Improved Exact Algorithms for Max-Sat*

Jianer Chen† Iyad A. Kanj‡

Abstract

In this paper we present improved exact and parameterized algorithms for the maximum satisfiability problem. In particular, we give an algorithm that computes a truth assignment for a boolean formula F satisfying the maximum number of clauses in time $O(1.3247^m|F|)$, where m is the number of clauses in F, and $|F|$ is the sum of the number of literals appearing in each clause in F. Moreover, given a parameter k, we give an $O(1.3695^k + |F|)$ parameterized algorithm that decides whether a truth assignment for F satisfying at least k clauses exists. Both algorithms improve the previous best algorithms by Bansal and Raman for the problem.

Key words. maximum satisfiability, exact algorithms, parameterized algorithms.

1 Introduction

The maximum satisfiability problem, abbreviated Max-Sat, is to compute for a given boolean formula F in conjunctive normal form, a truth assignment that satisfies the largest number of clauses in F. In the parameterized maximum satisfiability problem, we are given an additional parameter (positive integer) k, and we are asked to decide if F has a truth assignment that satisfies at least k clauses. It is easy to see that both Max-Sat and parameterized Max-Sat are NP-hard since they generalize the satisfiability problem [24]. According to the theory of NP-completeness [24], these problems cannot be solved in polynomial time unless P = NP.

The above fact does not diminish the need for solving these problems for their practical importance. Due to its close relationship to the satisfiability problem (Sat), Max-Sat has many applications in artificial intelligence, combinatorial optimization, expert-systems, and database-systems [5, 8, 23, 27, 28, 32, 36, 37]. Many approaches have been employed in dealing with the NP-hardness of the Max-Sat problem including approximation algorithms [3, 4, 9], heuristic algorithms [8, 9, 10, 36], and exact and parameterized algorithms [7, 12, 25, 31, 33].

In this paper we focus our attention on finding exact solutions for the Max-Sat problem. Acknowledging the apparent inevitability of an exponential time complexity for NP-hard problems, this line of research seeks designing efficient exponential time algorithms that improve significantly on the straightforward exhaustive search algorithms. Numerous NP-hard combinatorial problems were studied from this point of view. Examples of exponential time algorithms for NP-hard optimization problems include the Independent Set problem [11, 35], Dantsin et al.'s $O(1.481^n)$ time deterministic algorithm for the 3-Sat problem [17] (n is the number of variables in the formula),

*A preliminary version of this paper was presented at the *5th Latin American Symposium on Theoretical Informatics* (LATIN), and appeared in *Lecture Notes in Computer Science*, vol. 2286, (2002), pp. 341-355.

†Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, `chen@cs.tamu.edu`. This work was supported in part by the National Science Foundation under the Grant CCR-0000206.

‡Corresponding author. School of CTI, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604, `ikanj@cs.depaul.edu`. This work was supported in part by DePaul University Competitive Research Grant.

and Eppstein's $O(1.3289^n)$ time algorithm for the 3-COLORING problem [21] (n is the number of vertices in the graph). A closely related line of research to the above one that has been receiving a lot of attention recently, is the area of fixed-parameter tractability [19], which has found many applications in fields like databases, artificial intelligence, logic, and computational biology [20]. A famous problem that belongs to this category is the VERTEX COVER problem. Recently, there has been extensive research done aiming at improving the complexity of parameterized algorithms that solve the VERTEX COVER problem [6, 15, 16]. Other examples of parameterized problems include PLANAR DOMINATING SET [1, 2], k-DIMENSIONAL MATCHING [13, 19], and CONSTRAINT BIPARTITE VERTEX COVER [14, 22].

The MAX-SAT problem has played a significant role in both these lines of research. Considerable efforts have been paid trying to lower the worst-case complexity for the problem. Cai and Chen [12] presented an algorithm of running time $O(2^{2ck}cm)$ for the MAX-c-SAT problem. Mahajan and Raman [31] improved on that bound by presenting an algorithm of running time $O(cm+ck\phi^k)$ ($\phi = (1+\sqrt{5})/2 \approx 1.618$). Mahajan and Raman also gave an algorithm of running time $O(|F|+k^2\phi^k)$ for the parameterized MAX-SAT problem. They described how their algorithm for parameterized MAX-SAT implies an algorithm for the parameterized MAX-CUT problem. Niedermeier and Rossmanith [33] presented an algorithm of running time $O(1.3995^k k^2 + |F|)$ for parameterized MAX-SAT. For MAX-SAT they gave algorithms of running time $O(1.3803^m|F|)$ and $O(1.1272^{|F|}|F|)$, where m and $|F|$ are as defined above. Bansal and Raman [7] improved Niedermeier and Rossmanith's result and presented algorithms of running time $O(1.3802^k k^2 + |F|)$ for parameterized MAX-SAT, and $O(1.341294^m|F|)$ and $O(1.105729^{|F|}|F|)$ for MAX-SAT. Recently, Gramm et al. [25, 26] considered the MAX-2-SAT problem, a special case of the MAX-SAT problem. They gave an $O(1.148699^m|F|)$ algorithm for the problem. Their algorithm also implies better algorithms for the parameterized MAX-CUT problem.

