PDMI PREPRINT — 13/2002

A $2^{n/6.15}$-TIME ALGORITHM FOR X3SAT

Edward A. HIRSCH[1]
Sti.Petersburg Department of
Steklov Institute of Mathematics
27 Fontanka, 191011, St. Petersburg
Russia
E-mail: hirsch@pdmi.ras.ru

Alexander S. KULIKOV
St.Petersburg State University
Department of Mathematics and Mechanics
St.Petersburg
Russia
E-mail: ask@AK8436.spb.edu

June 28, 2002

Abstract

The exact 3-satisfiability problem (X3SAT) is: given a Boolean formula in 3-CNF, find a truth assignment, such that exactly one literal in each clause is set to true. It is well-known that X3SAT is NP-complete. In this paper, we present an exact algorithm solving X3SAT in time $O(2^{n/6.15})$, where n is the number of variables. This bound improves the previously known bound $O(2^{n/5.35})$ of [PRS02] (where also the bound $O(2^{n/6})$ was announced).

1

[1]The research described in this publication was made possible in part by Award No. RM1-2409-ST-02 of the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF). Also supported in part by RFBR grant No. 02-01-00089 and by grant No. 1 of the 6th RAS contest-expertise of young scientists projects (1999).

1 Introduction

General setting and our result. We consider the problem *X3SAT*: Given a formula in conjunctive normal form (CNF) where each clause contains at most three literals, find a truth assignment such that *exactly* one literal in each clause is set to true. It is well-known that X3SAT is NP-complete.

The best currently known bounds for XSAT (a more general problem without the restriction on the clause length) and X3SAT are $poly(|F|) \cdot 2^{n/4.09}$ [MSV81] and $O(2^{n/5.35})$ [PRS02] respectively ($poly(|F|)$ is a polynomial of the length of the input). By introducing a new simple transformation rule and distinguishing some new cases we prove a better $O(2^{n/6.15})$ bound for X3SAT. Note that contrary to 3SAT, for X3SAT we cannot eliminate not only "pure literals" but even variables occurring exactly once. Our new transformation rule eliminates the clauses containing two such variables.

Structure of our algorithm (and of the paper). Our algorithm is a typical splitting algorithm. That is, it uses a procedure that reduces the problem for a formula F to the problem for two formulas with smaller number of variables. More precisely, our algorithm constructs a tree by reducing exact satisfiability of the formula F to exact satisfiability of formulas $F[l]$ and $F[\bar{l}]$, where l is a literal of F (all necessary definitions are given in **Sect. 2**). Then, our algorithm simplifies each of the formulas $F[l]$ and $F[\bar{l}]$ according to transformation rules that do not affect the exact satisfiability of formulas. It is clear that the running time of our algorithm is within a polynomial factor of the number of leaves of the recursion tree. Essentially, it remains to list the transformation rules (see **Sect. 3**), describe the heuristic for choosing l and prove that the recurrence inequalities for the number of the leaves have "satisfactory" solutions (see **Sect. 4**).

Further directions. In the past two decades, many better and better exponential-time algorithms for combinatorial problems have been described (see, e.g., a survey [DHIV01] of (k-)SAT algorithms). A substantial part of these algorithms are splitting algorithms, and the algorithm of this paper is a typical one. An improvement to such algorithms is usually achieved by introducing more simplification rules (a *human* task) and distinguishing many combinatorial cases (a typical *computer* task). It would be interesting to design a computer program that, given (simple combinatorial) transformation rules, would output a *mechanically* proven upper bound corresponding to these rules. For example, one could investigate the affect of our new rule on the complexity of XSAT (or XkSAT) algorithms.

