clause. As usual we demand that a literal occurs at most once in a clause and a clause doesn't contain both, a literal x and its negation $\bar{x}$. We represent c by the set $\{x_1,\dots,x_l\}$ of literals in c. A conjunction $F=c_1\wedge\cdots\wedge c_r$ of clauses is called a formula in conjunctive normal form (CNF). We represent F by the set $\{c_1,\dots,c_r\}$ of its clauses. By $\square$ we denote the empty clause, which is unsatisfiable by definition and by $\emptyset$ the empty formula, which is a tautology by definition.

Let $k,n,r\in\mathbb{N}$. Then $K_k(n,r)$ denotes the set of all CNF-formulas F which contain at most n variables and r clauses for which each clause $c\in F$ has at most k literals. Set $K(n,r):=\bigcup_{k\in\mathbb{N}}K_k(n,r)$ and $K:=\bigcup_{n,r\in\mathbb{N}}K(n,r)$ is the set of all CNF-formulas.

A truth assignment $t:L\to\{\text{true},\text{false}\}$ satisfies a formula $F\in K$ iff

$$\forall c\in F:\ \exists x\in c:\ t(x)=\text{true}.$$

2. A branching algorithm for solving k-SAT

The algorithm which we will describe and analyse in this chapter is based upon branching at a clause as follows.

Let $F\in K_k(n,r)$ and $c=\{x_1,\dots,x_l\}\in F$, $l\le k$. If there is a satisfying truth assignment t for F, then particularly t must satisfy c. So we will split F into the following l formulas $F_1,\dots,F_l$, which are computed according to the following l truth assignments satisfying c.

$$\begin{aligned}
F_1:&\quad x_1=\text{ture},\\
F_2:&\quad x_1=\text{false},\ x_2=\text{true},\\
&\ \vdots\\
F_l:&\quad x_1=\cdots=x_{l-1}=\text{false},\ x_l=\text{true}.
\end{aligned}$$

Trivially F is satisfiable iff at least some F_i, $1\le i\le l$, is satisfiable. Moreover $F_1\in K_k(n-1,r-1)$, $F_2\in K_k(n-2,r-1),\dots,F_l\in K_k(n-l,r-1)$. Define by $T_k(F)$ the running time of the recursive algorithm based upon the above branching technique. Set

$$T_k(n,r):=\max\{T_k(F)\,|\,F\in K_k(n,r)\}.$$

Then $T_k(n,r)$ satisfies the recurrence relation

$$\begin{aligned}
&T_k(n,r)\le c && \text{for } n\le k,\\
&T_k(n,r)\le q(k,r)+\sum_{i=1}^{k}T_k(n-i,r-1) && \text{for } n>k.
\end{aligned}$$

where $q(k,r)$ is some polynomial.

It can be shown by induction on n that

$$T_k(n,r) = p(k,r)\,\beta_k^n, \quad \text{for some } \beta_k < 2, \text{ and some polynomial } p.$$

Here the smallest β_k which can be chosen is the greatest number satisfying $\beta_k = 2 - 1/\beta_k^k$. We get for example in the cases

$$\beta_3 \approx 1{,}8393, \quad \beta_4 \approx 1{,}928, \quad \beta_5 \approx 1{,}966.$$

In order to obtain an algorithm based upon this branching, which has the announced worst case running time, we have to do some extra work. We will use the notion of an autark truth assignment, which is defined as follows. Let $L' \subseteq L$ and t: $\mathrm{lit}(L') \to \{\mathrm{true}, \mathrm{false}\}$ be a truth assignment. Then t is called *autark* in F iff

$$\forall c \in F: \quad c \cap \mathrm{lit}(L') \neq \emptyset \to \exists x \in c \cap \mathrm{lit}(L'): t(x) = \mathrm{true}.$$

The importance of the notion autark for our approach is based upon the following consideration. If t: $\mathrm{lit}(L') \to \{\mathrm{true}, \mathrm{false}\}$ is autark in F, then all clauses $c \in F$ with $c \cap \mathrm{lit}(L') \neq \emptyset$ are simultaneously satisfied by t without having fixed any of the literals not in $\mathrm{lit}(L')$. If t is not autark in F, then there is a clause $c \in F$ such that $c \cap \mathrm{lit}(L') \neq \emptyset$ and $\forall x \in c \cap \mathrm{lit}(L')$: $t(x) = \mathrm{false}$. Thus the formula F', which is computed by evaluating F according to t contains the clause $c' = c - \mathrm{lit}(L')$ and $|c'| < |c|$.

