New Algorithms for Exact Satisfiability

Jesper Makholm Byskov
Bolette Ammitzbøll Madsen
Bjarke Skjernaa

BRICS Report Series RS-03-30

ISSN 0909-0878 October 2003

New Algorithms for Exact Satisfiability[*]

Jesper Makholm Byskov[†] Bolette Ammitzbøll Madsen
Bjarke Skjernaa[‡]
BRICS[§], Department of Computer Science
University of Aarhus, Denmark
{jespermn,bolette,skjernaa}@brics.dk

8th October 2003

Abstract

The Exact Satisfiability problem is to determine if a CNF-formula has a truth assignment satisfying exactly one literal in each clause; Exact 3-Satisfiability is the version in which each clause contains at most three literals. In this paper, we present algorithms for Exact Satisfiability and Exact 3-Satisfiability running in time $O(2^{0.2325n})$ and $O(2^{0.1379n})$, respectively. The previously best algorithms have running times $O(2^{0.2441n})$ for Exact Satisfiability (Monien, Speckenmeyer and Vornberger (1981)) and $O(2^{0.1626n})$ for Exact 3-Satisfiability (Kulikov and independently Porschen, Randerath and Speckenmeyer (2002)). We extend the case analyses of these papers and observe, that a formula not satisfying any of our cases has a small number of variables, for which we can try all possible truth assignments and for each such assignment solve the remaining part of the formula in polynomial time.

[*]Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

[†]Part of this research was done while visiting School of Information & Computer Science, University of California, Irvine.

[‡]Part of this research was done while visiting Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[§]Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

1 Introduction

The Exact Satisfiability (XSAT) problem is a variant of Satisfiability (SAT), where the difference is that in XSAT a clause is satisfied if *exactly* one of its literals is true. Exact 3-Satisfiability (X3SAT)[1] is the variant of XSAT in which each clause contains at most three literals. XSAT is NP-complete even when restricted to clauses containing at most three literals and all variables occurring only unnegated [8]. XSAT with all variables occurring at most twice can be solved in polynomial time [5][2].

XSAT can easily be solved in time $O(2^n)$ (we will ignore polynomial factors when stating running times, since these are all exponential) by enumerating all possible truth assignments to the n variables. In 1981, Schroeppel and Shamir were the first to give a faster algorithm. Their algorithm solves a class of problems of which XSAT and Knapsack are the most prominent in time $O(2^{n/2})$ and space $O(2^{n/4})$ [9]. The same year, Monien, Speckenmeyer and Vornberger [5] gave an algorithm solving only XSAT, but in time $O(2^{0.2441n})$[3] using only polynomial space. This is the previously best algorithm for XSAT.

X3SAT can of course be solved by an algorithm solving XSAT, but in recent years faster algorithms for X3SAT have been designed. The first was by Drori and Peleg [1] and runs in time $O(2^{0.2072n})$; several improvements followed and the previously best running time for X3SAT is $O(2^{0.1626n})$ and was achieved independently by Kulikov [2] and Porschen, Randerath and Speckenmeyer [6][4] in 2002.

Except for the algorithm by Schroeppel and Shamir [9], all the algorithms above are *branch-and-reduce* algorithms. A branch-and-reduce algorithm branches by making recursive calls on smaller formulas, such that the original formula is satisfiable if and only if at least one of the smaller formulas are satisfiable. In each branch, the algorithm reduces the formula by replacing it with another formula that is satisfiable if and only if the original formula is and that contains fewer variables. Fast branch-and-reduce algorithms rely on good decisions about what to

[1]Exact 3-Satisfiability is also called One-In-Three Satisfiability.

[2]They state that a generalised version of XSAT with variables occurring at most twice, called MAX($\{\leq, =, \geq\}, \cdot, 2$), reduces to Perfect Matching. The proof is in the technical report [4].

[3]Note that [5] only states the time $O(2^{n/4})$. The time complexity of $O(2^{0.2441n})$ is proved in the technical report [4].

[4]See Porschen, Randerath and Speckenmeyer [7] for an extended abstract.

branch on and good reduction rules.

In this paper, we present new branch-and-reduce algorithms for XSAT and X3SAT running in time $O(2^{0.2325n})$ and $O(2^{0.1379n})$, respectively. We introduce new reductions for both XSAT and X3SAT and improve the case analyses by a more careful analysis of the worst cases, which for some cases involves splitting them into more cases. Our main improvement, however, lies in our handling of *sparse* formulas: if the number of variables occurring at least three times in the formula is small, we can enumerate all possible truth assignments to these variables. For each assignment, the remaining formula contains only variables occurring at most twice, so we can decide in polynomial time, if it is satisfiable [5].

2 Preliminaries

2.1 Definitions

We are given a set of variables, which we will denote by using the letters x, y, z, w and u. A *literal* is either a variable x or the negation of a variable $\bar{x}$; we use $\tilde{x}$ to denote a literal that is either x or $\bar{x}$. A clause is a collection of literals, written as $(\tilde{x}_1, \ldots, \tilde{x}_l)$; we use the letter C to denote clauses. Sometimes, we will think of a clause as a set of literals (actually a multiset, since a clause can contain more than one of each literal); we use $(\tilde{x}, C)$ to denote a clause with $\tilde{x}$ and all the literals in C. A formula is a set of clauses usually written as $C_1 \wedge C_2 \wedge \cdots \wedge C_m$; we use the letter F to denote formulas. In intermediate steps of our algorithm we allow clauses to contain constants (*true* or *false*). The *size* of a formula is the number of literals and constants contained in the formula.

XSAT is the problem of given a formula F with m clauses over n variables to decide, if there exists an assignment to the variables, such that *exactly* one literal in each clause is true.

We let $V(F)$ denote the variables occurring in F. An (a, b)*-occurrence* is a variable occurring a times unnegated and b times negated or vice versa in F; a *unique* variable is a variable occurring only once. We will assume for simplicity, that when we look at a variable the first occurrence is unnegated.

We let $F[x \leftarrow y]$ where y is either a constant or a literal denote F with x replaced by y and $\bar{x}$ replaced by $\bar{y}$; similarly, we let $F[C \leftarrow \textit{false}]$ denote F with all literals in C replaced by *false* and their negations by *true*.

In X3SAT, a *cycle* is a list of clauses $(y_1, \tilde{z}_1, z_2)$, $(y_2, \tilde{z}_2, z_3)$, $\dots$, $(y_k, \tilde{z}_k, z_1)$ where neighbour clauses and the first and last clause share a variable and the z_i's are different variables.

2.2 Branching

Our algorithms make recursive calls on formulas with fewer variables. If C is a clause in the formula and $C' \subsetneq C$ then in a satisfying assignment for the formula either all literals in C' are *false* or exactly one is *true*. We use three different types of branches: branching on C' meaning that the recursive calls are on $C' \wedge F$ (in this case the formula can be reduced immediately afterwards, such that there will be fewer variables and clauses) and $F[C' \leftarrow \mathit{false}]$; we will denote the first branch "setting C' to *true*" and the second "setting C' to *false*". We can also branch on two variables x_1, x_2 meaning that the recursive calls are on $F[x_1 \leftarrow \mathit{true}]$, $F[x_2 \leftarrow \mathit{true}]$ and $F[x_1, x_2 \leftarrow \mathit{false}]$. Finally, we can branch on x_1; the two branches are then $F[x_1 \leftarrow \mathit{true}]$ and $F[x_1 \leftarrow \mathit{false}]$.

2.2.1 Sparse formulas

We call a formula *k-sparse*, if the number of variables occurring at least three times is at most n/k. To decide if a k-sparse formula is satisfiable, we enumerate all possible truth assignments to these n/k variables; for each assignment, all variables in the remaining part of the formula occur at most twice, so we can decide in polynomial time, if it is satisfiable. The total running time is $O(2^{n/k})$. We use this for XSAT with $k = 5$ and for X3SAT with $k = \frac{15}{2}$.

2.3 Branching vectors

In each branch of the algorithm, we remove a certain number of variables using the reductions; then we get a recursion for the running time of the form $T(n) = T(n - t_1) + T(n - t_2) + \cdots + T(n - t_k)$. We call $t = (t_1, t_2, \dots, t_k)$ the *branching vector* of this branch. The solution to the recursion is $T(n) = \alpha_t^n$, where α_t is the positive root of $1 - \frac{1}{x^{t_1}} - \frac{1}{x^{t_2}} - \cdots - \frac{1}{x^{t_k}}$; we call α_t the *value* of t and the value of a branching vector is decreasing as a function of the entries in the vector. Proofs of these results can be found in a manuscript by Kullmann and Luckhardt [3]. The logarithms of the values of all branching vectors occurring in this paper are either stated in Table 1 or are smaller than one of them by

Table 1: Branching vectors and the logarithms of their values (rounded)

(a) XSAT

t	$\log_2(\alpha_t)$	t	$\log_2(\alpha_t)$
$(12,1)$	0.2302	$(11,11,3)$	0.2216
$(8,2)$	0.2325	$(10,8,4)$	0.2325
$(6,3)$	0.2315	$(13,7,4)$	0.2258
$(5,4)$	0.2232		

(b) X3SAT

t	$\log_2(\alpha_t)$
$(12,4)$	0.1379
$(11,5)$	0.1317
$(9,6)$	0.1353
$(8,7)$	0.1336

monotonicity. The running time of the whole algorithm is $O(2^{\log_2 \alpha \cdot n})$, where α is the largest of the α_t's.