In this paper, we further improve on Bansal and Raman's algorithms. More specifically, we give an $O(1.3247^m|F|)$ algorithm for the MAX-SAT problem, and an $O(1.3695^k + |F|)$ algorithm for the parameterized MAX-SAT problem. Our techniques are basically similar to the previous ones in the careful case-by-case analysis, which seems unavoidable when designing exact algorithms for NP-hard problems using the search tree method [6, 7, 11, 14, 15, 16, 22, 25, 33, 35]. However, we add a number of nice observations to show how to make use of certain structures in the formula to yield more efficient algorithms and reduce the case-by-case combinatorial analysis. For instance, it has been observed before that when an instance of the MAX-SAT problem contains a literal with few occurrences, the case can be handled more efficiently. For this purpose, during a certain phase of our algorithm, we try to enforce the invariant that the formula contains a literal with few occurrences by using a subroutine that we call **Create-Low-Literal**. This results in a more efficient running time for that phase and for the whole algorithm as well. Also, we introduce a branching rule, **BranchRule 2**, that captures uniformly and systematically several branching cases, and renders them more efficient.

The paper is organized as follows. In Section 2, we present the basic notations, definitions, and transformation rules. In Section 3, we present our algorithm for the parameterized MAX-SAT problem. In Section 4, we give an algorithm for the MAX-SAT problem based on the algorithm in Section 3 for parameterized MAX-SAT, and a new technique that we introduce in Section 4. We conclude the discussion in Section 5 with some remarks and further research directions.

2 Definitions and background

2.1 Definitions and notations

We assume familiarity with the basic concepts of propositional logic, and we use a notation that is similar to that of [33]. The formula F is assumed to be given as a set of clauses in conjunctive normal form (CNF). Each clause is a disjunction of literals, where each literal is a variable or its negation. A variable will be denoted by an alphabetical character (e.g., x), its negation by its name with bar on the top of it (e.g., $\bar{x}$), and a literal by the variable name with a tilde sign on the top of it (e.g., $\tilde{x}$). A literal $\tilde{x}$ occurs *positively* (resp. *negatively*) in a clause C if x (resp. $\bar{x}$) occurs in C. Two occurrences of a literal $\tilde{x}$ have *opposite* signs, if in one of the occurrences $\tilde{x}$ occurs positively, and negatively in the other; otherwise, the two occurrences have the *same* sign. A literal $\tilde{x}$ is said to *occur with* another literal $\tilde{y}$, if there is a clause C in F containing both $\tilde{x}$ and $\tilde{y}$. A literal $\tilde{x}$ is said to be *dominated by* a literal $\tilde{y}$ if all occurrences of $\tilde{x}$ are with $\tilde{y}$ (we also say that $\tilde{y}$ *dominates* $\tilde{x}$). A *truth assignment* to F is a function that assigns every variable in F a boolean value *true* or *false*. For simplicity, we will denote from now on the boolean value *true* by 1 and *false* by 0. A *partial* assignment τ to F is an assignment to a subset of the variables in F. If τ is a partial assignment to the variables $\{x_1, \ldots, x_r\}$ in F, and if τ assigns the value 1 to the variables $\{x_1, \ldots, x_i\}$, and the value 0 to the variables $\{x_{i+1}, \ldots, x_r\}$, where $i \leq r$, we denote by F/τ, or $F[x_1, \ldots, x_i][x_{i+1}, \ldots, x_r]$, the formula resulting from F by replacing the variables in $\{x_1, \ldots, x_i\}$ in F by the value 1 and $x_{i+1}, \ldots, x_r$ by the value 0, and eliminating from F all clauses whose values have been determined upon that assignment.

The maximum number of simultaneously satisfiable clauses in F will be denoted by $maxsat(F)$. We call a partial assignment τ *safe*, if there is a truth assignment $\tau^{'}$ to F such that τ and $\tau^{'}$ agree on the variables $\{x_1, \ldots, x_r\}$ and $\tau^{'}$ satisfies $maxsat(F)$ clauses. In such case we also call the formula F/τ a safe formula.

A *subformula* H of F is a subset of clauses of F. A subformula H is said to be *closed* if no literal in H appears outside H. The *length* of a clause C is simply the number of literals in C, denoted $|C|$. The length of F, denoted $|F|$, is $\sum_{C \in F} |C|$. If a clause C has length r then C is called an r-clause. A literal $\tilde{l}$ is called an (i, j) literal, if $\tilde{l}$ occurs exactly i times positively and j times negatively. Similarly, $\tilde{l}$ is an (i^+, j^+) literal if it occurs at least i times positively and at least j times negatively. We can define in a similar fashion (i^-, j^-), (i, j^+), (i, j^-), (i^+, j), (i^+, j^-), (i^-, j), (i^-, j^+) literals. Without loss of generality, we can assume that each literal $\tilde{l}$ in F occurs at least as many times positively as negatively. If this is not the case, the literal $\tilde{l}$ can be renamed so that the above statement holds (i.e., if $\tilde{l}$ does not satisfy the above statement, we substitute $\tilde{l}$ with $\tilde{l'}$, where $l' = \bar{l}$, and now $\tilde{l'}$ satisfies the above statement). A formula F is called *simple*, if negative literals occur only in 1-clauses, and each pair of variables occurs together in at most one clause.

2.2 Transformation rules

A transformation rule is a rule that is applied to a formula F to transform it into a simpler formula G such that a solution of G can be mapped back to a solution of F. We describe next some standard transformation rules that can be carried in linear time, most of them appear in the literature [7, 31, 33], and their correctness can be easily verified.

