Upper Bounds for MaxSat: Further Improved*

Nikhil Bansal[1] and Venkatesh Raman[2]

[1] Carnegie Mellon University, Pittsburgh, USA nikhil@cs.cmu.edu
[2] The Institute of Mathematical Sciences, Chennai, India, vraman@imsc.ernet.in

Abstract. Given a boolean CNF formula F of length $|F|$ (sum of the number of variables in each clause) with m clauses on n variables, we prove the following results.

- The MAXSAT problem, which asks for an assignment satisfying the maximum number of clauses of F, can be solved in $O(1.341294^m|F|)$ time.
- The parameterized version of the problem, that is determining whether there exists an assignment satisfying at least k clauses of the formula (for some integer k), can be solved in $O(k^2 1.380278^k + |F|)$ time.
- MAXSAT can be solved in $O(1.105729^{|F|}|F|)$ time.

These bounds improve the recent bounds of respectively $O(1.3972^m|F|)$, $O(k^2 1.3995^k + |F|)$ and $O(1.1279^{|F|}|F|)$ due to Niedermeier and Rossmanith [10] for these problems. Our last bound comes quite close to the $O(1.07578^{|F|}|F|)$ bound of Hirsch[5] for the Satisfiability problem (not MAXSAT). En route to our main algorithms proving the above bounds, we develop an $O(1.324719^n|F|)$ algorithm for the case when every variable in the formula appears at most three times. This bound significantly improves the bound of $O(\sqrt{3}^n|F|)$ due to Raman, Ravikumar and Srinivasa Rao [13] for the problem.

1 Introduction

There are several approaches to deal with an NP-complete problem. A popular approach is to settle for a polynomial time approximate solution with provable guarantee on the quality of the solution. Another well studied approach is to explore some structure in the problem to design efficient algorithms (for example designing efficient algorithms for NP-complete graph problems in special classes of graphs). Another recent approach for the parameterized versions of the NP-complete problems is to look for fixed parameter tractable algorithms [4]. Despite all these approaches, often many NP-hard problems have to be solved exactly, in practice. Due to this reason, considerable attention has been paid to designing efficient (better than the naive 2^n) algorithms for several NP-hard problems[1, 9, 8, 12, 11, 6, 5, 2, 14]. Satisfiability, Vertex Cover, Independent Set are some of the problems for which such efficient exact algorithms are known. In this paper we develop efficient exact algorithms for another fundamental NP-hard problem MAXSAT (Maximum Satisfiability). Despite considerable advances in the study

* The work was done while the first author was at IIT Mumbai and visited IMSc Chennai as a summer student.

of exact algorithms for the Satisfiability problem, desigining efficient exact algorithms for MAXSAT has started receiving the attention of researchers only recently.

For the rest of the paper we will be dealing with boolean CNF formulae on n variables and m clauses. For a formula F, $|F|$ will denote the length of the formula which is the sum of the number of variables in each clause.

Raman, Ravikumar and Srinivasa Rao [13] showed that MAXSAT problem remained NP-hard even if every variable appears at most three times. They gave an $O(\sqrt{3}^n|F|)$ algorithm for this version of MAXSAT problem. For the general MAXSAT, Mahajan and Raman [7] gave an $O(|F|\phi^m)$ algorithm, where $\phi = 1.6181...$ is the golden ratio. This was improved to $O(|F|1.3995^m)$ by Niedermeier and Rossmanith [10]. In this paper, we improve this to $O(|F|1.341294^m)$. For the natural parameterized question, whether at least k clauses of the formula be satisfiable, Niedermeier and Rossmanith [10] improved the algorithm of Mahajan and Raman [7] to obtain a bound of $O(k^2 1.3995^k + |F|)$. In this paper we improve this to $O(k^2 1.380278^k + |F|)$ time.

In the development of efficient exact algorithms for SAT, bounds where the exponent is the length of the formula have been obtained (see, for example, [5, 6]). We also design algorithms for MAXSAT along similar lines to obtain a bound of $O(1.105729^{|F|}|F|)$ for MAXSAT improving upon the recent bound of $O(1.1279^{|F|}|F|)$ due to Niedermeier and Rossmanith [10]. This bound implies that if every variable occurs at most 6 times, then we have a $((2-\epsilon)^n|F|)$ algorithm for some positive $\epsilon < 1$. Our bound also comes quite close to the $O(1.07578^{|F|}|F|)$ bound of Hirsch[5] for the Satisfiability problem (not MAXSAT).

En route to our main algorithms proving the above bounds, we develop an $O(1.324719^n|F|)$ algorithm for the case when every variable in the formula appears at most three times. This bound significantly improves the bound of $O(\sqrt{3}^n|F|)$ due to Raman, Ravikumar and Srinivasa Rao [13] mentioned above. Our improved algorithm for the special case is also used later in our main algorithms.

We remark that all our algorithms employ some simplification rules as well as the standard Davis-Putnam type branching rules, and involve an extensive case analysis as is typically the case with most of the exact algorithms refered earlier [1, 9, 8, 6, 2, 10]. Further we emphasize that throughout the development of algorithms for these problems, even small improvements in the second or even the third digit after the decimal in the exponent can mean significant progress.

In the next section, we give some notations and definitions used in our algorithms. Section 3 gives some simplification or reduction rules. Section 4 gives some branching rules which will be later employed in the main algorithms. Section 5 gives our improved algorithm for MAXSAT version where each variable appears at most three times. In Section 6, two new algorithms are proposed for the general MAXSAT problem. The exponent in the time bound of the first one is the number of clauses and that of the second one is the length of the formula.