3 The algorithms

Both our algorithms have the following structure: first, the algorithm reduces the formula using the reductions from Section 3.1. If the reduced formula (we call a formula *reduced*, if none of the reductions from Section 3.1 is applicable) contains no clauses it is satisfied and if it contains the empty clause it can not be satisfied. If the formula only contains variables that occur at most twice in the formula, the algorithm decides in polynomial time if the formula is satisfiable; otherwise, the algorithm branches depending on whether the formula contains certain subformulas. The algorithm branches on the first matching case, which means that when it is in one case, no part of the formula matches any previous cases. The cases are described for XSAT in Section 3.2 and for X3SAT in Section 3.3. For simplicity, we ignore symmetries when this does not lead to confusion, so if we have two variables y_1 and y_2 or two clauses C_1 and C_2 that occur symmetrically and one of them has a certain property, we will just assume it is either.

3.1 Reductions

In this section, we present the reductions that are used in our algorithms; first, we present some common reductions used in both algorithms and then specific ones for the two algorithms. The reduction procedure for XSAT uses reductions (1) to (13) and the reduction procedure for X3SAT uses (1) to (8) and (14) and (15). The reductions are applied repeatedly top-down until no reduction applies. If any of the reductions either assign a variable both *true* and *false* or a constant is assigned its opposite value, the reduction procedures replace the whole formula with the empty clause.

When we branch, we call the algorithms recursively on smaller formulas. We show how to apply specific reductions to remove the stated number of variables. One can also show that applying the reductions top-down leads to the same or better branching vectors.

3.1.1 Common reductions

The common reductions are standard reductions also used by, e.g., Kulikov [2]. F denotes the entire left hand side of the reduction.

$$\begin{aligned}
(true, C) \wedge F' &\rightarrow F'[C \leftarrow \mathit{false}] && (1)\\
(false, C) \wedge F' &\rightarrow C \wedge F' && (2)\\
(x) \wedge F' &\rightarrow F'[x \leftarrow \mathit{true}] && (3)\\
(x, y) \wedge F' &\rightarrow F'[y \leftarrow \bar{x}] && (4)\\
(x, x, C) \wedge F' &\rightarrow F[x \leftarrow \mathit{false}] && (5)\\
(x, \bar{x}, C) \wedge F' &\rightarrow F'[C \leftarrow \mathit{false}] && (6)\\
(x, y, C) \wedge (x, \bar{y}, C') \wedge F' &\rightarrow F[x \leftarrow \mathit{false}] && (7)\\
(x, y, C) \wedge (\bar{x}, \bar{y}, C') \wedge F' &\rightarrow F[y \leftarrow \bar{x}] && (8)
\end{aligned}$$

3.1.2 Reductions for XSAT

Reduction (9) removes variables x that only occur unnegated and only in clauses with a unique variable or with literal y (and y is in no clauses without x). Note, that either k or l can be zero.

$$\underbrace{(x, y, C_1) \wedge \cdots \wedge (x, y, C_k) \wedge (x, u_1, C_1') \wedge \cdots \wedge (x, u_l, C_l') \wedge F'}_{x,y \notin V(F'),\ u_i \text{ unique}} \rightarrow F[x \leftarrow \mathit{false}] \qquad (9)$$

Reduction (9) is not used for X3SAT, as (14) or (15) can be used instead.

Reductions (10) and (11) are called resolution; resolution is a well-known technique for removing variables occurring both unnegated and negated and can also be used for solving SAT formulas. The idea is, that we can remove a variable x occurring both unnegated and negated and make all possible combinations of the clauses that contained x and the clauses that contained $\bar{x}$. If x is an (a,b)-occurrence, this will replace $a+b$ clauses with ab clauses, so we only use it for $(a,1)$- and $(2,2)$-occurrences.

$$\begin{aligned} (\bar{x}, C) \wedge \underset{x \notin V(F')}{(x, C_1)} \wedge \cdots \wedge (x, C_k) \wedge F' \;\rightarrow \\ (C, C_1) \wedge \cdots \wedge (C, C_k) \wedge F' \end{aligned} \tag{10}$$

$$\begin{aligned} (\bar{x}, C_1) \wedge \underset{x \notin V(F')}{(\bar{x}, C_2)} \wedge (x, C_1') \wedge (x, C_2') \wedge F' \;\rightarrow \\ (C_1, C_1') \wedge (C_1, C_2') \wedge (C_2, C_1') \wedge (C_2, C_2') \wedge F' \end{aligned} \tag{11}$$

Resolution is not used for X3SAT, as it creates clauses with more than three variables.

Reduction (12) removes clauses that have another clause as subset, and (13) reduces the formula if a clause shares all but one variable with another.

$$C \wedge \underset{C \subseteq C'}{C'} \wedge F' \;\rightarrow\; C \wedge F'[C' \setminus C \leftarrow \mathit{false}] \tag{12}$$

$$(x, C') \wedge (C, C') \wedge F' \;\rightarrow\; (x, C') \wedge (\bar{x}, C) \wedge F' \tag{13}$$

Reduction (12) and (13) are not used for X3SAT as (14) handles the same cases when the clauses have size three.

Lemma 1. *In a reduced* XSAT *formula, for all pairs of clauses, each has at least two variables that do not occur in the other.*

Proof. All clauses contain at least three literals by (3) and (4), so we only need to consider clauses having at least two variables in common. Common variables must occur the same (unnegated or negated) by (7) and (8). No clause is a subset of another by (12), and for any pair of clauses both have at least two literals that do not occur in the other by (13), so the lemma is true. □

3.1.3 Reductions for X3SAT

Reduction (14) is also a standard reduction, but we only use it for X3SAT. Reduction (15) reduces formulas containing two variables that only occur unnegated and only in clauses with a unique variable, except for one clause, where one occurs unnegated and the other negated. Note, that both variables can be unique themselves.

$$(x, y, z_1) \wedge (x, y, z_2) \wedge F' \;\rightarrow\; (x, y, z_1) \wedge F'[z_2 \leftarrow z_1] \tag{14}$$

$$\underset{\substack{x_1 \text{ and } x_2 \text{ only occur unnegated and} \\ \text{in clauses with a unique variable in } F'}}{(\bar{x}_1, x_2, y) \wedge F'} \;\rightarrow\; F[x_2 \leftarrow \mathit{false}] \tag{15}$$

Remark. A reduced X3SAT formula contains no constants or 1- or 2-clauses by (1) to (4) and no two clauses have more than one variable in common by (7), (8) and (14); also, no clause has more than one unique variable and all $(a, 0)$- and $(a, 1)$-occurrences that are not unique are in a clause with no unique variables by (15).

3.1.4 Soundness and complexity

The following lemma states that the reductions are *sound*, that is, the reduced formula is satisfiable if and only if the original formula is.

Lemma 2. *Reductions (1) to (15) are sound.*

Proof. To prove that (1) to (8) and (10) to (14) are sound we just note, that exactly one literal from a clause must be *true* and exactly one of x and $\bar{x}$ must be *true*.

In (9), x can be assumed to be *false*, as a satisfying assignment with x *true* can be changed to a satisfying assignment with x *false* by making y and the unique variables *true* instead. Similarly in (15), a satisfying assignment with x_2 *true* must have x_1 *true*, and it can be changed by setting x_1 and x_2 *false* and all the unique variables *true*. □

The next two lemmas show, that during the reduction procedures the size of the formula is never larger than the maximum of the size of the original formula and $2mn$. We use this to show, that the reduction procedures run in polynomial time in the size of the formula.

Lemma 3. *When the reduction procedures run on a formula F with m clauses and n variables the intermediate formulas are never larger than* $\max(|F|, 2mn)$.

Proof. For X3SAT it is obvious that the size of the formula is never larger than $\max(|F|, 3m)$.

For XSAT, none of the reductions increases the number of variables or clauses in the formula. The reduction procedure first applies reductions (1) to (6), which all decrease the size of the formula. After it has run, the formula contains no constants and no variable occurs more than once in the same clause; thus, the size of the formula is at most mn.

The only reductions which can make the formula larger are resolution ((10) and (11)), but the number of literals in a clause after resolution is still bounded by $2n$ and will be reduced to n before we perform resolution again; thus, after the first applications of (1) to (6), the size of the formula is always bounded by $2mn$. □

Lemma 4. *The reduction procedures run in polynomial time in the size of the formula.*

Proof. For each reduction, the algorithm can in polynomial time in the size of the formula check whether it is applicable and if so apply it.

Resolution ((10) and (11)) removes a variable from the formula, so they can be applied at most n times, since no reduction add variables. All the other reductions reduce the size of the formula (for (13) just note that C' has size at least two by (4)). Since the size of the formula is at most $\max(|F|, 2mn)$, the reduction procedures run in polynomial time in $|F|$. $\square$

3.2 The algorithm for XSAT

In this section, we present our algorithm for XSAT and show that it achieves a branching vector of $(8, 2)$ corresponding to a running time of $O(2^{0.2325n})$. The previously best algorithm is by Monien et al. [5] and has worst case branching vector $(11, 1)$ corresponding to a running time of $O(2^{0.2441n})$.

3.2.1 Variables occurring both unnegated and negated

If the formula F contains a variable occurring both unnegated and negated, it must occur at least three times unnegated and twice negated or vice versa; otherwise, it would have been removed by resolution ((10) or (11)). Let x be the corresponding literal occurring at least three times unnegated and twice negated as in the clauses in Figure 1. The

$$\begin{array}{llll}
(x, C_1) & (\bar{x}, C_1') & \quad & x = \mathit{true} : C_1 = C_2 = C_3 = \mathit{false} \\
(x, C_2) & (\bar{x}, C_2') & \quad & x = \mathit{false} : C_1' = C_2' = \mathit{false} \\
(x, C_3) & & &
\end{array}$$

Figure 1: Branching on at least a $(3, 2)$-occurrence x.

algorithm branches on x. By Lemma 1 and a simple counting argument, C_1, C_2 and C_3 must contain at least six different variables in total and C_1' and C_2' at least four; thus, branching on x yields a branching vector of at least $(7, 5)$.