TransRule 1. *Pure literal*

If $\tilde{x}$ is an $(i, 0)$ (resp. $(0, i)$) variable in F, let G be the formula resulting from F by assigning x the value 1 (resp. 0) and eliminating all satisfied clauses. Work on G with $maxsat(G) = maxsat(F) - i$.

TransRule 2. *Dominating unit-clause*

If an (i, j) literal $\tilde{x}$ occurs positively (resp. negatively) i' times, where $j \leq i' \leq i$ (resp. $i \leq i' \leq j$), in unit clauses, assign x the value 1 (resp. the value 0), eliminate all clauses containing x (resp. $\bar{x}$), and work on the remaining formula G with $maxsat(G) = maxsat(F) - i$ (resp. $maxsat(G) = maxsat(F) - j$).

TransRule 3. *Resolution*

If $F = (x \vee p_1 \vee \ldots \vee p_r) \wedge (\bar{x} \vee q_1 \vee \ldots \vee q_s) \wedge H$, where H does not contain $\tilde{x}$, then work on $G = (p_1 \vee \ldots \vee p_r \vee q_1 \vee \ldots \vee q_s) \wedge H$, with $maxsat(G) = maxsat(F) - 1$.

TransRule 4. *Reduction to problem kernel* [31]

Given a formula F and a positive integer k, then in linear time, we can compute a formula G and a positive integer $k' \leq k$ with $|G| \in O(k'^2)$, such that F has an assignment satisfying at least k clauses if and only if G has an assignment satisfying at least k' clauses. Moreover, such an assignment for F is computable from an assignment for G in linear time.

We describe briefly how the reduction to problem kernel method works. Let F be a formula with m clauses and k a positive integer. It is well known that if $m \geq 2k$ then an assignment to F satisfying at least k clauses always exists, and can be found in $O(|F|)$ time (see for instance Proposition 5 in [31]). Thus, we can assume that $m \leq 2k$. Let F_1 be the subformula of F consisting of all clauses that contain more than k literals. Suppose that F_1 contains b clauses. Let F_2 be the set of remaining clauses in F. Instead of working on the instance F trying to satisfy k clauses, we can equivalently work on the instance F_2 and try to satisfy $k - b$ clauses. To see why this is true, observe that if k clauses can be satisfied in F then $k - b$ clauses can be satisfied in F_2. Conversely, suppose that we satisfied $k - b$ clauses in F_2, then, without loss of generality, we can assume that there are at most $k - b$ literals that have been assigned truth values in F_2. Now F_1 contains b clauses each containing at least $k - (k - b) = b$ unassigned literals. It is easy to see that all clauses in F_1 can be satisfied in this case. Simply, pick a literal in the first clause in F_1 and assign it a value so that to satisfy the clause. Then repeat this process until the last clause is reached. Since each clause contains at least b literals and there are b clauses in F_1, we will always find a literal in the next clause that have not been assigned any value yet. The above reduction can be carried out in linear time with the use of a suitable data structure. (For a more detailed proof see Lemma 2 in [31].) Finally, note that since F_2 contains at most $2k$ clauses each containing at most k literals, we have $|F_2| = O(k^2)$.

3 An algorithm for parameterized Max-Sat

Recall that in the parameterized Max-Sat problem we are given a boolean formula F and a positive integer k, and we are asked to decide if there is a truth assignment for F satisfying at least k clauses. By **TransRule 4**, we can assume that $|F| = O(k^2)$. We will also assume that no clause contains more than one occurrence of each literal. This assumption is justified since if a literal occurs in a clause C more than once, then either all occurrences of the literal have the same sign, or at least two of them have opposite signs. In the former case all the occurrences can be removed and replaced with a single occurrence, and the resulting formula is equivalent to the original one. In the latter case the clause C will always be satisfied, so C can be removed, and the formula and the parameter can be updated accordingly.

The execution of our algorithm is recursive and is depicted by a search tree (branching tree). A node in the search tree represents a boolean formula and its children are the boolean formulas resulting from applying to the formula the first branching case that applies. By branching at a formula F in the search tree we mean replacing F with formulas of the form $F[x_1^1, \ldots, x_{r_1}^1][y_1^1, \ldots, y_{s_1}^1], \ldots, F[x_1^l, \ldots, x_{r_i}^l][y_1^l, \ldots, y_{s_i}^l]$, according to one of the branching rules of the algorithm, and then working on each of the formulas recursively. This technique is similar to the Davis-Putnam procedure [18]. If at any time in the algorithm F becomes empty while the parameter k is still positive, then we stop and report that no truth assignment for F satisfying k clauses exists. Also, at the beginning of each stage the algorithm applies the above transformation rules until they are no longer applicable.

Let $C(k)$ be the number of leaves in the search tree of our algorithm looking for an assignment satisfying k or more clauses in F. If we branch at a certain node x in the search tree by reducing the parameter k' by k'_1, k'_2, ..., k'_r, in each branch respectively, where $k'_1 \leq k'_2 \leq \ldots \leq k'_r$, then the following recurrence relation for the size of the search tree $C(k')$ rooted at x holds:

$$C(k') = \sum_{i=1}^{r} C(k' - k'_i)$$

It is well known that the solution to the above recurrence (see for instance [30]) is $C(k') = O(\alpha^{k'})$, where α is the unique positive root of the characteristic polynomial $p(x) = x^{k'_r} - x^{k'_r - k'_1} - \ldots - x^{k'_r - k'_{r-1}} - 1$. Now the size of the whole search tree will be $O(\alpha_{max}^k)$, where α_{max} is the largest root among all roots of the characteristic polynomials resulting from the branching cases of the algorithm. The running time of the algorithm will be $O(\alpha_{max}^k k^2 + |F|)$, since $O(|F|)$ time is needed to reduce $|F|$ to a boolean formula whose length is $O(k^2)$, and along each root-leaf path in the searching tree we spend $O(k^2)$ time. Using a technique introduced by Niedermeier and Rossmanith [34], we can get rid of the size of the kernel in the running time of the algorithm. The running time of the algorithm becomes $O(\alpha_{max}^k + |F|)$.