2 Notation

Let l be a literal in a formula F. We will call it an $(i,j)[p_1,p_2,\ldots,p_i][n_1,n_2,\ldots,n_j]$ literal if it occurs i times positively and j times negatively and the clauses containing l are of length $p_1 \le p_2 \le \ldots \le p_i$ and those containing $\bar{l}$ are of length $n_1 \le n_2 \le \ldots \le n_j$. If the length of the clauses is not important we will simply call it an (i,j) literal. If we say that some variable or literal occurs i^+ times, we will mean that it occurs at least i times. Similarly i^- denotes at most i occurrences. For a variable x, we say $\tilde{x}$ occurs in a clause C if $x \in C$ or $\bar{x} \in C$. We call x a k-variable, if x is some (i,j) literal such that $i+j=k$.

If a literal x occurs as a unit clause $\{x\}$, k times in the formula, we will denote the fact by $n_u(x)=k$. For a literal x, $l(x)$ will denote the sum of the lengths of the clauses containing x.

If F is a formula, $F[p_1,\ldots,p_k][n_1,\ldots,n_l]$ will denote the formula obtained by putting the literals $p_1=\cdots=p_k=true$ and $n_1=\cdots=n_l=false$. If C is a clause then $F[][C]$ will denote the formula obtained by putting all the literals in C as false.

Our algorithms for MAXSAT first go through some simplification and reduction rules where the given formula is simplified. On the reduced formula, we then apply some branching rules. If a branching rule for the formula F branches as $F[X_1][Y_1]$, $F[X_2][Y_2]\ldots, F[X_i][Y_i]$ it means that each of the formulae $F[X_j][Y_j], j=1$ to i is solved (recursively or otherwise) and the assignment that satisfies the maximum number (or the required number, in the parameterized problem) of clauses among them is returned. The branching rule ensures that the returned assignment actually satisfies the maximum (or the required) number of clauses of F. If the question to be answered is a decision question (like "can k clauses be satisfied?"), then the answer returned is 'yes' if any of the branches returns 'yes' and 'no' otherwise. If the "size" of F is q and the sizes of $F[X_j][Y_j] \le q - r_j, \forall j$, where "size" is the number of variables, the number of clauses or the number of clauses to be satisfied, as the case may be, then such a branching rule is said to have a *branching vector* $(r_1,r_2,\ldots,r_i)$.

If two clauses C_1 and C_2 contain some pair of complementary variables, we will denote the fact by $Co(C_1,C_2)$. A subset of clauses is said to be *closed* if no variable in this subset occurs outside this subset in the rest of the formula. For a clause C, $C - \{a_1,a_2,\ldots,a_k\}$ will denote the clause obtained by deleting all instances of the literals $a_1,a_2,\ldots,a_k$ from the clause C. We will denote the number of clauses in a formula F, by $m(F)$. The maximum number of clauses that are satisfied by an optimal assignment will be denoted by $opt(F)$.

3 Simplification rules

We list a set of rules to reduce a given CNF formula so that solving the MAXSAT problem for the original formula is equivalent to solving it for the reduced one. First we assume that the formula has no empty clause (which is vacuously false), and that every variable appears both positively and negatively - i.e. there are no $(i,0)$ or $(0,i)$ variables as they can be eliminated easily by assigning them an appropriate value.

1. **Elimination of $(1,1)$ literals:** If a is a $(1,1)$ literal, and $a \in C_1$ and $\bar{a} \in C_2$, then if $Co(C_1 - \{a\}, C_2 - \{\bar{a}\})$ just remove the clauses. Otherwise, replace them by a single clause $\{C_1 \cup C_2 - \{a, \bar{a}\}\}$.
2. **Replacement of almost common clauses:** If $\exists$ clauses C_1 and C_2 and a literal a, such that $C_1 - \{a\} = C_2 - \{\bar{a}\}$, then replace C_1 and C_2 by the clause $\{C_1 \cup C_2 - \{a, \bar{a}\}\}$.
   In particular this implies that we cannot have $\{a\}$ and $\{\bar{a}\}$ together as unit clauses in the same formula.

In both these rules, if F' is obtained from F by application of the rule, clearly $opt(F) - opt(F') = m(F) - m(F')$ and this justifies the rules.

Note that each simplification rule can be applied for a formula F in $O(|F|)$ time. Further after all simplification rules are applied, every (remaining) variable appears both positively and negatively and each variable appears at least three times in the formula. Also if a variable appears as a unit clause, all its occurrences in the unit clauses are either all pure or all negated.

4 Branching rules

In this section, we give some rules to abandon certain branches of the davis-putnam type branching algorithm. The reason for abandoning them (which we may not specifically mention in some rules due to page limitation) is that in one of the branches followed, the number of clauses satisfied is at least as many as the maximum number of clauses satisfied in any of the branches abandoned.

1. If x is a (k,l) literal, and $n_u(\bar{x}) \geq k$, then branch as $F[][x]$.
2. If x is a (k,l) literal, and $n_u(\bar{x}) = k - 1$, let $C_1, \ldots, C_k$ denote the clauses containing x. If $\exists\ i, j$ such that $Co(C_i, C_j)$, then branch as $F[][x]$ (as at least one of C_i and C_j is always satisfiable). Otherwise, branch as $F[][x]$ and $F[x][C_1 \cup \ldots \cup C_k - \{x\}]$.