The analysis of the algorithm presented below is based on the fact that when we evaluate a formula from $K_k(n,r)$ then every branching step which is preceeded also by a branching step leads to at most $k-1$ subproblems. In order to guarantee this we need the notion autark. We determine a shortest clause and investigate whether one of the truth assignments defined by this clause in the above sense is autark. If this is the case, then we can use this truth assignment and we don't have to branch in this step. Otherwise we branch and we know that all the formulas generated by this branching contain a clause of length at most $k-1$. The analysis will show that this behaviour is sufficient to guarantee our estimation. The constants α_k are chosen in such a way that α_k^n determines the growth of the sequence of generalized Fibonacci numbers $x_1, \ldots, x_n, \ldots$ which are defined by the recurrence relation $x_n = 2^n$ for $0 \leq n \leq k-2$ and $x_n = x_{n-1} + \cdots + x_{n-k+1}$ for $n \geq k-1$. Therefore the autarky tests make it possible to replace in the worst case estimation $\beta_k^n = \alpha_{k+1}^n$ by α_k^n. We have to pay for this improvement of the worst case bound by a slightly increased running time per branching step. Note that we do not investigate whether there exists some autark truth assignment but we look whether some given truth assignment is autark. The complexity of this test is of the same order than the complexity of the corresponding branching step.

Algorithm
Input $F \in K(n,r)$
Output. satisfiable, if F is satisfiable
unsatisfiable, otherwise
Procedure BSAT(F);

begin
1. **if** $F=\emptyset$ **then** return F satisfiable;
2. **if** $\square \in F$ **then** return F unsatisfiable;
3. determine a shortest clause $\{x_1, \ldots, x_l\} \in F$ and its length l;
4. notaut := true; $i := 1$;
   while $i \leq l$ **and** notaut **do**
   if $x_1 = \cdots = x_{i-1} =$ false, $x_i =$ true induces an autark truth assignment on $\mathrm{lit}(\{x_1, \ldots, x_i\})$
   then begin
   $F' := \{c \in F \mid c \cap \mathrm{lit}(\{x_1, \ldots, x_i\}) = \emptyset\}$;
   notaut := false;
   BSAT(F');
   if F' satisfiable **then** return F satisfiable
   else return F unsatisfiable
   end else $i := i+1$;
5. **if** notaut **then**
   begin
   unsat := true; $i := 1$;
   while $i \leq l$ **and** unsat **do**
   begin
   $F_i := \{c - \{x_1, \ldots, x_{i-1}, \bar{x}_i\} \mid c \in F,\ c \cap \{\bar{x}_1, \ldots, \bar{x}_{i-1}, x_i\} = \emptyset\}$;
   BSAT(F_i);
   if F_i satisfiable
   then begin
   unsat := false; return F satisfiable
   end else $i := i+1$;
   end;
   if $i = l+1$ **then** return F unsatisfiable
   end
end;

The meaning of the statements 1, 2 and 3 is clear. Now let $\{x_1, \ldots, x_l\}$ be the branching clause determined in 3. In statement 4 it is tested whether there is some $i, 1 \leq i \leq l$, such that $x_1 = \cdots = x_{i-1} =$ false, $x_i =$ true induces an autark truth assignment t for F. If there is such an i, then by definition of the notion autark every clause $c \in F$ containing some $x \in \mathrm{lit}(\{x_1, \ldots, x_i\})$ is satisfied by t. In this case the algorithm computes F' by deleting all such clauses from F and calls BSAT with F'.

Statement 5 will only be executed if during the preceeding execution of statement 4 none of the tested truth assignments had turned out to be autark. Then BSAT is called with F_i, computed in the same way as explained at the beginning of this chapter, for $i = 1, 2, \ldots$ in this order, until a first i is encountered such that F_i is satisfiable. In this case F is also satisfiable. Or none of the F_i, $1 \leq i \leq l$, is satisfiable, then F is unsatisfiable.

The correctness of this algorithm should be obvious.

The next lemma shows an important property of statement 5.

Lemma 1. *Let $F \in K_k(n,r)$. Then for all subproblems F_i generated in the while-loop of statement 5 there is a clause $c_i \in F_i$ with $|c_i| < k$.*

Proof. If statement 5 will be executed, then the following must have happened during the preceeding execution of statement 4. For all $1 \leq i \leq l$ the truth assignment t_i: lit$(\{x_1, \ldots, x_i\}) \to \{\text{true}, \text{false}\}$ induced by $x_1 = \cdots = x_{i-1} = \text{false}$, $x_i = \text{true}$ is not autark in F. Thus there is a clause $c \in F$ such that $c \cap \text{lit}(\{x_1, \ldots, x_i\}) \neq \emptyset$ and $\forall x \in c \cap \text{lit}(\{x_1, \ldots, x_i\})$: $t_i(x) = \text{false}$. Thus $c \cap \{\bar{x}_1, \ldots, \bar{x}_{i-1}, x_i\} = \emptyset$, and therefore $c_i := c - \{x_1, \ldots, x_{i-1}, \bar{x}_i\} \in F_i$, where c_i fulfills $|c_i| < k$. □

Lemma 2. *The execution time of* BSAT *between two successive calls is bounded by* $O(|F|)$, *when applied to* $F \in K$.