The following general branching rules prove to be useful in our algorithm.

BranchRule 1. If $\tilde{x}$ is an $(i, 1)$ literal, and $p_1, \ldots, p_r, \bar{q}_1, \ldots, \bar{q}_s$ occur with $\bar{x}$ in a clause C, then branch as $F[x][]$ and $F[q_1, \ldots, q_s][x, p_1, \ldots, p_r]$.

To see why the above branching rule is correct, observe that if the formula $F[p_i][]$ $(1 \leq i \leq r)$ or $F[][q_j]$ $(1 \leq j \leq s)$ is safe, then so is $F[x, p_i][]$ or $F[x][q_j]$ for any i or j, which are subbranches of the branch $F[x][]$. Hence when we branch as $F[][x]$, we can assume that $p_i = 0$ and $q_j = 1$, and branch as $F[q_1, \ldots, q_s][x, p_1, \ldots, p_r]$.

BranchRule 2. Let τ_1 be a partial assignment to F satisfying i clauses such that $F/\tau_1 = C_1 \wedge \ldots \wedge C_r \wedge G$, and τ_2 be a partial assignment to F satisfying j clauses such that $F/\tau_2 = C_1^{'} \wedge \ldots \wedge C_s^{'} \wedge G$, where $i \geq s + j$ and G is the maximal common subformula between F/τ_1 and F/τ_2. If there is a branching rule in which we branch as F/τ_1 and F/τ_2, then it suffices to branch as F/τ_1.

Proposition 3.1 BranchRule 2 *is correct.*

PROOF. To prove the proposition it suffices to prove that if F/τ_2 is safe then so is F/τ_1. Suppose that F/τ_2 is safe and let τ be a truth assignment to F satisfying $maxsat(F)$ clauses that agrees with τ_2. Suppose that τ satisfies l clauses in G. Then τ satisfies at most $l + s + j$ clauses in F, and hence, $maxsat(F) \leq l + s + j$. Now consider the truth assignment $\tau^{'}$ to F that agrees with τ on G and agrees with τ_1 (it does not matter what $\tau^{'}$ assigns to the other variables in F). Now $\tau^{'}$ satisfies at least $l + i$ clauses. Since $i \geq j + s$, $\tau^{'}$ satisfies at least $l + j + s \geq maxsat(F)$ clauses. It follows that $\tau^{'}$ is a truth assignment to F satisfying $maxsat(F)$ clauses that agrees with τ_1, and τ_1 is safe. □

The skeleton of the algorithm is described in Figure 1. We assume that after each application of a transformation rule or a branching case the formula F, the parameter k, and the truth assignment τ, are updated accordingly. In Step 4, the algorithm picks the first branching case among **Case 3.1-Case 3.13**, given below, that applies. The requirement that the cases should be considered in the given order plays a crucial role in the correctness of the algorithm. For the sake of clarity, for each branching case we will write the recurrence relation resulting from that case next to it.

```
Algorithm. Parameterized MAX-SAT
Input: a boolean formula $F$ containing $m$ clauses and a positive integer $k$
Output: a truth assignment $\tau$ satisfying at least $k$ clauses in $F$
        or report that no such a truth assignment exists

1. if $m > 2k$ then
   1.1. let $\tau$ be any assignment to the variables in $F$;
   1.2. if $\tau$ satisfies $\geq k$ clauses then return $\tau$
        else return $\tau'$, where $\tau'$ assigns the variable $x$ 1 if and only if $\tau$ assigns $x$ the value 0;
2. $\tau = \emptyset$;
3. apply TransRule 4 to $F$;
4. while ($F \neq \emptyset$) and ($\tau$ satisfies $< k$ clauses in $F$) do
      if any of TransRule 1-3 is applicable then apply it to $F$
      else branch according to the first case in Case 3.1-Case 3.13 that applies;
5. if $\tau$ satisfies fewer than $k$ clauses then
        report that no assignment to $F$ satisfies $\geq k$ clauses;
   else return($\tau$);
```

Figure 1: An algorithm for parameterized MAX-SAT

Case 3.1. There is an (i,j) literal $\tilde{x}$ with $i+j \geq 6$, or $i+j=5$ and $i,j>1$. Branch as $F[x][]$ and $F[][x]$. This gives a worst case recurrence $C(k) \leq C(k-1)+C(k-5)$.

Case 3.2. There is a $(3,1)$ or $(4,1)$ literal $\tilde{x}$ such that $\bar{x}$ does not occur as a unit clause. Let $\tilde{y}$ be a literal occurring with $\bar{x}$, and suppose that y occurs with $\bar{x}$ (the analysis in the other case is the same). By **BranchRule 1**, branch as $F[x][]$ and $F[][x,y]$. $C(k) \leq C(k-3)+C(k-2)$

Excluding **Case 3.1** and **Case 3.2**, we can assume that the formula contains only $(2,2)$ literals and $(4^-,1^-)$ literals, and all $(4,1)$ and $(3,1)$ literals have their negations as unit clauses.