3. Branching rules for $(1,k)$ literals.
   (a) If x is a $(1,k)[\ldots][1,\ldots]$ literal, then branch as $F[][x]$. This is a special case of rule 1.
   (b) If x is a $(1,k)[2^+][\ldots]$ literal, then if C is the clause containing x, branch as $F[][x]$ and $F[x][C - \{x\}]$. This is a special case of rule $2b$.
   (c) If x is a $(1,k)[\ldots][2,\ldots]$ literal, then if C is the clause of size 2 containing $\bar{x}$, branch as $F[][x]$ and $F[x, C - \{\bar{x}\}][]$. This gives a $(k,2)$ branch.
4. Let x be a (k,l) literal and let $C_1, \ldots, C_k$ be the clauses containing x. If there is a variable $\tilde{y}$ such that there are only one or two occurrences of $\tilde{y}$ in clauses other than $C_1, \ldots, C_k$, then branch as $F[x][]$ and $F[][x]$. This gives a $(k+1, l)$ branching vector. For, setting $x = true$ and applying the reduction rule will eliminate $\tilde{y}$, thus satisfying at least $k+1$ clauses.
5. If x is a $(2,2)$ literal, and C_1 and C_2 are the clauses containing $\bar{x}$. Branch as $F[x][]$, $F[][x, C_1 - \{\bar{x}\}]$ and $F[][x, C_2 - \{\bar{x}\}]$.
   Since, if there is an optimum assignment with $x = false$ and $C_1 - \{\bar{x}\} = true$ and $C_2 - \{\bar{x}\} = true$, then we might as well set $x = true$.
6. Let x be a $(1,3)[1][\ldots]$ literal and C_1, C_2 and C_3 are the clauses containing $\bar{x}$. Then, if there exist some two clauses, say C_2 and C_3 which do not contain any

pair of complementary variables, it is sufficient to branch as $F[x][], F[][x, C_1 - \{\bar{x}\}]$ and $F[][x, C_2 - \{\bar{x}\}, C_3 - \{\bar{x}\}]$. Otherwise, branch as $F[x][]$.
Since if $Co(C_1, C_2)$, $Co(C_1, C_3)$ and $Co(C_2, C_3)$, then it can be seen that for any assignment at least two of the clauses in C_1, C_2 and C_3 are satisfied. Otherwise, if there is an optimal solution with $x = false$ such that at least two of $C_1 - \{\bar{x}\}, C_2 - \{\bar{x}\}$ and $C_3 - \{\bar{x}\}$ are true, then setting $x = true$ does not decrease the number of clauses satisfied.

7. If x is a $(1, 2)$ literal, and C_1 and C_2 are the clauses containing $\bar{x}$. Branch as $F[x][], F[][x, C_1 - \{\bar{x}\}, C_2 - \{\bar{x}\}]$.

5 An algorithm for $(n, 3)$ MAXSAT

Suppose each variable appears atmost 3 times. This instance of MAXSAT is called $(n, 3)$ MAXSAT [13]. In this section, we give an efficient algorithm for $(n, 3)$ MAXSAT. We will call a branch *good*, if either it is a (1) branch, a $(2, 3)$ branch or a $(1, 5)$ branch. We give an algorithm which uses *good* branches both with respect to the number of satisfied clauses (for the parameterized case) and the number of variables. That is, for the exact MAXSAT problem, if the branching rule has a $(2, 3)$ branching vector, then two variables are eliminated in one branch and three in the other, and similarly for the other branches. Similarly for the parameterized problem where we would like to know whether k clauses can be satisfied, if the branching rule has a $(2, 3)$ branching vector, then two clauses are satisfied in one branch and three in the other, and similarly for the other branches.

Thus, the branching tree with the good branches at each node is grown at each step until we reach a leaf (where the answer is trivially true or trivially false). The time spent at each node is $O(|F|)$ and hence the total time spent is $O(|F|)$ times the size of the branching tree. Clearly, the size of the tree $S(n)$ satisfies, say, for the MAXSAT problem

$$S(n) \leq S(n-2) + S(n-3) + 1 \quad (1)$$

$$S(n) \leq S(n-1) + S(n-5) + 1 \quad (2)$$

$$\text{and } S(n) \leq S(n-1) + 1 \quad (3)$$

whose solution is $O(1.324719^n)$ with the first inequality being the most tight one.

For the parameterized question, one can first reduce the formula by spending $O(|F|)$ time to have only $O(k^2)$ clauses [7]. So applying the above branching algorithm after the reduction, we will have the running time of the algorithm to be $O((1.324719)^k k^2 + |F|)$ for the parameterized question and $(1.324719^n|F|)$ for the exact MAXSAT problem. In what follows, we give a branching algorithm where all the branchings are good.

Note that for all variables $\tilde{x}$, either x or $\bar{x}$ is a $(1, 2)$ literal. Also if $C = \{a_1, a_2, \ldots, a_k\}$ is a clause then setting $a_1 = true$ and applying reduction rules will also eliminate the variables $\tilde{a_2}, \ldots, \tilde{a_k}$ from the formula. In particular, if $k \geq 2$ at least two clauses will be satisfied on setting $a_1 = true$. Because, the clause containing a_1 is satisfied and at least another clause is satisfied during the elimination of $\tilde{a_2}, \ldots, \tilde{a_k}$, using rules in Section 3.

Algorithm

Apply all the reduction rules until not possible.