Proof. The bound certainly holds for the statements 1, 2 and 3. The execution time of statement 4 can also be bounded by $O(|F|)$, for if we have tested without success, whether $x_1 = \cdots x_{i-1} = \text{false}$, $x_i = \text{true}$ induces an autark truth assignment for F, then we can compute whether $x_1 = \cdots x_i = \text{false}$, $x_{i+1} = \text{true}$ induces an autark truth assignment for F by looking only at the clauses containing one of the literals $x_i, \bar{x}_i, x_{i+1}, \bar{x}_{i+1}$. This is true since we can mark during the preceeding computation all clauses which are fulfilled by setting $x_1 = \cdots = x_{i-1} = \text{false}$ and all clauses which contain some x_j, $1 \leq j \leq i-1$, but are not fulfilled by this setting.

Thus each literal in F is inspected at most twice and so the running time of statement 4 is bounded by $O(|F|)$. Statement 5 can be treated in a similar way. □

Note that the autarkness test in BSAT, though not needed for the correctness of the algorithm, requires nearly the same amount of time as the branching statement. Thus the autarkness test may double the running time of the algorithm in the worst case. However it is possible to combine both the autarkness and the branching statement by 'looking ahead' one subproblem. The estimated running time of the algorithm changed this way can the shown to be of the same order as that of BSAT, while its running time never exceeds the running time of BSAT without the autarkness test. On the other hand this can be achieved only for the price of using more memory, because we always have to remember two subproblems in the branching statement.

In a series of tests an implemented version of BSAT found autark truth assignments rarely and then nearly always in case of testing the first truth assignment $x_1 = \text{true}$. On the other hand tests have shown the number of recursive calls of our algorithm to be significantly smaller in the average compared with an algorithm based on the DPP-procedure in case of formulas with three literals per clause and $n \geq 20$ variables.

In order to compute the running time of the algorithm BSAT it remains to compute the number $T(F)$ of recursive calls BSAT has to perform to evaluate F. Define

$$T_k(n) := \max\{T(F) \mid F \in K_k(n,r)\} \quad \text{and}$$

$$T'_k(n) := \max\{T(F) \mid F \in K_k(n,r), \exists c \in F: |c| < k\}.$$

Then the following holds.

Lemma 3.

$$T_k(0) = T'_k(0) = 1,$$

$$T_k(n) \leq \max\left(\{T_k(n-i)+1 \mid 1 \leq i \leq k\} \cup \left\{1 + \sum_{i=1}^{j} T'_k(n-i) \mid 1 \leq j \leq k\right\} \cup \{1\}\right),$$

$$T'_k(n) \leq \max\left(\{T_k(n-i)+1 \mid 1 \leq i \leq k-1\} \cup \left\{1 + \sum_{i=1}^{j} T'_k(n-i) \mid 1 \leq j \leq k-1\right\} \cup \{1\}\right).$$

Proof. Let $F \in K_k(n,r)$. If $F = \emptyset$ or $\square \in F$, then $T(F) = 1$. Now let $\{x_1, \ldots, x_l\}$, $1 \leq l \leq k$, be a shortest clause determined by statement 3. If $\exists c \in F: |c| < k$, then $l \leq k-1$. Now assume that during the evaluation of statement 4 the truth assignment t for $\text{lit}(\{x_1, \ldots, x_i\})$ induced by $x_1 = \cdots = x_{i-1} = \text{false}$, $x_i = \text{true}$ is autark in F, $1 \leq i \leq l$. Then all clauses $c \in F$ with $c \cap \text{lit}(\{x_1, \ldots, x_i\}) \neq \emptyset$ are satisfied by t. So the resulting formula F' by which BSAT is called recursively belongs to the class $K_k(n-i, r-1)$ and therefore $T(F) = T(F') + 1 \leq T_k(n-i) + 1$.

If none of the above truth assignments is autark, then each generated subproblem F_i in the while-loop belongs to $K_k(n-i, r-1)$ and contains by Lemma 1 a clause $c_i \in F_i$ with $|c_i| < k$. This implies

$$T(F) \leq 1 + \sum_{i=1}^{l} T(F_i) \leq 1 + \sum_{i=1}^{l} T'_k(n-i). \qquad \square$$

In the following we only consider the case $k \geq 3$.