3.2.2 Two clauses having at least two variables in common

Suppose F contains two clauses having more than one variable in common as in Figure 2, with C_1 and C_2 not having any variables in common and $|C| \geq 2$. By Lemma 1, also $|C_1|, |C_2| \geq 2$. If the two clauses are not

$$\boxed{\begin{array}{lrl} (C, C_1) & C = \textit{true}: & C_1 = C_2 = \textit{false}, \text{ this removes } |C_1| + |C_2| \\ (C, C_2) & & \text{variables plus one if } |C| = 2 \text{ (by (4))} \\ & C = \textit{false}: & \text{this removes } |C| \text{ variables plus one} \\ & & \text{for each } C_i \text{ with } |C_i| = 2 \text{ (by (4))} \end{array}}$$

Figure 2: Two clauses having at least two variables in common.

two 5-clauses having exactly two variables in common, the algorithm branches on C as shown in the figure. Since C, C_1 and C_2 all have size at least two, this removes at least four variables when C is set to *true* and two when C is set to *false*. Now, if any of the clauses are 2-clauses this removes one extra variable in one branch and if they are at least 3-clauses we remove at least one more in the other branch. All in all we get branching vectors of at least $(5, 4)$, $(6, 3)$ or $(7, 2)$, but $(7, 2)$ only if we had two 5-clauses having two variables in common. Having excluded that case, which we deal with later, we have $(8, 2)$ as the worst case.

3.2.3 Variables occurring at least four times

If F contains a variable x occurring at least four times and either with at least eleven different variables or in a 3-clause, the algorithm branches on x. If it occurs with eleven different variables, this yields a branching vector of at least $(12, 1)$ and if x is in a 3-clause, we get a branching vector of at least $(9, 2)$, since x must occur with at least eight different variables by Lemma 1.

If F contains a variable x occurring at least four times that does not satisfy the previous case, we pick four of the clauses containing x; since the only clauses having more than one variable in common are 5-clauses having two variables in common, these must be 5-clauses pairwise having two variables in common as in Figure 3. Then the algorithm branches on (x, y_1). When it is set to *true*, y_1 is removed by (4) and the six other variables in the first two clauses are set to *false* by (6). The last two clauses reduce to (x, y_2, z_3) and (x, y_2, z_4), but then $z_3 = z_4$ by (13) and (4). So we get a branching vector of at least $(8, 2)$.

(x, y_1, y_3, y_5, z_1)	$(x, y_1) = \mathit{true} : y_3 = y_4 = y_5 = y_6 = z_1 = z_2 = \mathit{false},$
(x, y_1, y_4, y_6, z_2)	$y_1 = \bar{x}, z_3 \overset{(13)}{=} z_4$
(x, y_2, y_3, y_6, z_3)	
(x, y_2, y_4, y_5, z_4)	

Figure 3: A $(4, 0)$-occurrence x with only ten variables.

Lemma 5. *If a reduced formula does not satisfy any of the previous cases and it contains two variables x and y that occur in a clause together and they both also occur in a clause without the other, setting (x, y) to true removes both x and y.*

Proof. When the algorithm sets (x, y) to *true* then $y = \bar{x}$ by (4) and this makes x a $(1, 1)$-, $(2, 1)$- or $(2, 2)$-occurrence, which we remove by resolution ((10) or (11)). □

3.2.4 Two 5-clauses having exactly two variables in common

Suppose F contains two 5-clauses having two variables in common as the first two clauses in Figure 4(a). If both x_1 and x_2 occur in a clause

$(x_1, x_2, y_1, y_2, y_3)$	$(x_1, x_2, y_1, y_2, y_3)$
$(x_1, x_2, y_4, y_5, y_6)$	$(x_1, x_2, y_4, y_5, y_6)$
(x_1, y_1, C')	(x_1, z_1, z_2, z_3)
(a) The third clause contains y_1.	(b) The third clause contains no y's.

Figure 4: Two 5-clauses having two variables in common.

without the other, the algorithm branches on (x_1, x_2). Setting it to *true* removes all eight variables in the two clauses by Lemma 5, so this yields a branching vector of at least $(8, 2)$.

By (9), at least one of x_1 and x_2 must occur in another clause, so assume we have another clause with x_1. Then x_1 is a $(3, 0)$-occurrence and occurs in no other clauses. The third clause with x_1 can have at most one variable in common with each of the first two clauses apart from x_1 and if there is such a variable, the third clause must be a 5-clause.

The third clause contains $\mathbf{y_1}$ as in Figure 4(a). Now, C' contains three variables, one of which can be y_4, y_5 or y_6. If neither y_2 nor y_3

are unique, the algorithm branches on (y_2, y_3); setting it to *true* removes both y_2 and y_3 by Lemma 5, and x_1, x_2 and y_1 and setting (y_2, y_3) to *false* removes y_2 and y_3, so we have the clauses in Figure 5; then we apply the reductions shown in the figure and remove x_2 and y_1. In total, we get a

$$\begin{array}{lclcl}
(x_1, x_2, y_1) & & (x_1, x_2, y_1) & & \\
(x_1, x_2, y_4, y_5, y_6) & \overset{2\times(13)}{\longrightarrow} & (\bar{y}_1, y_4, y_5, y_6) & \overset{2\times(10)}{\longrightarrow} & (x_1, y_4, y_5, y_6, C') \\
(x_1, y_1, C') & & (\bar{x}_2, C') & &
\end{array}$$

Figure 5: The clauses from Figure 4(a) when $y_2 = y_3 =$ *false*.

branching vector of at least $(5, 4)$.

In the other case, one of y_2 and y_3 (say y_3) is unique, but not y_2 by (9). Now, y_1 occurs in no other clauses; otherwise, x_1 and y_1 are two variables occurring twice together and both in a clause without the other, which is the first case above. The algorithm branches on y_2. Setting it to *true* removes at least seven variables, since y_2 occurs in another clause. Setting y_2 to *false* leaves the three clauses in Figure 6, where y_3 is unique, and x_1, x_2 and y_1 only occur in these three clauses; then we replace those three

$$\begin{array}{lcl}
(x_1, x_2, y_1, y_3) & & (x_2, y_4, y_5, y_6) \\
(x_1, x_2, y_4, y_5, y_6) & \longrightarrow & (y_1, C') \\
(x_1, y_1, C') & &
\end{array}$$

Figure 6: Equivalent clauses.

clauses by the two clauses on the right: any truth assignment satisfying the original formula with x_1 *true* will satisfy the new formula if y_1 and x_2 are both changed to *true* and a satisfying assignment with x_1 *false* will also satisfy the new formula. On the other hand, a satisfying truth assignment to the new formula with both x_2 and y_1 *true* is a satisfying assignment to the original formula if x_1 is set to *true* and x_2, y_1 and y_3 are set to *false*. All other assignments satisfying the new formula can be changed to satisfy the original one by setting x_1 to *false* and choosing y_3 such that (x_2, y_1, y_3) is satisfied. By doing this replacement, both x_1 and y_3 are removed, so we get a branching vector of at least $(7, 3)$.

The third clause with $\mathbf{x_1}$ contains none of the y's. If the third clause is not a 4-clause, the algorithm branches on x_1. If the third clause is a 3-clause, this yields a branching vector of at least $(10, 2)$ and if the third clause is at least a 5-clause, this yields a branching vector of at least $(12, 1)$.

The remaining case is depicted in Figure 4(b); none of the z_i's is unique by (9), since the two other clauses with x_1 contains x_2, which is in no other clauses. If one of z_1, z_2 and z_3 occurs three times, say z_1, the algorithm branches on x_1, z_1. Setting x_1 to *true* removes eleven variables, setting z_1 to *true* removes at least eight and setting both to *false* removes all four variables in the third clause by Lemma 5. So we get a branching vector of at least $(11, 8, 4)$.

In the last case, z_1, z_2 and z_3 all occur exactly twice in F; then the algorithm branches on (x_1, z_1). All four variables in the third clause are removed in both branches by Lemma 5. Let the other clause containing z_1 be (z_1, C'). If it is a 3-clause, we remove an extra variable setting z_1 to *false*. If the clause contains one of the y's, we get a clause where this y occurs twice, when we set (x_1, z_1) to *true* and the variable is removed by (5). Both cases result in a branching vector of at least $(5, 4)$. Otherwise, (z_1, C') is at least a 4-clause containing none of the y's, but in that case when we set (x_1, z_1) to *true* and apply (10), we get the clauses (C', x_2, y_1, y_2, y_3) and (C', x_2, y_4, y_5, y_6). These two clauses contain at least seven variables each and have all but three of them in common. We then further branch on (C', x_2) and get a branching vector of at least $(6, 4)$ (see Figure 2). In total, we get a branching vector of at least $(10, 8, 4)$ ($(4, 4)$ followed by $(6, 4)$ in one branch).

3.2.5 Variables occurring three times

If F contains a variable occurring three times and not in three 4-clauses or two 4-clauses and a 5-clause, the algorithm branches on it. This yields a branching vector of at least $(7, 4)$, $(8, 3)$, $(9, 2)$ or $(12, 1)$ depending on the number of 3-clauses the variable is in.

A $(3, 0)$-occurrence in three 4-clauses. Suppose F contains a $(3, 0)$-occurrence x, which is in three 4-clauses as in Figure 7(a). Not all the clauses can contain a unique variable or F would have been reduced by (9), so assume that the first clause contains no unique variables. If y_1 occurs with at least four other variables, we branch on x, y_1. Setting x to *true* removes ten variables, setting y_1 to *true* removes at least eight and setting both to *false* removes four by Lemma 5. In total, we get a branching vector of at least $(10, 8, 4)$.