1. If a is a $(1,2)[\ldots][1,\ldots]$ literal, then branch as $F[][a]$ (Branching rule 1).
2. If a is a $(1,2)[2^+][2^+,2^+]$ literal, let C_1 be the clause containing a and C_2 and C_3 be the clauses containing $\bar{a}$.
   (a) Let $C_2 \cup C_3 - \{\bar{a}\}$ contains more than one variable.
   We branch as $F[][a]$ and $F[a][]$. Clearly $F[][a]$ satisfies the two clauses containing $\bar{a}$, another clause will be satisfied on reduction.
   Also $F[a][]$ will satisfy at least 2 clauses. So this gives a *good* $(2,3)$ branch by the number of satisfied clauses (for the parameterized question).
   The branch $a = false$ will eliminate at least 3 variables since $C_2 \cup C_3 - \{\bar{a}\}$ contains at least two variables. Also the branch $a = true$ will also eliminate at least 2 variables as observed above. Thus, we have a $(2,3)$ branch for the number of variables also.
   (b) Let $C_2 \cup C_3 - \{\bar{a}\}$ contains only one variable. This implies that the clauses are of size 2, So $C_2 = C_3 = \{\bar{a}, b\}$. We cannot have $C_2 = \{\bar{a}, b\}$ and $C_3 = \{\bar{a}, \bar{b}\}$, by reduction rule 4.
      i. If $\bar{b}$ lies in C_1 we will set $b = false$ resulting in a one-way branch $F[][a,b]$.
      ii. If not, then we branch as $F[][a,b], F[a,b][]$.
      This is essentially the branching rules $3b$ and $3c$ applied together with the observation that a and b together are either both true or both false in an optimal assignment. Note that $a = false$, $b = false$ satisfies at least three clauses and eliminates at least two variables. The assignment $a = true$, $b = true$ satisfies at least three clauses and eliminates at least three variables (a, b and $C_1 - \{a\}$ during the reduction). Thus this is a $(3,3)$ branch with respect to the clauses satisfied and a $(2,3)$ branch with respect to the number of variables eliminated.
3. If a is a $(1,2)[1][2^+,2^+]$ literal, let C_2 and C_3 be the clauses containing $\bar{a}$.
   (a) If $Co(C_2, C_3)$, branch as $a = true$, apply rule $2a$ with x as $\bar{a}$.
   (b) If C_2 and C_3 contain a common literal b other than $\bar{a}$, then $\bar{b}$ must occur in a unit clause (else step 2 of the algorithm would be applicable), thus we just branch as $a = true$.
   Thus we can assume that C_2 and C_3 do not contain any common variable.
   (c) If a is a $(1,2)[1][2,2^+]$ literal, say $C_2 = \{\bar{a}, b\}$, then we branch as $F[][a]$ and $F[a,b][]$. Branching rule $3c$.
   The branch $F[][a]$ satisfies at least 3 clauses and the other branch satisfies at least 2 clauses.
   Similarly the branch $F[][a]$ eliminates at least 3 variables, since $|C_2 \cup C_3 - \{\bar{a}\}| \geq 2$. $F[a,b][]$ clearly eliminates at least 2 variables, thus giving a $(3,2)$ branch.

(d) If a is a $(1,2)[1][3^+,3^+]$ literal, then we branch as $F[a][]$ and $F[][a, C_2 - \{\bar{a}\}, C_3 - \{\bar{a}\}]$. Branching rule $5c$.
Then clearly we have a $(1,5)$ branch for variables and also a $(1,6)$ branch for clauses, (all literals in $C_2 \cup C_3 - \{\bar{a}\}$ are $(2,1)[\ldots][1]$ literals and $|C_2 \cup C_3 - \{\bar{a}\}| \geq 4$).

Thus we have proved

Theorem 1. *For a formula F in CNF in which each variable occurs in at most 3 clauses, an assignment satisfying the maximum number of clauses can be computed in time $O(1.324719^n|F|)$. Also an assignment satisfying at least k clauses, if exists, can be computed in time $O(k^2 1.324719^k + |F|)$, where $|F|$ is the length of the formula.*

6 Algorithms for general MAXSAT

In this section we present two algorithms for the general MAXSAT. One has the number of clauses as the exponent in the running time and the other has the length of the formula as the exponent in the running time. As in the previous section, we will first employ the reduction rules as far as possible and work with the reduced formula. It is also assumed that after applying every branching rule, any applicable reduction rules are applied.

6.1 A bound with respect to the number of clauses

We note that if $\{x\}$ is a unit clause then whether x is set *true* or *false*, the clause will be eliminated. This means that if x is a (k,l) literal, such that $n_u(x) = i$ (recall that $n_u(x)$ is the number of times x occurs as a unit clause in the formula) and $n_u(\bar{x}) = j$, then branching as $F[x][]$ and $F[][x]$ gives us a $(k+j, l+i)$ branch for the non-parameterized case (number of clauses eliminated), and a (k,l) branch for the parameterized case (number of clauses satisfied). In the following if we say that a branch is (r,s) for the parameterized case, we mean that in one branch r clauses are satisfied and in the other s clauses. Similarly an (r,s) branch for the non-parameterized case is a branch where r clauses are eliminated in one and s in the other. Note also that an (r,s) branch for the parameterized case is also an (r,s) branch for the non-parameterized case.

Algorithm:

1. If there is some $(1, 5+)$ or a $(2+, 3+)$ literal x, branch as $F[x][]$ and $F[][x]$. This gives *good* branches for the parameterized question (and hence for the general MAXSAT as well).
2. If there is some $(1, 3+)[2+][\ldots]$ literal x, branch as $F[][x]$ and $F[x][\{C - \{x\}\}]$, where C is the clause containing x. This is a $(3,2)$ branch for the parameterized case. This is essentially branching rule 3b.
3. If there is some $(1,4)[1][\ldots]$ literal x, branch as $F[x][]$ and $F[][x]$. Clearly, a $(1,5)$ branch for the non-parameterized case and a $(1,4)$ branch for the parameterized case.
4. If there is some $(1,3)[1][2,\ldots]$ literal x, let C denote the length 2 clause containing $\bar{x}$ (choose any one if many). Branch as $F[][x]$ and $F[x, \{C - \{\bar{x}\}\}][]$. Branching rule $3c$. Clearly a $(3,2)$ branch for the parameterized case.