In order to compute a bound for $T_k(n)$ we introduce two functions $\varphi(n)$ and $\psi(n)$ bounding the functions $T_k(n)$ and $T'_k(n)$. Let φ and ψ be defined as follows

$$\begin{aligned}
&\varphi(0) = \psi(0) = 1,\\
&\varphi(n) = \psi(n) = 1 + \sum_{i=1}^{n} \psi(n-i) \quad \text{for } 1 \leq n \leq k-1,\\
&\varphi(n) = \max\left\{1 + \varphi(n-1), 1 + \sum_{i=1}^{k} \psi(n-i)\right\} \quad \text{for } n \geq k.\\
&\psi(n) = \max\left\{1 + \varphi(n-1), 1 + \sum_{i=1}^{k-1} \psi(n-i)\right\}
\end{aligned} \tag{2.1}$$

Obviously $\varphi(n)$ and $\psi(n)$ have the desired property. We will next show the following relation between φ and ψ.

Lemma 4. $\varphi(n) = 2\,\psi(n-1)$ *for* $n \geq 1$.

Proof. (By induction on n.) By a straightforward proof it can be shown that $\varphi(n) = \psi(n) = 2^n$ for $n \geq k-1$ and $\varphi(k) = 2^k$. So for $n \leq k$ the lemma holds. Now let $n > k$. Then

$$
\begin{aligned}
2\,\psi(n-1) &= \psi(n-1) + \psi(n-1) \\
&= \psi(n-1) + \max\left\{1 + \varphi(n-2), 1 + \sum_{i=1}^{k-1} \psi(n-1-i)\right\} \\
&= \psi(n-1) + \max\left\{1 + 2\,\psi(n-3), 1 + \sum_{i=1}^{k-1} \psi(n-1-i)\right\} \qquad \text{by ind. hyp.} \\
&= \psi(n-1) + \left(1 + \sum_{i=1}^{k-1} \psi(n-i-1)\right) \qquad \text{because } \psi(n) \text{ is increasing} \\
&= 1 + \sum_{i=1}^{k} \psi(n-i) \\
&= \max\left\{1 + 2\,\psi(n-2), 1 + \sum_{i=1}^{k} \psi(n-i)\right\} \qquad \text{because } \psi(n) \text{ is increasing} \\
&= \max\left\{1 + \varphi(n-1), 1 + \sum_{i=1}^{k} \psi(n-i)\right\} \qquad \text{by ind. hyp.} \\
&= \varphi(n). \qquad \square
\end{aligned}
$$

Note that $\psi(n) = 1 + \sum_{i=1}^{k-1} \psi(n-i)$, because by Lemma 4 $\varphi(n-1) = 2\,\psi(n-2)$ and $2\,\psi(n-2) \leq \sum_{i=1}^{k-1} \psi(n-i)$, because $\psi(n)$ is increasing.

By Lemma 4 and the above remark the function $\psi(n)$ is defined by

$$
\begin{aligned}
&\psi(0) = 1, \\
&\psi(n) = 1 + \sum_{i=1}^{n} \psi(n-i) \quad \text{for } n \leq k-1, \\
&\psi(n) = 1 + \sum_{i=1}^{k-1} \psi(n-i) \quad \text{for } n \geq k.
\end{aligned} \tag{2.2}
$$

Let $\hat{\psi}(n)$ be defined as follows:

$$
\begin{aligned}
&\hat{\psi}(0) = 1, \\
&\hat{\psi}(n) = \sum_{i=1}^{n} \hat{\psi}(n-i) \quad \text{for } n \leq k-1, \\
&\hat{\psi}(n) = \sum_{i=1}^{k-1} \hat{\psi}(n-i) \quad \text{for } n \geq k.
\end{aligned} \tag{2.3}
$$

Then the following holds.

Lemma 5. $\psi(n) < 3\,\hat{\psi}(n)$ *for all* $n \in \mathbb{N}$.

Proof. As mentioned above $\psi(n) = 2^n$, for $n \leq k-1$.