(x, y_1, y_2, y_3)	(x, y_1, y_2, y_3)	(y_1, z_1, z_2, z_3)
(x, y_4, y_5, y_6)	(x, y_4, y_5, y_6)	(y_2, z_4, z_5, z_6)
(x, y_7, y_8, y_9)	(x, y_7, y_8, y_9)	(y_3, z_7, z_8, z_9)

(a) A $(3, 0)$-occurrence in three 4-clauses.

(b) The variables y_1, y_2 and y_3 are all $(2, 0)$-occurrences.

Figure 7: A $(3, 0)$-occurrence x in three 4-clauses.

We can assume now, that y_1, y_2 and y_3 are all $(2, 0)$-occurrences and that their other clause is at most a 4-clause; otherwise, we have the previous case. If y_1 is in a 3-clause, the algorithm branches on (x, y_1); in both branches all variables in the clause with x and y_1 are removed by Lemma 5. Setting (x, y_1) to *false* also removes one of the other variables from the 3-clause by (4). This yields a branching vector of at least $(5, 4)$.

If none of the previous cases applies, the other clauses containing y_1, y_2 and y_3 must be the ones in Figure 7(b), where some of the z's can be one of y_4 up to y_9 and some of them can be the same variable. By (9), at most one z_i from each clause is unique. Suppose two z_i's from different clauses (say z_1 and z_4) are unique; then branching on (y_1, y_2) will remove x, y_1, y_2 and y_3 in both branches and setting (y_1, y_2) to *true* also makes z_1 and z_4 end up in the same clause and one is removed by (9). This also yields a branching vector of at least $(5, 4)$.

We can assume now, that say z_1, z_2 and z_3 are not unique and if any of them are a y, then z_1 is y_4. The algorithm branches on (y_1, z_1); in both branches y_1, z_1, z_2 and z_3 are removed. If z_1 was y_4, setting (y_1, z_1) to *true* makes $y_4 = \bar{y}_1$, so x is *false* by (7). This yields a branching vector of at least $(5, 4)$. If z_1 is not y_4, setting (y_1, z_1) to *false* reduces the first clause with x to (x, y_2, y_3). The algorithm then branches on x, which yields a branching vector of at least $(9, 3)$, since when x is set to *false*, y_2 and y_3 are removed by Lemma 5. In total, this yields a branching vector of at least $(13, 7, 4)$.

A $(3, 0)$-occurrence in two 4-clauses and a 5-clause. We have now removed all $(3, 0)$-occurrences except those in two 4-clauses and one 5-clause. If we have such a variable x and one of the 4-clauses contains another $(3, 0)$-occurrence y_1 we branch on x, y_1. Setting one of the $(3, 0)$-occurrences to *true* removes eleven variables and setting both to *false* removes three, so we get a branching vector of at least $(11, 11, 3)$.

If we have a $(3, 0)$-occurrence x in two 4-clauses and a 5-clause and one of the 4-clauses contains only $(2,0)$-occurrences other than x, we branch as in the previous section. The only difference having a 5-clause instead of a 4-clause makes, is that setting x to *true* removes an extra variable.

Sparse formulas. The only remaining case with variables occurring more than twice is a $(3,0)$-occurrence in two 4-clauses and one 5-clause, where both 4-clauses contain a unique variable and the other variables in the 4-clauses occur twice in F. Then F is 5-sparse, i.e., it contains at least four variables occurring at most twice for each variable occurring three times: we only count the variables in the 4-clauses. Each variable occurring three times occurs with two unique variables and four $(2,0)$-occurrences in the two 4-clauses. The $(2,0)$-occurrences might be in another clause with a variable occurring three times, but if this clause is a 4-clause it can contain at most one $(3,0)$-occurrence. Thus, we have at least two $(2,0)$-occurrences and two unique variables for each variable occurring at least three times, so the formula is 5-sparse and the algorithm solves the remaining formula in time $O(2^{n/5})$, where n is the number of remaining variables.

3.3 The algorithm for X3SAT

In this section, we give our algorithm for X3SAT and show that it achieves a branching vector of $(12,4)$ corresponding to a running time of $O(2^{0.1379n})$. The previously best algorithms are by Kulikov [2] and Porschen, Randerath and Speckenmeyer [6] and have branching vector $(9,4)$ corresponding to a running time of $O(2^{0.1626n})$.

Extra reductions For X3SAT we have some extra reductions which are only needed to remove certain cycles. We do not use them in the reduction procedure, but rather apply them, when they are needed.

We are concerned with cycles because, if we have k clauses and a variable in each is set to *false*, we would normally remove another variable from each of the remaining 2-clauses by (4), but if some of the clauses form a cycle on the variables not set to *false*, we may remove one less variable. As an example, $F[z_1 \leftarrow z_2, z_2 \leftarrow z_3, \ldots, z_{k-1} \leftarrow z_k, z_k \leftarrow z_1]$ only removes $k-1$ variables from F.

Reductions (19) and (23) do not remove any variables and we will also refer to them as transformations. They are only used, when they allow us to apply another reduction or get a previous case: this means that we call the algorithm recursively on the reduced formula; either a variable is removed by the reduction procedure or the algorithm will branch on one of the previous cases.

The first two reductions remove k-cycles with k or $k-1$ negations, the third some special k-cycles.

$$(y_1, \bar{z}_1, z_2) \wedge (y_2, \bar{z}_2, z_3) \wedge \cdots \wedge (y_k, \bar{z}_k, z_1) \wedge F' \rightarrow F[y_1, y_2, \ldots, y_k \leftarrow \mathit{false}] \tag{16}$$

$$(y_1, z_1, z_2) \wedge (y_2, \bar{z}_2, z_3) \wedge \cdots \wedge (y_k, \bar{z}_k, z_1) \wedge F' \rightarrow F[z_1 \leftarrow \mathit{false}] \tag{17}$$

$$\underset{\substack{y_i \text{ occur unnegated in a clause with the literal } x \text{ and the} \\ \text{parities of } k \text{ and the number of negations are different}}}{(\tilde{y}_1, \tilde{z}_1, z_2) \wedge (\tilde{y}_2, \tilde{z}_2, z_3) \wedge \cdots \wedge (\tilde{y}_k, \tilde{z}_k, z_1) \wedge F'} \rightarrow F[x \leftarrow \mathit{false}] \tag{18}$$

If there is a 3$-$cycle with one negation, we can use (19) to either add a clause with the three variables not on the cycle or if all the four clauses are there remove any one of them. If the 3-cycle has a unique variable in the clause without the negated variable, we can remove this clause by (20). If u is not unique, but also occurs in $(\bar{u}, y_2, y_3)$, but in no other clauses, we can remove that clause by (19) and still use this reduction.

$$(y_1, z_1, z_2) \wedge (y_2, z_2, z_3) \wedge (y_3, \bar{z}_3, z_1) \wedge F' \leftrightarrow (y_1, z_1, z_2) \wedge (y_2, z_2, z_3) \wedge (y_3, \bar{z}_3, z_1) \wedge (\bar{y}_1, y_2, y_3) \wedge F' \tag{19}$$

$$\underset{u \text{ unique}}{(u, z_1, z_2) \wedge (y_2, z_2, z_3) \wedge (y_3, \bar{z}_3, z_1) \wedge F'} \rightarrow (y_2, z_2, z_3) \wedge (y_3, \bar{z}_3, z_1) \wedge F' \tag{20}$$

If two 3-cycles without negations share two clauses, we can reduce the formula by (21) or (22).

$$(y_1, z_1, z_2) \wedge (y_2, z_2, z_3) \wedge (y_3, z_3, z_1) \wedge (y_1, y_2, z_4) \wedge F' \rightarrow (z_1, z_2, z_3) \wedge F'[y_1 \leftarrow z_3, y_2 \leftarrow z_1, y_3 \leftarrow z_2, z_4 \leftarrow z_2] \tag{21}$$

$$(y, z_1, z_2) \wedge (y, z_3, z_4) \wedge (y, z_5, z_6) \wedge (z_1, z_3, z_5) \wedge F' \rightarrow F[y \leftarrow \mathit{false}] \tag{22}$$

If we have a 3-cycle with no negations and one of the variables is unique, we can transform the formula by (23).

$$\underset{u \text{ unique}}{(y_1, z_1, z_2) \wedge (y_2, z_2, z_3) \wedge (u, z_3, z_1) \wedge F'} \rightarrow (y_1, z_1, z_2) \wedge (y_2, z_2, z_3) \wedge (\bar{y}_1, z_3, u) \wedge F' \tag{23}$$

If there is a 4-cycle with two negations and the negated variables occur nowhere else, the formula can be reduced by (24) or (25) (w is a new variable).

$$(y_1, z_1, z_2) \wedge (y_2, \bar{z}_2, z_3) \wedge (y_3, z_3, z_4) \wedge (y_4, \bar{z}_4, z_1) \wedge F' \underset{z_2, z_4 \notin V(F')}{\longrightarrow} (w, z_1, z_3) \wedge (\bar{w}, y_1, y_2) \wedge (\bar{w}, y_3, y_4) \wedge F' \tag{24}$$

$$(y_1, z_1, z_2) \wedge (y_2, z_2, z_3) \wedge (y_3, \bar{z}_3, z_4) \wedge (y_4, \bar{z}_4, z_1) \wedge F' \underset{z_3, z_4 \notin V(F')}{\longrightarrow} (y_1, z_1, z_2) \wedge (\bar{y}_1, y_2, w) \wedge (\bar{w}, y_3, y_4) \wedge F' \tag{25}$$

Lemma 6. *Reductions (16) to (25) are sound.*

Proof. In (16), the z's are either all *true* or all *false* or there would be a clause with two true literals. So the y's must be *false*.

In (17), if z_1 is *true* z_2 must be *false*, but then also z_3 must be *false* and the remaining z's must be *false*; then in the last clause, both z_1 and $\bar{z}_k$ are *true*, which is a contradiction.