Case 3.3. There is a $(2,2)$ literal $\tilde{y}$ that occurs with a $(2,1)$ literal $\tilde{x}$. Assume that y occurs with x (the case is the same if y occurs with $\bar{x}$). Branch as $F[y][]$ and apply either **TransRule 1** or **TransRule 3** to $\tilde{x}$, and $F[][y]$. $C(k) \leq C(k-3)+C(k-2)$

Case 3.4. There are two $(2,1)$ literals $\tilde{x}$ and $\tilde{y}$ such that $\bar{y}$ occurs with $\bar{x}$. Branch as $F[x][]$ and $F[y][x]$. $C(k) \leq C(k-2)+C(k-3)$

Case 3.5. There is a $(2,1)$ literal $\tilde{x}$ such that $\bar{x}$ does not occur as a unit clause. Let C be the clause containing $\bar{x}$. We distinguish the following subcases.

Subcase 3.5.1 $|C| \geq 3$. If there is a $(3^+,1)$ literal $\tilde{y}$ that occurs in C, let $\tilde{t}$ be another literal that occurs in C. Assume that t occurs in C (the analysis is the same in the case $\bar{t}$ occurs in C). Since $\tilde{y}$ is a $(3^+,1)$ literal, by **Case 3.2**, y occurs with $\bar{x}$. By **BranchRule 1**, we can branch as $F[x][]$ and $F[][x,y,t]$. Noting that $\bar{y}$ occurs as a singleton, and hence, in a different clause than $\bar{t}$, we branch with recurrence $C(k) \leq C(k-2)+C(k-3)$. If there is no $(3^+,1)$ literal occurring in C, then all literals in C must be $(2,1)$ literals. Moreover, since **Case 3.4** was excluded, all $(2,1)$ literals except x occur positively in C. Let $\tilde{r}$ and $\tilde{s}$ be two $(2,1)$ literals such that r and s occur in C. By **BranchRule 1**, we can branch as $F[x][]$ and $F[][x,r,s]$. Since **Case 3.4** was eliminated, $\bar{r}$ and $\bar{s}$ occur in different clauses. $C(k) \leq C(k-2)+C(k-3)$

Subcase 3.5.2. C is a 2-clause. Let $\tilde{y}$ be the literal occurring with $\bar{x}$. Since all the previous cases were excluded, it must be the case that y occurs with $\bar{x}$. Observe that when $y=1$, x becomes a pure variable, and hence can be assigned the value 1. It follows that branch $F[y][]$ is equivalent to $F[y,x][]$. Moreover, since $\tilde{y}$ is a $(2^+,1)$ literal, by **BranchRule 2**, the two branches $F[y,x][]$ and $F[x][y]$ reduce to $F[y,x][]$. Since we can always branch as $F[y][]$ and $F[x][y]$ and $F[][x,y]$, and since $F[y][]$ reduces to $F[y,x][]$, and $F[y,x][]$ and $F[x][y]$ reduce to $F[y,x][]$ (from the above discussion), it follows that we can always branch as $F[y,x][]$ and $F[][x,y]$. $C(k) \leq C(k-3)+C(k-2)$

Case 3.6. For every $(2,1)$ literal $\tilde{x}$, $\bar{x}$ ocurs as a unit clause. We distinguish the following subcases.

Subcase 3.6.1 There is a $(4,1)$ literal $\tilde{y}$ that occurs with x. Since **Case 3.2** was excluded, y has to occur with x. Branch as $F[y][]$ and apply **TransRule 1** or **TransRule 3** to $\tilde{x}$, and $F[][y]$. $C(k) \leq C(k-5)+C(k-1)$

Subcase 3.6.2 There is a $(2^+,1)$ literal $\tilde{y}$ that occurs with x. Since **Case 3.2** and **Case 3.5** were excluded, y occurs with x. Let C be a clause containing x and y. If C is a 2-clause, branch as $F[y][]$ and apply **TransRule 1** or **TransRule 3** to $\tilde{x}$, and $F[x][y]$. This is correct since by **BranchRule 2**, branching as $F[][x,y]$ and $F[x][y]$ reduces to branching as $F[x][y]$. Thus, we branch with the recurrence relation $C(k) \leq C(k-3)+C(k-3)$. Observe that since **TransRule 2**

was applied, no clause containing x can be a unit clause. Thus, we can assume at this point that both clauses containing x have cardinality at least three. Now let t be another variable occurring with x in C and let z be another variable occurring with x in the other clause. Note that since all the previous cases were excluded, we must have $\bar{y}$, $\bar{t}$, and $\bar{z}$ occurring as unit clauses. Branch as $F[][x]$ and $F[x][z,t,y]$. To see why this is true, note that by **BranchRule 2**, branching as $F[x,l][]$ and $F[l][x]$, where $l \in \{y,t,z\}$, reduces to branching as $F[l][x]$ which is a subbranch of $F[][x]$. $C(k) \leq C(k-1) + C(k-5)$

Now we can assume that we do not have any $(2,1)$ literals. Note that in the previous cases when we had a $(2,1)$ literal, we were always able to branch with a worst case recurrence relation $C(k) \leq C(k-1) + C(k-5)$. Suppose that there is a $(4,1)$ literal $\tilde{x}$. Clearly, since **Case 3.2** was excluded, $\bar{x}$ occurs as a unit clause.