2 Background

Let V be a set of Boolean variables. The negation of a variable v is denoted by $\bar{v}$. *Literals* are variables and their negations. If l denotes a negated variable $\bar{v}$, then $\bar{l}$ denotes the variable v. We call two literals l_1 and l_2 *related* and write $l_1 \sim l_2$ if l_1 and l_2 correspond to the same variable (clearly, for each literal $l \sim \bar{l}$). We call a variable and a literal correspoding to it *unique* if this variable occurs in the formula only once. In addition to ordinary literals, we allow a special *true literal* $\mathcal{T}$, which always has the value *True* and does not correspond to any variable. (We also call it a $\mathcal{T}$*-literal* and do not denote it by other letters.) In our paper, we suppose that a clause can contain the $\mathcal{T}$*-literal*. A k*-clause* is a clause having *exactly* k literals. A *formula in CNF* (or simply *formula*) is a finite set of clauses, where no clause contains any variable more than once. The *length* of the formula is the total number of literals in it.

For a literal l and a formula F, the formula $F[l]$ is obtained by setting the value of l to *True*, i.e., by removing all occurences of $\bar{l}$ and by replacing all occurences of l by $\mathcal{T}$-*literal*. For two literals l_1 and l_2 corresponding to different variables the formula $F[l_1 = l_2]$ is obtained by replacing all occurrences of l_1 by l_2 and all occurrences of $\bar{l_1}$ by $\bar{l_2}$. Note that $F[l_1 = l_2]$ contains less variables than F.

3 Transformation Rules

A *correct* transformation rule replaces a formula F with a simpler formula F' such that $F \in$ X3SAT iff $F' \in$ X3SAT. We now list the transformation rules we use in our algorithm. Each transformation rule is executed in polynomial time and does not increase the number of variables or/and the number of clauses in the formula. Furthermore, each transformation rule reduces the length of formula. All rules, except Rule 4, are taken from [PRS02].

For each transformation rule we suppose that it is applied to formulas for which all previous rules cannot be applied, i.e., before applying any transformation rule our algorithm applies rules with smaller numbers as long as at least one of them is applicable.

Rule 0: Clause containing $\mathcal{T}$-*literal*. Suppose that F contains a clause C containing at least one $\mathcal{T}$-*literal*. We distinguish the following cases:

1. If $C = (\mathcal{T})$ then **Rule 0** replaces F with $F - \{C\}$.
2. If $C = (\mathcal{T}v)$ then **Rule 0** replaces F with $F[\bar{v}]$.
3. If $C = (\mathcal{T}vu)$, where $v \not\sim u$, then **Rule 0** replaces F with $F[\bar{v}, \bar{u}]$.
4. If $C = (\mathcal{T}vv)$ then **Rule 0** replaces F with $F[\bar{v}]$.
5. If $C = (\mathcal{T}\bar{v}v)$ then **Rule 0** replaces F with the empty clause (it means that F is not exactly satisfiable).
6. If C is one of the clauses $(\mathcal{T}\mathcal{T}), (\mathcal{T}\mathcal{T}\mathcal{T}), (\mathcal{T}\mathcal{T}v)$ then **Rule 0** replaces F with the empty clause.

Rule 1: Clause containing related literals. Suppose that F contains a clause C containing related literals. We distinguish the following cases:

1. If C is one of the clauses $(xx), (xxx), (\bar{x}xx)$ then **Rule 1** replaces F with the empty clause.
2. If $C = (x\bar{x})$ then **Rule 1** replaces F with $F - \{C\}$.
3. If $C = (xxy)$, where $x \not\sim y$ then **Rule 1** replaces F with $F[\bar{x}, y]$.
4. If $C = (x\bar{x}y)$, where $x \not\sim y$ then **Rule 1** replaces F with $F[\bar{y}]$.

Rule 2: Unit clause elimination. **Rule 2** eliminates 1-clauses, i.e., it replaces F with $F[v]$ if F contains the clause (v).

Rule 3: Binary clause elimination. **Rule 3** eliminates 2-clauses, i.e., it replaces F with $F[v = \bar{u}]$ if F contains the clause (vu).

Rule 4: Clause containing two unique literals. **Rule 4** replaces F with $F - \{C\}$ if C is a clause containing at least two unique literals.

Rule 5: Linked literals. Suppose that F contains clauses $(ax_1y_1), (ax_2y_2), (ax_3y_3)$ and also contains a clause C containing literals that are related to the literals x_1, x_2, x_3. We call the literals a, x_1, x_2, x_3 *linked*, if $C = (x_1x_2x_3)$, or $C = (\bar{x_1}\bar{x_2}x_3)$, or $C = (\bar{x_1}\bar{x_2}\bar{x_3})$. In such a case **Rule 4** replaces F with $F[\bar{a}]$.