Reduction (18) is proved with a simple counting argument: let n_1 be the number of y's that are negated and n_2 the number of z's occurring negated. In a satisfying assignment with x *true*, n_1 of the clauses will be satisfied by the y's, n_2 of the clauses will be satisfied by the z's occurring negated, and an even number of clauses will be satisfied by the z's occurring only unnegated. This is only possible, if the parities of k and $n_1 + n_2$ are the same.

To prove the soundness of (19), we prove that all assignments satisfying the left hand side of the reduction will also satisfy the right hand side (the opposite is trivial). If y_1 is *true*, z_1 and z_2 must be *false* and y_2 and y_3 must have different values, so $(\bar{y}_1, y_2, y_3)$ is satisfied. If y_1 is *false*, $z_2 = \bar{z}_1$ and y_2 and y_3 are both *false* by (8), so $(\bar{y}_1, y_2, y_3)$ is also satisfied in this case.

As u is unique in (20), the first clause just ensures that z_1 and z_2 are not both *true*, but this is also ensured by the two other clauses, as z_1 is in a clause with $\bar{z}_3$ and z_2 with z_3, so we can remove the first clause.

In (21), setting $y_1 = \bar{z}_3$ leads to a contradiction: by the first and second clause z_2 is *false* and by the second and fourth clause y_2 is *false*, which makes z_3 *true*; now, z_1 should both be *false* (by the third clause) and *true* (by the first clause), so in a satisfying assignment $y_1 = z_3$; then $y_2 = z_1$ by the first and second clause, $z_4 = z_2$ by the second and fourth clause and $y_3 = z_2$. With these substitutions all four clauses have become (z_1, z_2, z_3), and three of the copies are discarded.

In (22), setting y to *true* will set all the z's to false, but then the clause with only z's is not satisfied; thus, y must be *false*.

In (23), the last clause on the left just ensures that not both z_1 and z_3 are *true* and the last clause on the right that not both $\bar{y}_1$ and z_3 are *true*. But by the first two clauses z_1 and z_3 are both *true* if and only if $\bar{y}_1$ and z_3 are both *true*, so we can replace the last clause on the left by the last clause on the right.

In (24), as z_2 does not occur elsewhere the first two clauses just ensure, that exactly one of y_1, y_2, z_1 and z_3 is *true*. This is also achieved by the clauses (w_1, z_1, z_3) and $(\bar{w}_1, y_1, y_2)$ (w_1 is a new variable). Similarly, the last two clauses can be replaced by (w_2, z_1, z_3) and $(\bar{w}_2, y_3, y_4)$, but then $w_1 = w_2$ by (14) and we get the three clauses on the right hand side of (24). Similarly in (25), the last three clauses just ensure that exactly one of y_2, y_3, y_4, z_1 and z_2 is *true*, but this can also be expressed by the clauses (w_1, z_1, z_2), $(\bar{w}_1, y_2, w_2)$ and $(\bar{w}_2, y_3, y_4)$; then $w_1 = y_1$ by (14), so we get the three clauses on the right hand side. $\square$

3.3.1 General branching

Now, we state our algorithm for X3SAT. If we have an (a, b)-occurrence x occurring in the clauses in Figure 8, we let $Y_1 = \{y_1, y_2, \dots\}$ be the set of variables that occur in a clause with x, $Y_2 = \{y'_1, y'_2, \dots\}$ those that occur in a clause with $\bar{x}$ and $Y = Y_1 \cup Y_2$. We let y's be variables in Y_1, y''s be

$$\boxed{\begin{array}{lll} (x, y_1, y_2) & (\bar{x}, y'_1, y'_2) & x = \mathit{true} : y_1 = y_2 = \cdots = y_{2a-1} = y_{2a} = \mathit{false}, \\ (x, y_3, y_4) & (\bar{x}, y'_3, y'_4) & \qquad\qquad y'_2 = \bar{y}'_1, y'_4 = \bar{y}'_3, \dots, y_{2b} = \bar{y}'_{2b-1} \\ \vdots & \vdots & x = \mathit{false} : y_2 = \bar{y}_1, y_4 = \bar{y}_3, \dots, y_{2b} = \bar{y}_{2b-1}, \\ (x, y_{2a-1}, y_{2a}) & (\bar{x}, y'_{2b-1}, y'_{2b}) & \qquad\qquad y'_1 = y'_2 = \cdots = y'_{2a-1} = y'_{2a} = \mathit{false} \end{array}}$$

Figure 8: Branching on an (a, b)-occurrence x.

variables in Y_2, z's be variables that are not x and not in Y and w's be variables that are not x. By looking at $\bar{x}$ instead of x we swap Y_1 and Y_2.

If we branch on x in Figure 8, we get a branching vector of at least $(2a + b + 1, 2b + a + 1)$ from the above clauses. If $a + b \geq 5$, this yields a branching vector of at least $(11, 6)$, $(10, 7)$ or $(9, 8)$. For variables occurring fewer times, we also need to consider the other clauses in which the y's occur. We start with a lemma showing some cases, in which we can reduce F.

Lemma 7. *If a reduced formula F contains a clause with three variables from Y that is not $(\bar{y}_1, y_3, y_5)$ or if F contains a clause $(\bar{y}_1, \bar{y}_3, z_1)$ or $(\tilde{y}_1, \tilde{y}'_1, z_1)$, where at least one of y_1 and y'_1 is negated, F can be reduced.*

Proof. If F contains a clause with y_1 and y'_1 where at least one of them is negated or with y_1 and y_3 where both of them are negated, it contains a 3-cycle with two or three negations and we reduce it by (16) or (17). If F contains the clause (y_1, y'_1, y_3), we add the clause $(y_2, y'_2, \bar{y}_3)$ by (19) and have the previous case. The only case left is if F contains the clause (y_1, y_3, y_5); then x must be *false* by (22). $\square$

If x is a $(3,1)$-occurrence or a $(2,2)$-occurrence, one of the variables in one clause with $\bar{x}$ say y'_1 must occur in another clause by (15) and the clause must be $(\tilde{y}'_1, w, z_1)$ by Lemma 7, since y'_1 can not occur with two other y''s as x only occurs negated in at most two clauses. From the clauses in Figure 8, we get a branching vector of at least $(8,6)$ or $(7,7)$, but setting x to *false*, also removes z_1 by (1) or (4) and we get a branching vector of at least $(8,7)$. Now, we have removed all variables occurring at least four times in the formula, except $(4,0)$-occurrences.

3.3.2 Branching on $(2,1)$-occurrences

By (15), at least one y from each clause with x and two from one are in other clauses. We want to show, that in all cases we can either reduce F or branch on x and get a branching vector of at least $(8,7)$ or $(9,6)$. From the clauses with x, we get a branching vector of at least $(6,5)$ (see Figure 8). We want to show, that we can always remove at least four more variables in total in the two branches from the other clauses with the variables from Y. First, we prove two lemmas showing, when we can reduce the formula.

Lemma 8. *If a reduced formula F contains the clause $(\tilde{y}_1, \tilde{y}'_1, z_1)$ and a clause containing z_1 and a variable from $\{y_1, y_2, y'_1, y'_2\}$ and the clauses are not (y_1, y'_1, z_1) and $(y_2, y'_2, \bar{z}_1)$, F can be reduced.*

Proof. By Lemma 7, if the first clause is not (y_1, y'_1, z_1), F can be reduced. If F does not contain the clause $(y_2, y'_2, \bar{z}_1)$, we add it by (19) and since the second clause was not this one we have two clauses sharing at least two variables and we reduce F by one of (7), (8) or (14). $\square$

$$\begin{array}{lll}
(x, y_1, y_2)\ (y_1, y_1', z_1) & (x, y_1, y_2)\ (\bar{y}_1, y_3, z_1) & (x, y_1, y_2)\ (y_1, y_3, z_1) \\
(x, y_3, y_4) & (x, y_3, y_4) & (x, y_3, y_4) \\
(\bar{x}, y_1', y_2') & (\bar{x}, y_1', y_2') & (\bar{x}, y_1', y_2') \\
\text{(a)} & \text{(b)} & \text{(c)}
\end{array}$$

Figure 9: A $(2, 1)$-occurrence x and a clause with two variables from Y.

Lemma 9. *If a reduced formula F contains the clause $(\tilde{y}_1, \tilde{y}_3, z_1)$ and a clause containing z_1 and a variable from $\{y_1, y_2, y_3, y_4\}$ and the clauses are not $(\bar{y}_1, y_3, z_1)$ and $(y_2, \bar{y}_4, z_1)$, F can be reduced.*

Proof. By Lemma 7, at most one of y_1 and y_3 is negated. Suppose one is negated, then we have the clauses in Figure 9(b). If the other clause is not $(y_2, \bar{y}_4, z_1)$, we add this clause by (19). Now, the other clause with z_1 and one of the y's will share at least two variables with one of the other two clauses with z_1 and we reduce by one of (7), (8) or (14).

Suppose we have the clauses in Figure 9(c). By symmetry, we can assume that the second clause with z_1 is $(\tilde{y}_2, \tilde{z}_1, w)$. If none of y_2 and z_1 is negated, we reduce F by (21) and if both are negated there is a 3-cycle with two negations, so we reduce F by (17). If only one is negated we have a 3-cycle with one negation and we add a clause with x and y_3 by (19), where one of them is negated and reduce F by (7). $\square$

Note, that we can not have a clause with three variables from Y when x is a $(2, 1)$-occurrence, by Lemma 7.

Suppose F contains the clauses in Figure 9(a); using (19) we can transform F to contain one or both of (y_1, y_1', z_1) and $(y_2, y_2', \bar{z}_1)$. Now, if there is only one other clause, C, with variables from Y, we reduce F: we can choose to let F contain one of the above clauses such that at most three variables from Y occur in this clause and C and reduce F by (15). Suppose that F contains two other clauses with variables from Y and that they contain different z's, none of which is z_1; then branching on x yields $(8, 7)$ or $(9, 6)$: in both branches, y_1 or y_1' is set to *false* in (y_1, y_1', z_1), so z_1 is removed by (4). In each of the other clauses, one variable is removed when the y is set to either *true* or *false*.