5. If there is some $(2,2)[1,\ldots][\ldots]$ literal x. According to branching rule 2, we get that, if C_1 and C_2 are the clauses containing $\bar{x}$, branch as $F[x][]$ if $Co(C_1, C_2)$. Otherwise branch as $F[x][]$ and $F[][x, C_1 \cup C_2 - \{\bar{x}\}]$. Clearly a $(2,3)$ branch for the parameterized case, since $|C_1 \cup C_2 - \{\bar{x}\}| \geq 1$.
   After the above rules can no longer be applied, the formula just contains $(1,3)[1][3^+, 3^+, 3^+]$ literals and their complements, $(2,2)[2^+, 2^+][2^+, 2^+]$ literals and *3-variables*.
6. If $\tilde{y}$ is a *3-variable* which occurs in a clause containing a (2,2) literal x, then branch as $F[x][]$ and $F[][x]$. Thus, branching rule 4 gives a $(2,3)$ branch for the parameterized case.
7. If $\tilde{y}$ is a *3-variable* which occurs in a clause containing a $(3,1)[\ldots][1]$ literal x, then if $\tilde{y}$ occurs in all the 3 clauses containing x, branch as $F[][x]$. This is because y can be assigned so that at least 2 of the 3 clauses containing x are satisfied with $x = \mathit{false}$. Otherwise, branch as $F[x][]$ and $F[][x]$. This gives a $(1,5)$ branch for non-parameterized case and a $(1,4)$ branch for the parameterized case.
   Now any clause containing a *4-variable* contains only *4-variables*.
   So the subset of clauses containing *3-variable* is closed. This can be eliminated efficiently using the algorithm in Section 5. Thus the formula now contains only *4-variables*.
8. If x is a $(2,2)$ literal and some $\tilde{y}$ occurs in the two clauses containing $\bar{x}$ or in the two clauses containing x , then branch as $F[x][]$ and $F[][x]$. A $(2,3)$ branch for the parameterized case.
9. If x is a $(1,3)$ literal and any $\tilde{y}$ occurs in two or more clauses containing $\bar{x}$ then branch as $F[][x]$ and $F[x][]$. Thus, branching rule 4 gives a $(1,4)$ branch and a $(1,5)$ branch for the parameterized case and the non-parameterized case respectively.

At this state, note that if x is a $(1,3)$ literal, and C_1, C_2 and C_3 are the clauses containing $\bar{x}$, then except for $\bar{x}$ they all have disjoint variables. Also, let a_1 be a $(2,2)$ literal, and C_1 and C_2 be the clauses containing $\bar{a_1}$. Let a_2 be any other literal, then after steps 8 and 9 of the algorithm at most one of C_1 and C_2 can contain $\bar{a_2}$.

This observation gives us the following lemma.

Lemma 1. *Let $a_1, a_2, \ldots, a_l$ be $(2,2)$ literals and let $C_1, \ldots, C_k$ be precisely the clauses that contain some $\bar{a}_i$, $\quad 1 \leq i \leq l$. After step 9 of the algorithm $k \geq T(l)$ where $T(l) = \min\max\{\lceil \frac{2l}{j} \rceil, j+1\}$, where the minimum is taken over all $1 \leq j \leq l$. In particular $k \geq \lceil \sqrt{(2l + \frac{1}{4})} + \frac{1}{2} \rceil$.*

Proof. Without loss of generality, let C_1 be the clause such that $\{C_1 \cap \{a_1, a_2, \ldots, a_l\}\}$ has maximum cardinality j. Then the number of clauses needed is at least $\lceil \frac{2l}{j} \rceil$. Also by the condition imposed by step 9 of the algorithm, each of the other occurrences of the j variables must occur in different clauses, thus at least j additional clauses are needed. Thus number of clauses k will be at least $T(l) = \min_j \max\{\lceil \frac{2l}{j} \rceil, j+1\}$. It can be verified that $T(l) \geq \lceil \sqrt{(2l + \frac{1}{4})} + \frac{1}{2} \rceil$.

10. Let x be a $(1,3)[1][\ldots]$ literal and let C_1, C_2 and C_3 be the clauses containing $\bar{x}$. If C_1 contains a $(2,2)$ literal a or has size 4 or more then branch as $F[x][], F[][x, C_1 - \{\bar{x}\}], F[][x, C_2 - \{\bar{x}\}, C_3 - \{\bar{x}\}]$. This is essentially branching rule $5b$.
    (a) If y is a $(1,3)$ literal such that $\tilde{y} \in C_i$, then it is $\bar{y}$ that belongs to C_i. Since all $(1,3)$ literals are $(1,3)[1][\ldots]$ literals.
    (b) If there are p $(3,1)$ literals $u_1, \ldots, u_p$ and q $(2,2)$ literals $v_1, v_2, \ldots, v_q$ in $C_1 - \{\bar{x}\}$, then setting $C_1 - \{\bar{x}\} = false$ eliminates at least $p + T(q)$ clauses, other than C_1, C_2 and C_3. This is because, as $\bar{u_i}'s$ are present as unit clauses, setting $u_i's = false$ eliminates at least p clauses and since the $\tilde{v_i}'s$ will occur in clauses other than C_2 and C_3 (after step 9 of algorithm). setting $v_i's = false$ will eliminate another $T(q)$ clauses.

    $F[][x, \{C_1 - \{\bar{x}\}\}]$ satisfies at least 3 clauses other than C_1, C_2 and C_3, since $p + T(q) \geq 3$ for $\{C_1 - \{\bar{x}\}\}$. Thus 6 clauses are satisfied and 7 are eliminated by the assignment.

    Similarly, since $\{C_2 - \{\bar{x}\}\}$ and $\{C_3 - \{\bar{x}\}\}]$ have no common variable, the assignment $F[][x, \{C_2 - \{\bar{x}\}\}, \{C_3 - \{\bar{x}\}\}]$ satisfies 4 clauses other than C_1, C_2 and C_3. Thus 7 clauses are satisfied and 8 are eliminated by the assignment. Thus we have a $(1,7,8)$ branch for the non-parameterized case and a $(1,6,7)$ branch for the parameterized case.