Rule 6: Linked clauses. We call clauses C_1 and C_2 *linked* if they contain at least two common variables. We distinguish the following three cases:

1. If $C_1 = (xyz), C_2 = (xyt)$, then **Rule 6** replaces F with $F[z = t]$.
2. If $C_1 = (xyz), C_2 = (\bar{x}yt)$, then **Rule 6** replaces F with $F[\bar{y}, t = x, z = \bar{x}]$.
3. If $C_1 = (xyz), C_2 = (\bar{x}\bar{y}t)$, then **Rule 6** replaces F with $F[\bar{z}, \bar{t}, x = \bar{y}]$.

Rule 7: Small closed subformula. We can easily check whether $F' \in$ X3SAT if F' contains at most, say, 10 variables. Clearly, $F \in$ X3SAT iff $(F - F') \in$ X3SAT and $F' \in$ X3SAT. **Rule 7** replaces F with $(F - F')$ if F' is exactly satisfisfiable, and with the empty clause otherwise.

Rule 8: Simple formula. We call formula F *simple* if it has the property that every clause $C \in F$ that contains a variable occurring at least three times in F also contains a unique variable. We need only a polynomial time to check whether such a formula is exactly satisfiable or not (see [PRS02]). Thus, **Rule 8** replaces a simple formula F with the clause $(\mathcal{T})$ if F is exactly satisfiable, and with the empty clause otherwise.

Lemma 1. Transformation rules described above are correct.

Proof. It is easy to see the correctness of rules 0-7. The correctness of rule 8 is proved in [PRS02]. □

4 Main Algorithm

In this section, we present an algorithm which solves X3SAT in time $O(2^{n/6.15})$, where n is the number of variables of the input formula. We first present the algorithm and then estimate its running time and show its correctness using several lemmas.

Algorithm X3SAT

Input: A formula F in 3-CNF.
Output: If F is exactly satisfiable, then *True*, otherwise *False*.

Method

1. Apply **Rules 0–8** to F as long as at least one of them is applicable.

2. If F contains no more clauses, then return *True*.

3. If F contains the empty clause, then return *False*.

4. If F contains a variable occurring at least four times, then return $(\mathrm{X3SAT}(F[a]) \vee \mathrm{X3SAT}(F[\bar{a}]))$, where a is the literal corresponding to this variable.

5. If F contains clauses $(\bar{a}x_1y_1), (ax_2y_2), (ax_3y_3)$, then return $(\mathrm{X3SAT}(F[a]) \vee \mathrm{X3SAT}(F[\bar{a}]))$.

6. If F contains clauses $(ax_1y_1), (ax_2y_2), (ax_3y_3), (x_3t_1t_2), (p_1p_2z)$, where $p_1 \sim x_1, p_2 \sim x_2, z \sim x_3$ or $z \sim y_3$, and all the literals of first four clauses correspond to different variables, then return $(\mathrm{X3SAT}(F[x_3]) \vee \mathrm{X3SAT}(F[\bar{x_3}]))$.

7. If F contains clauses $(ax_1y_1), (ax_2y_2), (ax_3y_3)$, then return $(\mathrm{X3SAT}(F[a]) \vee \mathrm{X3SAT}(F[\bar{a}]))$.

After applying transformation rules to F, Algorithm X3SAT makes two recursive calls for formulas with smaller number of variables (unless F becomes trivial) in one of the steps 4–7.

Remark 1. Clearly, the total running time of the algorithm is the total running time of the two recursive calls plus time spent to make these calls. Therefore, the running time is within a polynomial factor of the number of leaves of the recursion tree. This number in its turn is determined by the maximum of the solutions of the recurrence inequalities corresponding to the nodes of the recursion tree (cf., e.g., [KL97]).

Lemma 2. At step 4 algorithm makes a splitting corresponding in the worst case to the recurrence inequality $T(n) \le T(n-9) + T(n-5)$ on the number of leaves of splitting tree.