Suppose, on the other hand, that F does not contain two such clauses; then z_1 is in no more clauses with y_1, y_2, y_1' or y_2' by Lemma 8 and if it is in a clause with y_3, the clause contains no other variable from Y. Suppose F contains the clause $(\tilde{y}_3, \tilde{z}_1, z_2)$; then z_1 can be in no more clauses with variables from Y. F must contain another clause with a variable from Y

and since it does not contain z_1 or any z_3, it must contain z_2 and two variables from Y. Using Lemma 8 and 9, we have that this clause must be (y_1, y'_2, z_2) or $(y_2, y'_1, \bar{z}_2)$ (or (y'_1, y_2, z_2) or $(y'_2, y_1, \bar{z}_2)$, which we handle similarly) and we add the other by (19). Now, we have the clauses in Figure 10, but then we have a 3-cycle with two negations which we remove

(x, y_1, y_2)	(y_1, y'_1, z_1)	$(\tilde{y}_3, \tilde{z}_1, z_2)$	(y_1, y'_2, z_2)
(x, y_3, y_4)	$(y_2, y'_2, \bar{z}_1)$		$(y_2, y'_1, \bar{z}_2)$
$(\bar{x}, y'_1, y'_2)$			

Figure 10: A special case for $(2, 1)$-occurrence x, in which we can reduce.

by (17): the 3-cycle contains the bottom clause in the fourth column, the clause in the third column and one of the clauses in the second column (which one depends on whether z_1 is negated in $(\tilde{y}_3, \tilde{z}_1, z_2)$).

If the two other clauses with variables from Y do not contain z_1 and not two different z's, they must be of the form $(\bar{y}_1, y_3, z_2)$ and $(y_2, \bar{y}_4, z_2)$ or (y_3, y'_1, z_2) and $(y_4, y'_2, \bar{z}_2)$ by Lemma 8 and 9, but then we remove one of the clauses by (19) and have the previous case with only one other clause with variables from Y. This completes all cases with the clauses in Figure 9(a).

Suppose F contains the clauses in Figure 9(b). If y_1 is only in the clause $(\bar{y}_1, y_3, z_1)$ and the one with x, we replace the first clause by $(y_2, \bar{y}_4, z_1)$ by applying (19) twice and reduce F by (20). If y_1 is in another clause, it must be a $(2, 1)$-occurrence and since x and y_3 are in a clause together, we have the previous case.

Suppose F contains the clauses in Figure 9(c). If y_2 is unique, we transform F by (23) and replace the clause (x, y_1, y_2) by $(x, \bar{z}_1, y_2)$, but then we have the previous case. Now, y_2 and symmetrically y_4 must occur in another clause. If they occur together in a clause, they must occur unnegated or we have the previous case, but then we reduce F by (21). The clauses with y_2 and y_4 do not contain z_1 by Lemma 9 and no y' or negated y by the previous cases, so they must contain two different z's. As before, this yields a branching vector of at least $(9, 6)$ branching on x.

If no two variables from Y occur in the same clause, we get $(9, 6)$ or $(8, 7)$ branching on x: at least two of the variables in Y_1 and at least one of those in Y_2 must be in another clause, which removes at least two extra variables in the *true* branch and one in the *false* branch. Now, at least one more of the variables from Y must be in another clause. This removes an extra variable in one branch, unless there are three clauses with a variable from Y_1 and the z's in these three clauses form a 3-cycle, but then F can be reduced by Lemma 10.

Lemma 10. *If a reduced formula F contains a 3-cycle consisting of the clauses $(\tilde{y}_1, \tilde{z}_1, z_2)$, $(\tilde{y}_2, \tilde{z}_2, z_3)$ and $(\tilde{y}_3, \tilde{z}_3, z_1)$, F can be reduced.*

Proof. If more than one of the z's are negated, F is reduced by (16) or (17), and if exactly one is negated, we use (19) to add a clause with y_1, y_2 and y_3, but y_1 and y_2 are already together in a clause with x, so we can reduce F. If none of the z's is negated we look at whether y_1 and y_2 are negated. If both are negated we have a 3-cycle with two negations, which we reduce by (17) and if none of them is negated, the formula is reduced by (21). If either y_1 or y_2 is negated, we add a clause with x, z_1 and z_3 by (19) where one of z_1 and z_3 are negated and reduce the formula by (7). $\square$

3.3.3 Branching on $(4,0)$-occurrences

To get the desired branching vector for $(4,0)$-occurrences, we show that if there are no variables occurring both unnegated and negated except $(1,1)$-occurrences, we can extend Lemma 7 and reduce in all cases with a clause with three y's.

Lemma 11. *If a reduced formula F containing only $(a,0)$- and $(1,1)$-occurrences contains a clause $(\tilde{y}_1, \tilde{y}_3, \tilde{y}_5)$ or a clause $(\bar{y}_1, \tilde{y}_3, w)$, F can be reduced.*

Proof. The only case not covered by Lemma 7 is if F contains the clause $(\bar{y}_1, y_3, w)$, where w is either y_5 or a z. Now, y_1 must be a $(1,1)$-occurrences so we replace this clause by $(y_2, \bar{y}_4, w)$ by applying (19) twice and then reduce F by (20). $\square$

If x is a $(4,0)$-occurrence, branching on x yields a branching vector of at least $(9,5)$ from the clauses in Figure 8. At least five of the y's must occur in another clause, but then they must occur with at least two different z's or we reduce F by Lemma 9 or 11. When we set x to *true* all the y's are *false* and at least two z's are removed and we get a branching vector of at least $(11,5)$.

3.3.4 Branching on $(3,0)$-occurrences

When we look at a $(3,0)$-occurrence x, we know by Lemma 11 that no three variables from Y occur together and if two occur together they must both occur unnegated.

F contains a clause with two variables from Y. Suppose F contains the clause (y_1, y_3, z_1) and two of y_2, y_4 and z_1 are unique, then we use (23) (with the other unique variable as y_2 in (23)) and remove one of the unique variables by (20), and if only one of them is negated, we use (23) and get a $(2, 1)$-occurrence, which is a previous case. If z_1 is in a clause with any of the variables y_1, y_2, y_3 or y_4, we reduce F by Lemma 9. If two of y_1, y_2, y_3 and y_4 occur together in a second clause they must both be unnegated by Lemma 11, but then we reduce F by (14), (21) or (22), so we must have the clauses in Figure 11, where z_2 is a new

$$\begin{array}{ll}
(x, y_1, y_2)\ (y_1, y_3, z_1) & y_2 = \mathit{true} : x = y_1 = \mathit{false}, y_4 = \bar{y}_3, y_6 = \bar{y}_5, z_1 = \bar{y}_3 \\
(x, y_3, y_4)\ (\tilde{y}_2, w, z_2) & y_2 = \mathit{false} : y_1 = \bar{x}, y_3 \overset{(7)}{=} \mathit{false}, y_4 = \bar{x}, z_1 = \bar{y}_1 \\
(x, y_5, y_6) & \tilde{y}_2 = \mathit{true} : w = z_2 = \mathit{false} \\
 & \tilde{y}_2 = \mathit{false} : z_2 = \bar{w}
\end{array}$$

Figure 11: A clause with two y's. We branch on y_2.

variable, and w can be either y_5 (it can not be negated by Lemma 11) or z_3 (another new variable). If w is y_5, the clause has two y's and z_2 can not be unique by the above. The algorithm branches on y_2. Let us first look at what happens in the first four clauses; setting y_2 to *true* removes the six variables depicted in Figure 11, and setting y_2 to *false* makes $y_1 = \bar{x}$. Then we have the two clauses (x, y_3, y_4) and $(\bar{x}, y_3, z_1)$ which makes y_3 *false* by (7) and we remove the last two variables in Figure 11 for at total of five variables. Now, we look at what happens with the clause $(\tilde{y}_2, w, z_2)$; when $\tilde{y}_2$ is set to *false*, we remove z_2, and when $\tilde{y}_2$ is set to *true*, both w and z_2 are set to *false*. This removes two additional variables in this branch: w is either y_5 or z_3, but neither has gotten a value in any of the branches. Now, either z_2 is in another clause or w is z_3 and is in another clause. This clause can at most contain one of y_1, y_2, y_3, y_4 or z_1 by Lemma 9, so we remove an extra variable from this clause when z_2 or z_3 is set to *false*. In total, we remove three variables in one branch and one in the other. This yields a branching vector of at least $(9, 6)$ or $(8, 7)$.

Cycles At this point, the only clauses containing y's except the clauses with x contain only one y. By (15), at least four of the y's are not unique, so there must be at least four such clauses. We want to branch on x; when we set it to *true*, all the y's are set to *false* and the literals from clauses with unnegated y's are set to the negation of each other and the ones from clauses with negated y's are set to *false*. This removes at least as many extra variables as there are clauses with y's, unless some of these clauses form a cycle as in Figure 12. We are only concerned with 3-, 4-

(x, y_1, y_2) $(\tilde{y}_1, \tilde{z}_1, z_2)$	(x, y_1, y_2) $(\tilde{y}_1, \tilde{z}_1, z_2)$	(x, y_1, y_2) $(\tilde{y}_1, \tilde{z}_1, z_2)$
(x, y_3, y_4) $(\tilde{y}_i, \tilde{z}_2, z_3)$	(x, y_3, y_4) $(\tilde{y}_i, \tilde{z}_2, z_3)$	(x, y_3, y_4) $(\tilde{y}_i, \tilde{z}_2, z_3)$
(x, y_5, y_6) $(\tilde{y}_j, \tilde{z}_3, z_1)$	(x, y_5, y_6) $(\tilde{y}_j, \tilde{z}_3, z_4)$	(x, y_5, y_6) $(\tilde{y}_j, \tilde{z}_3, z_4)$
	$(\tilde{y}_k, \tilde{z}_4, z_1)$	$(\tilde{y}_k, \tilde{z}_4, z_5)$
		$(\tilde{y}_l, \tilde{z}_5, z_1)$
(a) 3-cycle.	(b) 4-cycle.	(c) 5-cycle.