    Now the clauses containing $(1,3)[1][3,3,3]$ literals and their complements form a closed subformula F_1. The remaining clauses form a closed subformula F_2 of $(2,2)$ literals. So, $F = F_1 \wedge F_2$. Rule 11 describes how to eliminate F_1 efficiently.
11. If a is a $(1,3)$ literal, then the clauses with $\tilde{a}$ will be $\{a\}, \{\bar{a}, b, c\}, \{\bar{a}, \ldots\}, \{\bar{a}, \ldots\}$. We branch as $F[][a], F[a,b][]$ and $F[a,c][b]$.

    We know that b and c will be $(3,1)[3,3,3][1]$ literals, if a is true and both b and c are false, then a can be set to $false$. For the parameterized case, $F[][a]$ and $F[a,b][]$ satisfies 3 and 4 clauses respectively, $F[a,c][b]$ satisfies 5 clauses (the unit clauses $\{a\}, \{\bar{b}\}$ and the 3 clauses containing c). Thus we have a $(3,4,5)$ branch for the parameterized case. Similarly we have a $(4,5,6)$ branch for the non-parameterized case.
12. If x is a $(2,2)[2^+, 2^+][3^+, 3^+]$ literal, let C_1 and C_2 be the clauses containing $\bar{x}$. Apply branching rule $5a$ and branch as $F[x][], F[][x, \{C_1 - \{\bar{x}\}\}]$ and $F[][x, \{C_2 - \{\bar{x}\}\}]$.

    The second and the third branch will eliminate at least $2 + T(2) = 5$ clauses each. $F[x][]$ eliminates 2 clauses and also leads to $3-variables$ in the formula; now either these variables would occur as a closed subformula or in a clause with some $(2,2)$ literal. Thus the next step would at least be a $(2,3)$ branch or better, since steps 6 of this algorithm and branching rules for $(n,3)$ MAXSAT are all *good* for the parameterized case. Thus we have a $(4,5,5,5)$ branch for the parameterized case.

13. Any $4 - variable$ a will be of the type $(2,2)[2,2^+][2,2^+]$. Let $C_1 = \{a, b\}$ be the clause of size 2 containing a. We will show how to branch with $a = false$. The case with $a = true$ is similar.
    (a) If $\tilde{b}$ is present in some clause containing $\bar{a}$. Then just branch as $F[b][a]$. By rule 8 of this algorithm, it must be that $\bar{b}$ occurs in the clause with $\bar{a}$; thus setting $a = false$ the clauses containing $\tilde{b}$ will reduce to $\{b\}, \{b, \ldots\}$ and $\{\bar{b}, \ldots\}$; thus setting $b = true$ is sufficient, and this is essentially branching rule $3a$.
    (b) Otherwise, If C_1' and C_2' are the clauses containing $\bar{b}$, then branch as $F[b][a]$ and $F[][a, b, C_1' \cup C_2' - \bar{b}]$. Clearly $F[b][a]$ eliminates 4 clauses, and the second branch eliminates the clauses containing $\bar{b}$, the unit clause $\{b\}$ and another $3(= T(2))$ clauses corresponding to $C_1' \cup C_2' - \bar{b}$. So at least 6 clauses are eliminated in the non-parameterized case and at least 5 clauses in the parameterized case.
    Thus in the worst case the branch is a $(4, 6, 4, 6)$ branch for the non-parameterized case and a $(4, 5, 4, 5)$ branch for the parameterized case.

For the non-parameterized case, the branch $(4, 5, 5, 5)$ is the most restrictive giving an $O(1.341294^m|F|)$ algorithm for MAXSAT by solving the appropriate recurrence relation.

For the parameterized case, the branch $(1, 4)$ is the most restrictive giving an $O(1.380278^k k^2 + |F|)$ algorithm by reducing the problem to the kernel and applying the above branching algorithm.

Thus we have proved the following.

Theorem 2. *For a formula F in CNF on n variables and m clauses, an assignment satisfying the maximum number of clauses can be found in time $O(1.341294^m|F|)$. Also an assignment satisfying at least k clauses of F, if exists, can be found in time $O(k^2 1.380278^k + |F|)$.*

6.2 A bound with respect to the length of the formula

In this section, we call a branching vector *good* if either it is a $(4, 11)$, $(5, 10)$, $(6, 8)$ or a $(7, 7)$ vector.

We observe that if x is a (p, q) literal, setting $x = true$ reduces the length of the formula by at least $l(x) + q$ (recall that $l(x)$ is the sum of the lengths of the clauses containing x.) Also since every variable $\tilde{x}$ is a $3^+ - variable$ after applying the reduction rules, setting a value to x reduces the length of the formula by at least 3.

Using these facts and observing that both x and $\bar{x}$ cannot be present as unit clauses together, simply branching as $F[x][]$ and $F[][x]$ gives the recurrence $T(|F|) \leq T(|F| - 3) + T(|F| - 4) + |F|$ for the running time. This gives a very simple $O(1.220745^{|F|}|F|)$ algorithm.

In what follows, we argue through a number of cases to obtain a more efficient algorithm.

Algorithm

1. If $\tilde{x}$ is a $7^+ - variable$, then branching as $F[x][]$ and $F[][x]$ gives a good branch.

2. If x is a $(3,3)$ literal, then either $l(x)$ or $l(\bar{x}) \geq 6$, since only one of x and $\bar{x}$ can appear in a unit clause. Thus, branching as $F[x][]$ and $F[][x]$ gives us a good branch.