Proof. Consider the formula F after the application of transformation rules. Clearly, F contains only 3-clauses, F does not contain linked clauses, clauses with two or three unique variables, and closed subformulas.

Suppose that F contains clauses $(\bar{a}x_1y_1), (\bar{a}x_2y_2), (ax_3y_3), (ax_4y_4)$. Note that after setting the value of a to *True* transformation rules change formula as follows: $x_1 = \bar{y_1}, x_2 = \bar{y_2}, x_3 = 0, y_3 = 0, x_4 = 0, y_4 = 0$. The obtained formula does not contain variables corresponding to the literals $a, x_1, x_2, x_3, y_3, x_4, y_4$ (note that since there are no linked clauses in the formula, these variables are all different). Hence, the obtained formula has at least 7 variables less than F. Similarly, so does the formula obtained by setting the value of a to $False$. Thus, we have the following recurrence inequality corresponding to this case: $T(n) \le T(n-7) + T(n-7)$. Similarly, we have the recurrence inequality $T(n) \le T(n-8) + T(n-6)$ in case F contains clauses $(\bar{a}x_1y_1), (ax_2y_2), (ax_3y_3), (ax_4y_4)$. In case F contains clauses $(ax_1y_1), (ax_2y_2), (ax_3y_3), (ax_4y_4)$ we have the following recurrence inequality: $T(n) \le T(n-9) + T(n-5)$. □

Note that if the condition of step 4 does not hold for the formula F then F does not contain variables appearing in it more than three times. Thus, in steps 5–7 the variable corresponding to the literal a appears in the formula only in the clauses considered in these steps.

Lemma 3. At step 5 algorithm makes a splitting corresponding in the worst case to the recurrence inequality $T(n) \le T(n-7) + T(n-6)$ on the number of leaves of splitting tree.

Proof. Suppose that F satisfies the condition of step 5. Note that formula does not contain clauses with two unique variables. Therefore, without lost of generality we can assume that x_1 is not unique.

Consider the formula $F[\bar{a}]$. Since the literal x_1 is not unique, the formula F has a clause containing a literal related to x_1. Let it be the clause $(x_1t_1t_2)$ (the analysis in case F contains the clause $(\bar{x_1}t_1t_2)$ is similar and much simpler). Apply all possible transformation rules to the formula $F[\bar{a}]$. Then the following substitutions are made: $x_1 = 0, y_1 = 0, x_2 = \bar{y_2}, x_3 = \bar{y_3}, t_1 = \bar{t_2}$. Denote the obtained formula by F_1. Now we prove that F_1 contain at least 6 variables less than F. In order to do this we have to show that the substitution $t_1 = \bar{t_2}$ reduces the number of variables, i.e., that this substitution is not equivalent to any of the substitutions $x_2 = \bar{y_2}, x_3 = \bar{y_3}$, and that at least one of the literals t_1, t_2 is equivalent to none of the literals a, x_1, y_1. Clearly, F_1 does not contain the variables corresponding to the literals a, x_1, y_1, x_2, x_3. Note that t_1 and t_2 are related to none of the literals a and y_1 (otherwise, F would contain linked clauses). It remains to show that it is impossible that $x_i \sim t_1$ and $y_i \sim t_2$ (where $i = 2$ or 3). However, this would imply that the corresponding clauses are linked.

Now consider the formula $F[a]$. We can suppose again that x_1, x_2 are not unique. The formula F contains a clause $(x_2t_3t_4)$ or a clause $(\bar{x_2}t_3t_4)$, where literals t_3 and t_4 are not related to the literals a and y_2. Assume that F contains the clause $(x_2t_3t_4)$ (the remaining case is similar). Apply all possible transformation rules to the formula $F[a]$. They change this formula as follows: $x_2 = 0, y_2 = 0, x_3 = 0, y_3 = 0, x_1 = \bar{y_1}, t_3 = \bar{t_4}$. Denote the obtained formula by F_2. Since F contains no linked clauses, at least one of the literals t_3 and t_4 is not related neither to x_1 nor to y_1. By the same reason it is not possible that each of the literals t_3 and t_4 is related to one of the literals x_2, y_2, x_3, y_3. Hence, setting $t_3 = \bar{t_4}$ eliminates one more variable from the formula. F_1 contains at least 7 variables less than F.