Case 3.7. There is a literal $\tilde{y}$ occurring with x such that either $\tilde{y}$ is a $(3,1)$ literal, or a $(4,1)$ literal that has multiple occurrences with x, or a $(2,2)$ literal that is not dominated by x. Branch as $F[x][]$ and $F[][x]$. In the first branch four clauses are satisfied and y becomes a $(2^-,1^-)$ literal allowing a further branch with a recurrence relation $C(k) \leq C(k-1) + C(k-5)$ according to one of the cases **Case 3.3-Case 3.6**. In the second branch one clause is satisfied. $C(k) \leq C(k-5) + C(k-9) + C(k-1)$

Case 3.8. There is a $(2,2)$ literal y that is dominated by x. Branch as $F[y][]$ and $F[][y]$ and in both branches $\tilde{x}$ becomes a $(2,1)$ literal allowing a further branch with a recurrence $C(k) \leq C(k-1) + C(k-5)$. $C(k) \leq 2C(k-3) + 2C(k-7)$

Excluding **Case 3.7** and **Case 3.8**, no $(2,2)$ or $(3,1)$ literal occurs with a $(4,1)$ literal, and every two $(4,1)$ literals can have at most one common occurrence.

Case 3.9. All literals are $(4,1)$ literals and the formula is simple. Let $\tilde{x}$ be a $(4,1)$ literal and let C_1 be the clause with minimum cardinality containing x. Let $i = |C_1|$. We distinguish two subcases.

Subcase 3.9.1. $i \leq 5$. Let $y_1, \dots, y_{i-1}$ be the variables occurring with x in C_1. Note that by **BranchRule 2**, $F[x][]$ and $F[][x, y_1, \dots, y_{i-1}]$ reduce to $F[x][]$. Thus, we can branch as $F[x][]$ and $F[y_1][x]$ and $F[y_2][x, y_1]$ and $\dots$ and $F[y_{i-1}][x, y_1, \dots, y_{i-2}]$. We get a worst case recurrence $C(k) \leq C(k-4) + C(k-5) + C(k-6) + C(k-7) + C(k-8)$.

Subcase 3.9.2. $i > 5$. Let C_1, …, C_4 be the clauses containing x, and suppose that besides x, C_1 contains the variables $x_1^1, \dots, x_1^p$, C_2 the variables $x_2^1, \dots, x_2^q$, C_3 the variables $x_3^1, \dots, x_3^r$, and C_4 the variables $x_4^1, \dots, x_4^s$, where p, q, r, $s > 4$. Branch as $F[][x]$ and $F[x][x_1^1, \dots, x_1^p]$ and $F[x][x_2^1, \dots, x_2^q]$ and $F[x][x_3^1, \dots, x_3^r, x_4^1, \dots, x_4^s]$. To justify the above branch, note that if three variables other than x in three distinct clauses in $C_1, \dots, C_4$ have the value 1, then by **BranchRule 2**, we can safely set x to 0, and the branch becomes a subbranch of $F[][x]$. Thus, either we branch as $F[][x]$, or $F[x][]$ and no three clauses among $C_1, \dots, C_4$ have variables other than x set to 1. That is, we branch as $F[][x]$ and $F[x][x_1^1, \dots, x_1^p]$ (i.e., $x = 1$ but no other variable in C_1 is 1), and $F[x][x_2^1, \dots, x_2^q]$ (i.e., $x = 1$ and no other variable in C_2 is 1), and $F[x][x_3^1, \dots, x_3^r, x_4^1, \dots, x_4^s]$ (in this case we know that at least one variable in C_1 is set to 1 and at least one variable in C_2 is set to 1, and hence, by our assumption, the variables that occur with x in C_3 and C_4 can be set to 0). Since all the negations of the variables occur in unit clauses, we get a worst case recurrence $C(k) \leq C(k-1) + 2C(k-8) + C(k-12)$.

Now we can assume that the resulting formula does not contain $(2,1)$ and $(4,1)$ literals.

Case 3.10. There is a $(2,2)$ literal $\tilde{x}$. Clearly since **TransRule 2** was applied, not both occurrences of x or $\bar{x}$ are in unit clauses. Branch as $F[x][]$ and $F[][x]$. Since not both occurrences of x are in unit clauses, there must exist a literal $\tilde{y}$ that occurs with x. When we branch as $F[x][]$, the number of occurrences of $\tilde{y}$ is reduced by at least 1 and at most 2. Thus, $\tilde{y}$ becomes a $(2^-,1^-)$ literal in the resulting formula allowing either one of **TransRule 1** through **TransRule 3** to be applied, or a further branch with a recurrence $C(k) \leq C(k-1) + C(k-5)$ according to one of the cases **Case 3.3-Case 3.6**. The case is similar when we branch as $F[][x]$. We get a worst case recurrence $C(k) \leq 2C(k-3) + 2C(k-7)$.

Now we can assume that all literals are $(3,1)$ literals.