   Now the formula contains only $(1,k)$ literals, for $k \leq 5$ or $(2,k)$ literals, for $k \leq 4$.
3. If x is a $(1,k)[\ldots][1,\ldots]$ literal, then just branch as $F[][x]$. This is from branching rule $3a$.
4. The following rules give *good* branches for $k \geq 3$.
   (a) If x is a $(1,k)[2^+][\ldots]$ literal, then just branch as $F[][x]$ and $F[x][C-\{x\}]$ where C is the 2^+-clause containing x. This is branching rule $3b$ and this gives a $(2k+1, k+4)$ branch.
   (b) If x is a $(1,k)[1][2,\ldots]$ literal, then just branch as $F[][x]$ and $F[x, C - \{\bar{x}\}][]$. C is the clause containing $\bar{x}$ of size 2. This is essentially branching rule $3c$. Clearly this gives us a $(2k+1, k+4)$ branch.
   (c) If x is a $(1,k)[1][3^+,\ldots]$ literal, and $l(\bar{x}) \geq 10$, then just branch as $F[][x]$ and $F[x][]$. This gives a $(k+1, l(\bar{x})+1)$ branch.

   So now if x is a $(1,k)$ literal, it has to be a $(1,3)[1][3,3,3]$ literal or a $(1,2)$ literal.
5. If x is a $(2,k)[\ldots][1,1,\ldots]$ literal then, we branch as $F[][x]$. This is simply the branching rule 1.
6. The following rules for eliminating $(2,k)$ literals, give *good* branches for $k \geq 3$.
   (a) If x is a $(2,k)[\ldots][1,2^+,\ldots]$ literal then, we branch as $F[x][]$ and $F[][x]$. This gives a $(4+k, 2k+1)$ branch, since x cannot appear in a unit clause.
   (b) If x is a $(2,k)[\ldots][2^+,\ldots]$ and $l(x) \geq 3$, we branch as $F[x][]$ and $F[][x]$. This gives a $(3+k, 2k+2)$ branch.
7. If x is a $(2,k)[1,1][2^+,\ldots]$ literal, branching as $F[x][]$ and $F[][x]$ gives a $(2+k, 2k+2)$ branch, which is *good* for $k \geq 4$.
8. If x is a $(2,3)[1,1][2^+,\ldots]$ literal, let C_1, C_2 and C_3 denote the clauses containing $\bar{x}$,
   (a) If $Co(C_i, C_j)$ for some $1 \leq i, j \leq 3$, branch as $F[x][]$.
   (b) If C_1, C_2 and C_3 contain only one variable other than $\bar{x}$, then $C_1 = C_2 = C_3 = \{\bar{x}, c\}$, thus we branch as $F[x, c][]$ and $F[][x]$, which is at least an $(8,8)$ branch. This is because since c will be almost a $6-variable$, setting $x = true$ makes c a $(3^+, 3^-)[1,1,1,\ldots][\ldots]$ literal, and so $c = true$ in an optimal solution.
   (c) If C_1, C_2 and C_3 contain 2 or more variables other than $\bar{x}$, then branching as $F[x][]$ and $F[][x, C_1 \cup C_2 \cup C_3 - \{\bar{x}\}]$ gives a $(5,11)$ branch.

   Thus the formula contains only $3-variables$,$(2,2)$ literals and $(1,3)[1][3,3,3]$ literals.
   (a) The rules below eliminate $(1,3)[1][3,3,3]$ literals. If x is a $(1,k)[1][3,3,3]$, and some clause containing $\bar{x}$ contains a $3-variable$ $\tilde{y}$, then if all the 3 occurrences of $\tilde{y}$ are in clauses containing $\bar{x}$ then, just branch as $F[x][]$. Otherwise branching as $F[x][]$ and $F[][x]$ gives a $(4,11)$ branch.

(b) If x is a $(1,3)[1][3,3,3]$, and some clause containing $\bar{x}$ contains two or more occurrences of a $4-variable$ $\tilde{y}$, then branching as $F[x][]$ and $F[][x]$ gives a $(4,11)$ branch.
(c) Now, if x is a $(1,3)[1][3,3,3]$ literal, then the clauses containing $\bar{x}$ contain single occurrences of $4-variables$ other than $\bar{x}$. We show that branching according to the rule $5b$, gives a $(4,16,22)$ branch. Clearly $F[x]$ reduces the length by 4, $F[][x]$ reduces the length of the formula by 10, now in F[][x], $|C_1| = 2$ and $|C_2 \cup C_3| = 4$ and since the variables in $C_1 \cup C_2 \cup C_3$ are $3-variables$, we reduce the length of the formula by at least 3 for each assignment of these.

The formula contains only $(2,2)$ literals and $3-variables$.

9. If x is a $(2,2)[3^+,3^+][2^+,2^+]$ literal or a $(2,2)[2,3^+][2,3^+]$ literal, branching as $F[x][]$ and $F[][x]$, gives a *good* branch.