In the same way it can easily be checked that $\hat{\psi}(n) = 2^{n-1}$, for $1 \leq n \leq k-1$. So for $n \leq k-1$ the lemma holds. Now let $n \geq k$. Then

$$\begin{aligned}
\psi(n) &= 1 + \sum_{i=1}^{k-1} \psi(n-i) \\
&\leq 3\,\hat{\psi}(n-1) + \sum_{i=2}^{k-1} \psi(n-i) && \text{by ind. hyp. since } \psi(n-1) \leq 3\,\hat{\psi}(n-1) - 1 \\
&< 3\,\hat{\psi}(n-1) + 3 \sum_{i=2}^{k-1} \hat{\psi}(n-i) && \text{by ind. hyp. since } k \geq 3 \\
&= 3\,\hat{\psi}(n). \qquad \square
\end{aligned}$$

Lemma 6. $\hat{\psi}(n) \leq \alpha^n$ *for all* $n \in \mathbb{N}$, *where* α *is the greatest number satisfying* $\alpha = 2 - 1/\alpha^{k-1}$.

Proof. (By induction on n.) We will first prove the lemma for $0 \leq n \leq k$.

For $n = 0$ the bound trivially holds.

As mentioned above $\hat{\psi}(n) = 2^{n-1}$ for $1 \leq n \leq k-1$, and $\hat{\psi}(k) = 2^{k-1} - 1$.

So it is sufficient to show $2^{k-1} \leq \alpha^k$, because this implies

$$2^{k-1-l} \leq 2^{k-1}/\alpha^l \leq \alpha^{k-l} \quad \text{for } 0 \leq l \leq k-1.$$

Now

$$2^{k-1} = \left(\alpha + \frac{1}{\alpha^{k-1}}\right)^{k-1} = \alpha^{k-1}\left(1 + \frac{1}{\alpha^k}\right)^{k-1}.$$

We have to show:

$$\left(1 + \frac{1}{\alpha^k}\right)^{k-1} < \alpha.$$

This can be seen as follows. Because $x + 1/x^{k-1} < 2$ for $x = \frac{8}{5}$, we obtain $\frac{8}{5} < \alpha < 2$. That implies $(1 + 1/\alpha^k)^{k-1} < (1 + (\frac{5}{8})^k)^{k-1}$. The term $(1 + (\frac{5}{8})^k)^{k-1}$ is monotonously decreasing in k ($k \geq 3$), and $(1 + (\frac{5}{8})^3)^2 < \frac{8}{5}$. Thus

$$\left(1 + \frac{1}{\alpha^k}\right)^{k-1} < \tfrac{8}{5} < \alpha.$$

so we finally have shown

$$\hat{\psi}(k) < 2^{k-1} < \tfrac{8}{5}\,\alpha^{k-1} < \alpha^k.$$

Now let $n > k$. Then

$$\hat{\psi}(n) = \sum_{i=1}^{k-1} \hat{\psi}(n-i) \leq \sum_{i=1}^{k-1} \alpha^{n-i} = \alpha^{n-k+1} \sum_{i=0}^{k-2} \alpha^i = \alpha^{n-k+1}\,\frac{\alpha^{k-1} - 1}{\alpha - 1}.$$

It remains to show $(\alpha^{k-1}-1)/(\alpha-1)=\alpha^{k-1}$.

$$\alpha=2-\frac{1}{\alpha^{k-1}} \Rightarrow \alpha^k=2\alpha^{k-1}-1 \Rightarrow \alpha^{k-1}=\frac{\alpha^{k-1}-1}{\alpha-1}$$

So we have proved the lemma. □

Now we can estimate the number $T_k(n)$ of recursive calls of the algorithm BSAT:

$$T_k(n)\leq\varphi(n)=2\,\psi(n-1)\leq 6\,\hat{\psi}(n-1)\leq\frac{6}{\alpha}\,\alpha^n.$$

We get the final result of this chapter because of this estimation and because of Lemma 2.

Theorem. *The algorithm* BSAT *computes every formula* $F\in K_k(n,r)$ *in* $O(|F|\cdot\alpha^n)$ *steps, where* α *is the greatest number satisfying* $\alpha=2-1/\alpha^{k-1}$.

References

[1] S.A. Cook, The complexity of theorem-proving procedures, Proc. 3rd Ann. ACM Symp. on Theory of Computing (Assoc. Comput. Mach., New York, 1971) 151–158.

[2] J. Franco and M. Paull, Probabilistic analysis of the Davis–Putnam Procedure for solving the satisfiability problem, Discrete Appl. Math. 5 (1983) 77–87.

[3] A. Goldberg, Average case complexity of the satisfiability problem, Proc. 4th Workshop on Automated Deduction (Austin, TX, 1979) 1–6.

[4] A. Goldberg, P. Purdom and C. Brown, Average time analyses of simplified Davis–Putnam procedures, Inform. Process. Lett. 15 (2) (1982) 72–75.

[5] B. Monien and D. Janssen, Über die Komplexität der Fehlerdiagnose bei Systemen, Z. Ang. Mathem. Mech. (1977) T315–T317.