Thus, in this case we have the following recurrence inequality: $T(n) \le T(n-7) + T(n-6)$. □

Lemma 4. At step 6 algorithm makes a splitting corresponding in the worst case to the recurrence inequality $T(n) \le T(n-9) + T(n-4)$ on the number of leaves of splitting tree.

Proof. Consider the formula satisfying the condition of the step 6 after the application of transfomation rules. Let $C = (p_1p_2z)$.

Suppose that C contains a literal related to x_3. Since there are no linked literals in F and F does not satisfy the condition of step 5, $C = (\bar{x_1}x_2x_3)$ or $C = (x_1\bar{x_2}x_3)$. Without lost of generation assume that $C = (\bar{x_1}x_2x_3)$. The transformation rules change the formula $F[x_3]$ as follows: $x_2 = 0, x_1 = 1, a = 0, y_1 = 0, y_2 = 1, y_3 = 0, t_1 = 0, t_2 = 0$. The formula obtained after the application of transformation rules to $F[\bar{x_3}]$ contains at least 4 variables less than F. Therefore, we have the following recurrence inequality: $T(n) \le T(n-9) + T(n-4)$.

Now consider the case that C contains a literal related to y_3. We need to consider two cases: $C = (\bar{x_1}x_2y_3)$ or $C = (x_1x_2\bar{y_3})$ (the case $C = (x_1\bar{x_2}y_3)$ is similar, all other cases are impossible since there are no linked literals). Let $C = (\bar{x_1}x_2y_3)$. Thus, F contains clauses $(ax_1y_1), (ax_2y_2), (ax_3y_3), (x_3t_1t_2), (\bar{x_1}x_2y_3)$. The transformation rules change $F[x_3]$ as follows: $t_1 = 0, t_2 = 0, a = 0, y_3 = 0, y_1 = \bar{x_1}, y_2 = \bar{x_2}, x_1 = x_2$. Thus, the obtained formula contains at least 8 varaibles less than F. Then let us set $x_3 = 0$ and denote the obtained formula by F_1. Note that F_1 contains the clause (ay_3). Thus, Rule 3 replaces F_1 with $F_1[y_3 = \bar{a}]$, in particular it replaces the clause $(\bar{x_1}x_2y_3)$ with the clause $(\bar{x_1}x_2\bar{a})$. Clauses $(\bar{x_1}x_2\bar{a})$ and (ax_1y_1) are linked. Then, Rule 6 changes the formula as follows: $x_2 = 0, y_1 = 0, x_1 = \bar{a}$. Therefore, the transformation rules change the formula $F[\bar{x_3}]$ as follows: $t_1 = \bar{t_2}, x_2 = 0, y_1 = 0, y_3 = \bar{a}, x_1 = \bar{a}, y_2 = \bar{a}$. The obtained formula contains at least 7 variables less than F. Thus, we have the recurrence inequality $T(n) \leq T(n-8) + T(n-7)$. Similarly in case $C = (x_1 x_2 \bar{y_3})$ $F[x_3]$ contains at least 9 variables less than F ($a = 0, y_3 = 0, t_1 = 0, t_2 = 0, x_1 = 0, x_2 = 0, y_1 = 1, y_2 = 1$), and $F[\bar{x_3}]$ contains at least 5 variables less than F ($y_3 = \bar{a}, t_1 = \bar{t_2}, x_1 = \bar{a}, y_1 = x_2$). Therefore, we have the recurrence inequality $T(n) \leq T(n-9) + T(n-5)$ in this case. □

Lemma 5. At step 7 algorithm makes a splitting corresponding in the worst case to the recurrence inequality $T(n) \leq T(n-9) + T(n-4)$ on the number of leaves of splitting tree.