   Thus the formula just contains $(2,2)[2^-,2^-][2^-,2^-]$ literals.
10. If x is a $(2,2)[1,2][2,2]$ literal. Let C denote the $2-clause$ containing x and D_1, D_2 denote the clauses containing $\bar{x}$.
    (a) If $C \cup D_1 \cup D_2 - \{x,\bar{x}\}$ contains only one variable, then the clauses containing $\tilde{x}$, must be $\{x\}, \{x,c\}, \{\bar{x},\bar{c}\}$ and $\{\bar{x},\bar{c}\}$. We branch as $F[x][c]$, it can be seen that this is a valid branch as c is a $3-variable$ or a $(2,2)$ literal.
    (b) If $C \cup D_1 \cup D_2 - \{x,\bar{x}\}$ contains three variables. Branching as $F[x][]$ and $F[C-x][x, D_1 \cup D_2 - \{\bar{x}\}]$, we get a $(5,13)$ branch. This is because $F[x][]$ reduces the length by 5, and assignment to x reduces the length by at least 4 and assignment to each of the other three variables reduces the length by at least 3.
    (c) If $C \cup D_1 \cup D_2 - \{x,\bar{x}\}$ contains two variables then the clauses containing $\tilde{x}$ can be
        i. $\{x\}, \{x,b\}, \{\bar{x},\bar{b}\}$ and $\{\bar{x},c\}$. Branching as $F[x][]$ and $F[C-x][x, D_1 \cup D_2 - \{\bar{x}\}]$, we get a $(7,10)$ branch. For if b is a $3-variable$, then it will be eliminated by setting $x = true$. Otherwise if b is a $(2,2)$ literal, then the clauses containing $\tilde{b}$ will be $\{\bar{b}\}, \{b,\ldots\}$ and $\{\bar{b},\ldots\}$, which can be directly eliminated by setting $b = false$.
        ii. $\{x\}, \{x,b\}, \{\bar{x},c\}$ and $\{\bar{x},c\}$. Branching as $F[x][]$ and $F[C-x][x, D_1 \cup D_2 - \{\bar{x}\}]$, we get a $(7,10)$ branch, since setting $x = true$, then D_1, D_2 will become unit clauses $\{c\}, \{c\}$, for which we will only have one way branch $c = true$.
11. If x is a $(2,2)[2,2][2,2]$ literal, then
    (a) if some clause containing x contains a $3-variable$ $\tilde{y}$, then if number of occurrences of $\tilde{y}$ in the clauses containing x is 1, then setting $x = true$ reduces the length of the formula by at least 8. So $F[x][]$ and $F[][x]$ gives a $(6,8)$ branch. Otherwise, if the number of occurrences of $\tilde{y}$ in the clauses containing x is 2 then setting $x = false$ gives 2 unit clauses containing $\tilde{y}$, which can be eliminated directly. So $F[x][]$ and $F[][x]$ gives a $(6,8)$ branch.

(b) Otherwise branch as $F[x][]$ and $F[][x]$ and apply rule 11 of this algorithm for each instance. Setting $x = true$ or $x = false$ reduces the length of the formula by 6, but this leads to a $(2,2)[1,..][2,2]$ literal. So, we either have a $(7,10)$ branch in the next step or a $(5,13)$ branch. Thus we a $(13,16,13,16)$ branch or a $(11,19,11,19)$ branch in the worst case.

12. The formula now contains only $3-variables$, we now apply the algorithm for $(n,3)$ MAXSAT of Section 5. Since the length of the formula l, will be equal to $3n$ (since $2^{-} - variables$ do not exist in the formula), we can eliminate the formula in $O(1.324719^{\frac{l}{3}}|F|)$ time.
The branch $(4,11)$ is the most restrictive thus giving an $O(1.105729^{|F|}|F|)$ algorithm by solving the appropriate recurrence relation.

Thus we have proved

Theorem 3. *For a formula F in conjunctive normal form, an assignment satisfying the maximum number of clauses can be found in time $O(1.105729^{|F|}|F|)$.*

7 Conclusions

It would be interesting to see how our algorithms perform in practice. Also it would be useful to identify some general branching techniques which would reduce the number of cases in the algorithms.

References

1. R. Balasubramanian, M. R. Fellows and V. Raman, 'An Improved fixed parameter algorithm for vertex cover' *Information Processing Letters* , **65** (3):163-168, 1998.
2. R. Beigel and D. Eppstein, '3-coloring in time $O(1.3446^n)$: A no-MIS Algorithm', In Proc of IEEE Foundations of Computer Science (1995) 444-452.
3. E. Dantsin, M. R. Gavrilovich E. A. Hirsch and B. Konev, 'Approximation algorithms for MAX SAT: a better performance ratio at the cost of a longer running time', PDMI preprint 14/1998, Stekolov Institute of Mathematics at St. Petersburg, 1998.
4. R. G. Downey and M. R. Fellows. Parameterized Complexity. Springer-Verlag, November 1998.
5. E.A. Hirsch, 'Two new upper bounds for SAT', Proc. of 9th Symposium on Discrete Algorithms, (1998) 521-530.
6. O. Kullmann, 'Worst-case analysis, 3-SAT decision and lower bounds: approaches for improved SAT algorithms', DIMACS Proc. SAT Workshop 1996, AMS, 1996.
7. M. Mahajan and V. Raman, 'Parameterizing above guaranteed values: MaxSat and MaxCut'. Technical Report TR97-033, ECCC Trier, 1997. To appear in Journal of Algorithms.
8. B. Monien, E. Speckenmeyer, 'Solving Satisfiability in less than 2^n steps', *Discrete Applied Mathematics* , **10** (1985) 287-295.
9. R. Niedermeier and P. Rossmanith, 'Upper bounds for Vertex Cover: Further Improved'. in Symposum on Thoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science, Springer Verlag. March (1999).
10. R. Niedermeier and P. Rossmanith, 'New Upper Bounds for MAXSAT', International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, Springer Verlag (1999).
11. R. Paturi, P. Pudlak, M. Saks, and F. Zane. 'An improved exponential-time algorithm for k-SAT', In Proc. of 39th IEEE Foundations of Computer Science (1998).

12. P. Pudlak, 'Satisfiability-algorithms and logic', In Proc. of 23rd Conference on Mathematical Foundations of Computer Science, **1450** in Lecture Notes in Computer Science Springer Verlag, August (1998) 129-141.
13. V. Raman, B. Ravikumar and S. Srinivasa Rao, 'A Simplified NP-Complete MAXSAT problem' *Information Processing Letters*, **65**, (1998) 1-6.
14. J. M. Robson, 'Algorithms for the Maximum Independent Sets'. *Journal of Algorithms* **7**, (1986) 425-440.
15. I. Schiermeyer, 'Solving 3-Satisfiability in less than 1.579^n steps', Lecture Notes in Computer Science, Springer Verlag **702**, (1993) 379-394.