Case 3.11. There are two literals $\tilde{x}$ and $\tilde{y}$ that occur three times together. Since by **BranchRule 2** $F[x][y]$ and $F[y][x]$ reduce to $F[x][y]$, we can branch as $F[x][]$ and apply **TransRule 1** to $\tilde{y}$, and $F[][x,y]$. $C(k) \leq C(k-4) + C(k-2)$

Case 3.12. There are two literals $\tilde{x}$ and $\tilde{y}$ that occur twice together. Let C be the clause that contains y and does not contain x. Let t be a variable that occurs with y in C. Since **TransRule 2** was applied, such a variable must exist; otherwise, there will be two unit clauses one containing y and the other $\bar{y}$ (such case has been already taken care of by **TransRule 2**). Branch as $F[][y]$ and $F[y][x,t]$. The reason is that by **BranchRule 2**, $F[y][]$ and $F[y][x]$ reduce to $F[y][x]$ and $F[t,x][y]$ and $F[y,t][x]$ reduce to $F[t,x][y]$ which is a subbranch of $F[][y]$. $C(k) \leq C(k-1) + C(k-5)$

Case 3.13. F is a simple formula. Let $\tilde{x}$ be a $(3,1)$ literal and let C_1 be the clause with minimum cardinality containing x. Let $i = |C_1|$. As in **Case 3.9**, we distinguish two subcases.

Subcase 3.13.1. $i \leq 3$. Let y_i, y_{i-1} be the variables occurring with x in C_1. Branch as $F[x][]$ and $F[y_{i-1}][x]$ and $F[y_i][x, y_{i-1}]$ (in case y_i exists). $C(k) \leq C(k-3) + C(k-4) + C(k-5)$

Subcase 3.13.2. $i > 3$. Let C_1, C_2, C_3 be the clauses containing x, and suppose that besides x, C_1 contains the variables $x_1^1, \ldots, x_1^p$, C_2 the variables $x_2^1, \ldots, x_2^q$, and C_3 the variables $x_3^1, \ldots, x_3^r$, where p, q, $r > 2$. Branch as $F[][x]$ and $F[x][x_1^1, \ldots, x_1^p]$ and $F[x][x_2^1, \ldots, x_2^q, x_3^1, \ldots, x_3^r]$. Basically, this case is similar to **Subcase 3.9.2**. If two variables other than x in two distinct clauses in $C_1, \ldots, C_3$ have the value 1, then by **BranchRule 2**, we can safely set x to 0, and the branch becomes a subbranch of $F[][x]$. Thus, either we branch as $F[][x]$, or $F[x][]$ and no two clauses among $C_1, \ldots, C_3$ have variables other than x set to 1. $C(k) \leq C(k-1) + C(k-6) + C(k-9)$

Theorem 3.2 *Given a boolean formula F and a parameter k, then in time $O(1.3695^k + |F|)$ we can either compute a truth assignment for F satisfying at least k clauses, or we can report that such assignment does not exist.*

Proof. At each stage of the algorithm, either one of the above transformation rules, or one of the cases **Case 3.1-Case 3.13** is applicable to F. It takes linear time to apply one of the transformation rules or to update F after applying a branching rule. In all the above branching rules we branch with a recurrence relation not worse than $C(k) \leq 2C(k-3) + 2C(k-7)$. Thus, the size of the branching tree is not larger than $O(\alpha^k)$ where $\alpha \approx 1.3695$ is the unique positive root of the characteristic polynomial $x^7 - 2x^4 - 2$ associated with the recurrence $C(k) \leq 2C(k-3) + 2C(k-7)$. From the above discussion, the running time of the algorithm is $O(1.3695^k k^2 + |F|)$. Using the

```
Create-Low-Literal

Precondition: $F$ only contains $(2,2)$ and $(3,1)$ literals with at least one $(2,2)$ literal
              such that TransRule 1 through TransRule 3 are not applicable
Postcondition: $F$ contains a $(2^-,1^-)$ literal

1.  let $\tilde{y}$ be a $(2,2)$ literal;
2.  branch as $F[y][]$ and $F[][y]$;
```

Figure 2: The subroutine Create-Low-Literal

technique introduced in [34], the running time of the algorithm becomes $O(1.3695^k + |F|)$. □

Theorem 3.2 is an improvement on Bansal and Raman's $O(1.3802^k k^2 + |F|)$ algorithm for parameterized Max-Sat [7].

4 An algorithm for Max-Sat

We say a literal has *low occurrence* if it is a $(2^-,1^-)$ literal. From the above discussion we can observe the importance of having literals with low occurrences. Basically, if we have a literal with low occurrence, then either the literal is a $(1^-,1^-)$ literal and we can apply one of the transformation rules to reduce the parameter directly (like **TransRule 1** or **TransRule 3**), or the literal is a $(2,1)$ literal, and we can always branch with a worst case recurrence $C(k) \leq C(k-1)+C(k-5)$. For the non-parameterized case (i.e., for the general Max-Sat problem where the parameter is the total number of clauses), this recurrence becomes $C(m) \leq C(m-1)+C(m-5)$. Note that the difference between the parameterized Max-Sat and Max-Sat, is that in the parameterized Max-Sat the parameter is reduced by i only when i clauses are satisfied. However, for the Max-Sat problem, the parameter m is reduced by i when i clauses are eliminated (that is, their values have been determined). Hence, a branch of the form $C(k) \leq C(k-i)+C(k-j)$ for the parameterized case implies directly a branch of the form $C(m) \leq C(m-i)+C(m-j)$ for the general Max-Sat. The idea then becomes to take advantage of literals of low occurrences. Since the existence of a literal with low occurrence is not always guaranteed, we try to enforce it after each branching case of our algorithm. To do that, we will use a subroutine called **Create-Low-Literal**. This subroutine assumes that the formula contains only $(3,1)$ and $(2,2)$ literals, with at least one $(2,2)$ literal, such that none of **TransRule 1** through **TransRule 3** is applicable, and it guarantees that a $(2^-,1^-)$ literal exists in F after its termination. We give this subroutine in Figure 2.