Figure 12: Cycles.

and 5-cycles; we can not have 2-cycles, as these would have been removed by (7), (8) or (14) and if we have at least a 6-cycle, we remove at least five z's, when we set x to *true*, but this yields a branching vector of at least $(12, 4)$, which is what we are after.

Lemma 12. *If a reduced formula does not satisfy any of the previous cases, the same y can not occur twice in a 3-, 4- or 5-cycle.*

Proof. If a y occurs twice in a 3-, 4- or 5-cycle, it occurs at least three times in F so it must be a $(3, 0)$-occurrence. It can not occur in two neighbouring clauses on the cycle by (7) and (14); since we are dealing with at most 5-cycles, it must then occur in two clauses $(y_1, \tilde{z}_1, z_2)$ and $(y_1, \tilde{z}_3, z_4)$, where z_2 and z_3 are together in another clause on the cycle. Then y_1 is a $(3, 0)$-occurrence with two of the variables it occurs with occurring together in another clause, but that is a previous case. □

Lemma 13. *If a reduced formula F containing only $(a, 0)$- and $(1, 1)$-occurrences contains two clauses $(\tilde{y}_i, z_1, z_2)$ and $(\tilde{y}_j, \tilde{z}_2, z_3)$, where at least two of y_i, y_j and z_2 are negated, we can reduce F.*

Proof. If y_i and y_j are from the same clause with x, that clause and the two clauses in the lemma form a 3-cycle with two or three negations, so we can reduce F by (16) or (17). If y_i and y_j are from different clauses with x, the clauses form a 4-cycle with two or three negations together with the two clauses where y_i and y_j occur with x, and we reduce F by (17), (24) or (25), since we do not have $(2, 1)$-occurrences. □

In the following, we show how to deal with the remaining cases of 3-, 4- and 5-cycles.

3-cycles. If F contains a 3-cycle as in Figure 12(a), the y's must be from different clauses with x by Lemma 10. If the cycle contains more than one negated z, the formula is reduced by (16) or (17) and if there is exactly one negated z, we add a clause with three y's by (19). If this clause does not contain exactly one negation, we reduce F by Lemma 7; otherwise, the negated y has become a $(2,1)$-occurrence, which is a previous case. Now, suppose that none of the z's is negated; if at least two of the y's are negated, we reduce F by Lemma 13 and if none of the y's is negated we reduce F by (18). So assume the cycle consists of the clauses in Figure 13; then we branch on y_1 and get a branching vector of at least

$$\begin{array}{ll|l}
(x, y_1, y_2) & (\bar{y}_1, z_1, z_2) & y_1 = \mathit{true} : x = y_2 = \mathit{false}, y_4 = \bar{y}_3, y_6 = \bar{y}_5, z_2 = \bar{z}_1, \\
(x, y_3, y_4) & (y_3, z_2, z_3) & \qquad z_3 \overset{(7)}{=} \mathit{false}, y_3 = \bar{z}_2, y_5 = \bar{z}_1 \\
(x, y_5, y_6) & (y_5, z_3, z_1) & y_1 = \mathit{false} : y_2 = \bar{x}, z_1 = z_2 = \mathit{false}, \\
 & & \qquad y_3 = y_5 = \bar{z}_3, y_4 = y_6
\end{array}$$

Figure 13: A 3-cycle with only y_1 negated. We branch on y_1.

$(9,7)$, as shown in the figure. The (7) above the equality means, that this follows from (7): since $z_2 = \bar{z}_1$, we get two clauses with z_1 and z_3 and z_1 is negated in one of them.

4-cycles. Suppose F contains a 4-cycle as in Figure 12(b). If there are two or more negated z's, we reduce the formula by (16), (17), (24) or (25), since F contains no $(2,1)$-occurrences. If there is only one negated z, the two y's in the clauses with the negated variable must be unnegated by Lemma 13; let these clauses be (y_1, z_1, z_2) and $(y_i, \bar{z}_2, z_3)$. If an even number of the y's are negated, we reduce the formula by (18); otherwise, there must be exactly one negated y and the cycle looks like the one in Figure 14(a), but then $x = z_1$. Suppose $x = \bar{z}_1$; then we replace z_1 by $\bar{x}$

(x, y_1, y_2) (y_1, z_1, z_2)	(x, y_1, y_2) (y_1, z_1, z_2)	(x, y_1, y_2) (y_1, z_1, z_2)
(x, y_3, y_4) $(y_i, \bar{z}_2, z_3)$	(x, y_3, y_4) (y_2, z_2, z_3)	(x, y_3, y_4) (y_3, z_2, z_3)
(x, y_5, y_6) $(\bar{y}_j, z_3, z_4)$	(x, y_5, y_6) (y_3, z_3, z_4)	(x, y_5, y_6) (y_2, z_3, z_4)
(y_k, z_4, z_1)	(y_i, z_4, z_1)	(y_4, z_4, z_1)
(a) $x = z_1$.	(b) $z_2 = z_4$.	(c) Remove last clause.

Figure 14: 4-cycles.

and get that both y_1 and y_k are in clauses with both x and $\bar{x}$, so they must be *false* by (7), but then $z_2 = x$, so y_i must also be *false* by (7). Now, both z_3 and z_4 must be equal to x, but then x must be *false* by (5) and y_j must also be *false*. This is a contradiction: since none of the y's are the same by Lemma 12, at least two of them are from the same clause with x, but they are all *false* and so is x.

Suppose that none of the z's is negated. If an odd number of the y's are negated we reduce F by (18) and if two y's in neighbouring clauses are negated, we reduce F by Lemma 13. If two y's in "opposite" clauses on the cycle are negated, x must be *false*: if x is *true*, all y's are set to *false* and by the two clauses with negated y's all the z's are set to *false*, but then the other two y's must be *true*, a contradiction, so x must be *false*. In the remaining cases, none of the y's is negated. If two y's in neighbouring clauses are from the same clause with x we have the cycle in Figure 14(b), but then $z_2 = z_4$: suppose $z_2 = \bar{z}_4$; then z_1 and z_3 must be *false* by (7), but then $y_1 = y_2 = \bar{z}_2 = z_4 = \bar{y}_3$. Then y_1 and y_2 must be *false*, but then both x and y_3 must be *true*, a contradiction.

Lemma 14. *If a reduced formula F contains a 4-cycle as in Figure 14(c), F is satisfiable iff F with the last clause on the cycle removed is satisfiable.*

Proof. It is trivially true, that if F is satisfiable, so is F with the last clause removed. Suppose that F without the last clause is satisfied. If x is *true*, all the y's are *false*, so from the other three clauses, we have that $z_1 = \bar{z}_2 = z_3 = \bar{z}_4$ so the fourth clause is satisfied. If x is *false*, $y_1 = \bar{y}_2$, so exactly one of the z's are *true*; if it is one of z_1 or z_4, y_3 must be *true* by the second clause so y_4 must be *false* and if it is one of z_2 or z_3, y_3 must be *false* and hence y_4 must be *true* by their common clause with x. In both cases, the last clause is satisfied. $\square$

The only remaining 4-cycles have no negations and no y occurring more than once. Then at least two of the y's must be from the same clause with x and they are not in neighbour clauses on the cycle. If the other two y's are also from the same clause with x, we transform F by removing the last clause on the cycle by Lemma 14. If they are not from the same clause with x the 4-cycle must look like the one in Figure 14(c), except y_4 is replaced by y_5. Then we add the clause (y_4, z_4, z_1) by Lemma 14 and get that $y_4 = y_5$ by (14).

F contains a clause with a negated variable from Y. There must be at least four clauses with variables from Y other than the ones with x. We branch on x; setting it to *false* removes four variables and setting it to *true* sets all the y's to *false* and in each of the other clauses with y's either sets one of the z's to the negation of the other or both to *false*. This removes at least twelve variables, as at least one y is negated and the z's do not form a 3- or 4-cycle.

5-cycles. If F contains a 5-cycle as in Figure 12(c), none of the y's is negated by the previous case; also, none of the y's are the same by Lemma 12, so there must be four clauses on the 5-cycle, as in Figure 15, that contain only y's from two clauses with x. The y's will always satisfy

(x, y_1, y_2)	(y_1, z_1, z_2)	An even number of the z's are negated : $z_1 = z_5$
(x, y_3, y_4)	$(y_i, \tilde{z}_2, z_3)$	An odd number of the z's are negated : $z_1 = \bar{z}_5$
(x, y_5, y_6)	$(y_j, \tilde{z}_3, z_4)$	
	$(y_k, \tilde{z}_4, z_5)$	

Figure 15: Four clauses from a 5-cycle.

an even number of these clauses: if x is *true*, they satisfy zero and if x is *false* they satisfy two, as they pairwise become each other's negation, since they were from only two different clauses with x. Now, let us look at z_2, z_3 and z_4. The negated ones will always satisfy exactly one of the clauses and the unnegated will either satisfy zero or two; thus, if the number of negated z's is odd, z_1 and z_5 must satisfy an odd number of the clauses for F to be satisfiable, so $z_1 = \bar{z}_5$ and if the number of negated z's is even, $z_1 = z_5$ by the same argument. In both cases, we have reduced F.

F contains at least five clauses with y's and z's. Since the clauses do not form 3-, 4- or 5-cycles at this point, branching on x yields at least $(12, 4)$.