Proof. Consider the formula F containing clauses $(ax_1y_1), (ax_2y_2), (ax_3y_3)$ (similarly to previous lemmas we suppose that no transformation rule is applicable to F). Without lost of generality assume that the literals x_1, x_2, x_3 are not unique. Let us denote by S the set of literals l such that l is related to one of the literals $a, x_1, y_1, x_2, y_2, x_3, y_3$.

Suppose that F contains clause $(\bar{x}_i t_1 t_2)$, where $t_1 \notin S, t_2 \notin S$. Clearly, we have the recurrence inequality $T(n) \leq T(n-9) + T(n-4)$ in this case.

Now suppose that F contains clause $(x_i t_1 t_2)$, where $t_1 \notin S, t_2 \notin S$. Since F does not satisfy the condition of step 6, there is one more clause in F containing both a literal from S and a literal not from S (denote this literal by w). (Recall that x_1, x_2, x_3 are not unique and thus must form at least two such clauses; if instead they form just one clause of literals related to it, then $(x_i t_1 t_2)$ confirms the condition of step 6.) Clearly, the transformation rules remove w from the formula $F[a]$. Thus, in this case we also have the recurrence inequality $T(n) \leq T(n-9) + T(n-4)$.

Now consider the remaining case, i.e., that F does not contain a clause consisting of one literal from S and two literals not from S. Since the variables corresponding to the literals of set S do not form a closed subformula and since F does not satisfy the condition of step 6, F contains a clause (xyt), where $x \in S$, $y \in S$, $t \notin S$. The transformation rules change the formula $F[a]$ as follows: $x_1 = 0, y_1 = 0, x_2 = 0, y_2 = 0, x_3 = 0, y_3 = 0, t = 0$ or $t = 1$ (it depends on the literals x and y). Note that the variables corresponding to the literal t and to the literals of the set S also do not make a closed subfomula in F, hence one of these variables occurs in some other clause together with a different variable. Clearly, the transformation rules eliminate this new variable from $F[a]$. Therefore, the formula $F[a]$ contains at least 9 variables less than F. Thus, the recurrence inequality for this case is as follows: $T(n) \leq T(n-9) + T(n-4)$. □

Theorem 1. Given a formula F in 3-CNF containing n variables, algorithm X3SAT correctly checks whether F is exactly satisfiable, and gives its answer in time $O(2^{n/6.15})$.

Proof. Correctness. The correctness of transformation rules of our algorithm is shown in Lemma 1. After the application of transformation rules the formula F contains a variable occurring at least three times (otherwise the formula is simple). Let a be the literal corresponding to this variable. Clearly, F satisfies the condition either of the step 5 or of the step 7 of our algorithm (note that if F contains clauses $(\bar{a}x_1y_1), (\bar{a}x_2y_2), (ax_3y_3)$, then it satisfies the condition of the step 5, and if F contains clauses $(\bar{a}x_1y_1), (\bar{a}x_2y_2), (\bar{a}x_3y_3)$, then it satisfies the condition of the step 7).

Running time. The running time of our algorithm is within a polynomial factor of the number of leaves of the recursion tree which is the largest of the solutions of recurrence inequalities for the nodes of this tree (cf. Remark 1). The previous lemmas prove that in the worst case the number of leaves is the solution of $T(n) \leq T(n-9) + T(n-4)$, which is $2^{n/6.15249\ldots}$. □

References

[DHIV01] E. Dantsin, E. A. Hirsch, S. Ivanov, and M. Vsemirnov, Algorithms for SAT and Upper Bounds on Their Complexity. ECCC Technical Report 01-012, 2001. Electronic address: `ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/2001/TR01-012/index.html`. A Russian version appears in *Zapiski Nauchnykh Seminarov POMI*, vol. 277, pp. 14–46, 2001.

[KL97] O. Kullmann, and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. Manuscript, 1997. Available from: `http://cs-svr1.swan.ac.uk/~csoliver/`

[MSV81] B. Monien, E. Speckenmeyer, and O. Vornberger. Upper bounds for Covering Problems. *Methods of Operations Research*, 43:419–431, 1981.

[PRS02] S. Porschen, B. Randerath, and E. Speckenmeyer. X3SAT is decidable in time $O(2^{n/5})$. *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*, 231–235, 2002.