F contains exactly four clauses with y's and z's. As at least four variables from Y occur in clauses without x and no two variables from Y occur together, there must be exactly four such clauses. Furthermore, two of the variables from Y are unique, and the others are $(2, 0)$-occurrences; otherwise, there would be more than four clauses with variables from Y.

Since there are no cycles, we have already seen how to get $(11,4)$ branching on x. If the formula is $\frac{15}{2}$-sparse, we can solve the remaining formula in time $O(2^{\frac{2n}{15}}) = O(2^{0.13333n})$. We want to branch on formulas with too many $(3,0)$-occurrences; we either prove that we remove an extra variable when branching on x or we branch on a different variable.

Suppose x_1 is a $(3,0)$-occurrence occurring in the three first clauses in Figure 16. The other clause with y_3 contains no unique variable by (15). If it contains a variable x_2, that only occurs in other clauses with unique variables as in Figure 16, we branch on x_1; setting it to *true* removes

(x_1, y_1, y_2)	(x_2, y_3, z)	All other clauses with x_2 contain a unique variable.
(x_1, y_3, u_1)		
(x_1, y_5, u_2)		

Figure 16: Unique variables.

eleven variables and setting it to *false* removes x_1, y_2, u_1 and u_2, but then y_3 is unique, so all clauses with x_2 contain a unique variable and we set x_2 to *false* by (15). Then also $z = \bar{y}_3$ and we get a branching vector of at least $(11,6)$.

If another $(3,0)$-occurrence x_2 occurs with one of the variables from Y it must be either y_1 or y_2: suppose x_2 is in a clause with y_3. Now, y_3 is a $(2,0)$-occurrence with a unique variable in its clause with x, so its other clause can not contain unique variables; then the remaining clauses with x_2 must contain unique variables, but that is the previous case. Suppose y_1 is in a clause with x_2, then that clause can not contain a unique variable, as this would also be the previous case (by looking at x_2 instead of x_1). The two other clauses with x_2 must then contain a unique variable, so they do not contain y_3 or y_5 and if they contain y_2, we have a $(3,0)$-occurrence with two of the variables it occurs with occurring together in another clause, which is a previous case. So we must have the clauses in Figure 17 (to easier distinguish variables occurring with different $(3,0)$-occurrences, we use y' and y'' to denote variables occurring with other $(3,0)$-occurrences than x_1 in the rest of this section). Then we branch on y_2; setting it to *true* removes eight variables as shown in Figure 17 and setting it to *false* we get $z_4 = \bar{z}_3$ and $y_1 = \bar{x}_1$, as shown in the figure. Then we have $\bar{x}_1$ in a clause with x_2 and both of them only occur unnegated in clauses with unique variables elsewhere, so x_2 is set to *false* by (15), and we remove the remaining variables in the figure for a branching vector of at least $(8,7)$.

$$\begin{array}{llll} (x_1, y_1, y_2) & (y_1, x_2, y_2') & (x_2, y_3', u_1') & y_2 = \mathit{true} : x_1 = y_1 = z_3 = z_4 = \mathit{false}, \\ (x_1, y_3, u_1) & (y_2, z_3, z_4) & (x_2, y_5', u_2') & \qquad\quad u_1 = \bar{y}_3, u_2 = \bar{y}_5, y_2' = \bar{x}_2 \\ (x_1, y_5, u_2) & (y_3, z_5, z_6) & & y_2 = \mathit{false} : y_1 = \bar{x}_1, z_4 = \bar{z}_3, \\ & (y_5, z_7, z_8) & & \qquad\quad x_2 \overset{(15)}{=} \mathit{false}, y_2' = x_1, \\ & & & \qquad\quad u_1' = \bar{y}_3', u_2' = \bar{y}_5' \end{array}$$

Figure 17: One variable from Y occurs with another $(3, 0)$-occurrence.

Now, no variable from Y occurs with another $(3, 0)$-occurrence. Suppose both z_5 and z_6 occur with a $(3, 0)$-occurrence. Then z_5 and z_6 can not be negated, as a $(3, 0)$-occurrence where one of the variables it occurs with occurs negated in another clause is a previous case; also, the clause with z_5 and z_6 contains y_3, which is only in one other clause and that clause contains a unique variable, so if the clause with z_5 (or z_6) and the $(3, 0)$-occurrence contains a unique variable, we have the case in Figure 16 (with the $(3, 0)$-occurrence as x_1 and z_5 as y_3). Now, z_5 and z_6 can not occur in another clause together, so we must have the clauses in Figure 18. We branch on z_6; setting it to *true* removes nine variables

$$\begin{array}{llll} (x_1, y_1, y_2) & (y_1, z_1, z_2) & (x_2, z_5, y_2') & (x_3, z_6, y_2'') \\ (x_1, y_3, u_1) & (y_2, z_3, z_4) & (x_2, y_3', u_1') & (x_3, y_3'', u_1'') \\ (x_1, y_5, u_2) & (y_3, z_5, z_6) & (x_2, y_5', u_2') & (x_3, y_5'', u_2'') \\ & (y_5, z_7, z_8) & & \end{array}$$

$$z_6 = \mathit{true} : y_3 = z_5 = x_3 = y_2'' = \mathit{false}, u_1'' = \bar{y}_3'', u_2'' = \bar{y}_5'', u_1 = \bar{x}_1, y_2' = \bar{x}_2$$

$$z_6 = \mathit{false} : y_2'' = \bar{x}_3, z_5 = \bar{y}_3, x_2 \overset{(15)}{=} \mathit{false}, y_2' = y_3, u_1' = \bar{y}_3', u_2' = \bar{y}_5'$$

Figure 18: Both z_5 and z_6 occur with a $(3, 0)$-occurrence.

as shown in the figure, and setting z_6 to *false* sets $y_2'' = \bar{x}_3$ and $z_5 = \bar{y}_3$. Now, x_2 is in a clause with $\bar{y}_3$ and both variables only occur unnegated in clauses with unique variables elsewhere, so we set x_2 to *false* by (15) and remove y_2', u_1' and u_2'. In total, we get a branching vector of at least $(9, 7)$.

Sparse formulas. Now, no y can occur with another $(3, 0)$-occurrence, so there are at least six variables occurring at most twice for each $(3, 0)$-occurrence. Also, as z_5 and z_6 (and by symmetry z_7 and z_8) do not both occur with a $(3, 0)$-occurrence, we can assume that neither z_5 nor z_7 occur with a $(3, 0)$-occurrence. As they both occur at most twice, they are in clauses with at most four different variables; thus, we count each of them as one fourth of a variable for each of the $(3, 0)$-occurrences, whose y they occur with. This means, that there are at least six and a half variables occurring at most twice for each $(3, 0)$-occurrence; thus, the formula is $\frac{15}{2}$-sparse and we solve it in time $O(2^{\frac{2n}{15}}) = O(2^{0.13333n})$.

4 Conclusion

The main result of this paper is the following theorem.

Theorem 1. *The presented algorithms for* XSAT *and* X3SAT *run in time $O(2^{0.2325n})$ and $O(2^{0.1379n})$, respectively.*

Proof. When our algorithms are applied to a formula F with m clauses and n variables, the sizes of the intermediate formulas are never larger than $\max(|F|, 2mn)$: they are never larger during the reduction procedures by Lemma 3 and after the reduction procedures have run, the formula has size at most mn. In some of the branches we add a clause, but when we call the algorithm recursively, the reduction procedure will remove a clause, so the size is never larger than $\max(|F|, 2mn)$, which is polynomial in the size of the original formula. Also, no reduction or branching adds variables. The number of recursive calls are at most $O(2^{0.2325n})$ and $O(2^{0.1379n})$, respectively, by Section 3.2 and Section 3.3. For each recursive call, the reduction procedure runs in polynomial time in the size of the formula by Lemma 4 and we can in polynomial time decide, which case to branch on. Since we ignore the polynomial factors, we get the stated running times. $\square$

Both our algorithms are extensions of known branch-and-reduce algorithms. One important addition to the algorithms are new reductions, which limit the number of possible structures of the formula. The other important addition is the concept of sparse formulas, which in certain situations enables us to simply enumerate all possible assignments to the variables we would otherwise branch on and leaves us with a problem that is solvable in polynomial time. One could hope that the concept of sparse formulas is also useful in other algorithms.

Acknowledgements

We would like to thank the anonymous referees for many helpful comments. We are also grateful to our supervisors Peter Bro Miltersen and Sven Skyum for their help with this paper.

References

[1] L. Drori and D. Peleg. Faster exact solutions for some NP-hard problems. *Theoretical Comput. Sci.*, 287(2):473–499, 2002.

[2] A. S. Kulikov. An upper bound $O(2^{0.16254n})$ for exact 3-satisfiability: A simpler proof. *Zapiski nauchnyh seminarov POMI*, 293:118–128, 2002. Can be found at `http://logic.pdmi.ras.ru/~kulikov/x3sat_e.ps.gz`.

[3] O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity, 1997. Manuscript. Can be found at `http://cs-svr1.swan.ac.uk/~csoliver/tg.ps.gz`.

[4] B. Monien, E. Speckenmeyer and O. Vornberger. Upper bounds for covering problems. Bericht 7, Universität Paderborn, 1980.

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

[6] S. Porschen, B. Randerath and E. Speckenmeyer. Exact 3-satisfiability is decidable in time $O(2^{0.16254})$, 2002. To appear in *Annals of Mathematics and Artificial Intelligence.*

[7] S. Porschen, B. Randerath and E. Speckenmeyer. X3SAT is decidable in time $O(2^{n/5})$, 2002. Manuscript. Can be found at `http://gauss.ececs.uc.edu/Conferences/SAT2002/Abstracts/ porschen.ps`

[8] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.

[9] R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.