Proposition 4.1 *If the subroutine* **Create-Low-Literal** *is called on a formula F containing only $(2,2)$ and $(3,1)$ literals, with at least one $(2,2)$ literal, such that* **TransRule 1** *through* **TransRule 3** *are not applicable, then when the subroutine terminates, the invariant that F contains a $(2^-,1^-)$ literal is satisfied. Moreover, the subroutine branches with the recurrence $C(m) \leq 2C(m-2)$.*

Proof. It is easy to see the correctness of the above proposition. The subroutine **Create-Low-Literal** works by picking a $(2,2)$ literal $\tilde{y}$ and branching at it. The existence of such a literal is guaranteed by the precondition of the subroutine. Since **TransRule 2** is not applicable, we know that both y and $\bar{y}$ occur with other literals say $\tilde{r}$ and $\tilde{s}$, respectively. Now when we branch as $F[y][]$ two clauses are satisfied and $\tilde{r}$ becomes a $(2^-,1^-)$ literal. Similarly, when we branch as $F[][y]$, $\tilde{s}$ becomes a $(2^-,1^-)$ literal. Thus, the invariant is always maintained at the end of the subroutine **Create-Low-Literal**. The branch in Step 2 is described by the recurrence relation $C(m) \leq 2C(m-2)$. The proof follows. $\square$

Our algorithm is divided into two phases. In the first phase we apply **Case 3.1** and **Case 3.2** of the algorithm in Section 3 to eliminate all literals (i,j) where $i+j \geq 6$ or $i+j=5$ with $i,\ j>1$, and all $(3,1)$ and $(4,1)$ literals whose negations do not occur as unit clauses. **Case 3.1** and **Case 3.2** guarantee a recurrence relation $C(k) \leq C(k-1)+C(k-5)$ for the parameterized case, and hence a recurrence relation $C(m) \leq C(m-1)+C(m-5)$ for the non-parameterized case. Now if F contains a $(4,1)$ literal $\tilde{x}$, then $\bar{x}$ has to occur as a unit clause, and we branch as $F[x][]$ and $F[][x]$. In the first branch four clauses are satisfied and one is eliminated (the unit clause containing $\bar{x}$), and in the second branch one clause is satisfied. Hence, we branch with the recurrence relation $C(m) \leq C(m-1)+C(m-5)$. After this phase of the algorithm, we know that F does not contain any (i,j) literal where $i+j \geq 5$, and all $(3,1)$ literals have their negations occurring as unit clauses.

The next phase of the algorithm works in stages. The algorithm at each stage picks the first branching rule that applies and uses it. Also, at the beginning of each stage we will maintain the following invariant: The formula F has a $(2^-,1^-)$ literal or all literals are $(3^-,1^-)$ literals (i.e., no $(2,2)$ literal exists). The way we keep this invariant is by guaranteeing that after each branching case, if F does not consist solely of $(3^-,1^-)$ literals, then either a $(2^-,1^-)$ literal remains in the formula, or we can create one by applying the **Create-Low-Literal** subroutine. We can assume, without loss of generality, that a $(2^-,1^-)$ literal exists in the first stage of the algorithm. Otherwise, we can introduce a new unit clause containing a new variable x and increase the parameter m by 1. It can be easily seen that the running time of the algorithm will not increase by more than a multiplicative constant. Before calling **Create-Low-Literal**, we apply **TransRule 1** through **TransRule 3** as long as they are applicable, then we check that a $(2,2)$ literal exists in the remaining formula. Otherwise, **Create-Low-Literal** is not called. The algorithm is described in Figure 3. We assume that after each application of a transformation rule or a branching case, the formula F and the truth assignment τ are updated accordingly. In the **else** statement in Step 5, the algorithm picks the first branching case of branching cases **Case 4.1-4.12**, given below, that applies. We can assume, without loss of generality, that before the application of any branching case the formula F does not contain any closed subformula H of small size (bounded by 10, for instance). Otherwise, an assignment satisfying the maximum number of clauses in H can be computed in constant time, and the reduction in the parameter can be used to create a $(2^-,1^-)$ literal using **Create-Low-Literal**. This case needs to be considered since at the beginning of the algorithm we can create a $(2^-,1^-)$ literal (in case it does not exist) at the expense of increasing the running time by a multiplicative constant. However, we cannot afford doing that all the time.

So if at a certain point all the $(2^-, 1^-)$ literals are contained in a closed subformula H, we want to make sure that if we branch and H becomes empty, a $(2^-, 1^-)$ literal still remains in the formula. Thus, if the size of H becomes small (bounded by a constant), we can solve H easily without any branching, and use this reduction in the parameter to create a $(2^-, 1^-)$ literal in the remaining formula. Again, we emphasize that the requirement that the cases below should be considered in the given order plays a crucial role in the correctness of the algorithm.

```
Algorithm. Max-Sat
Input: a boolean formula $F$ containing $m$ clauses
Output: a truth assignment $\tau$ satisfying the maximum number of clauses in $F$
1. $\tau = \emptyset$;
2. while ($F \neq \emptyset$) and (any of Case 3.1-3.2 is applicable) then
       apply it to $F$;
3. while ($F \neq \emptyset$) and ($F$ contains a $(4, 1)$ literal $\tilde{x}$) do
       branch as $F[x][]$ and $F[][x]$;
4. if ($F \neq \emptyset$) and ($F$ does not contain a $(2^-, 1^-)$ literal) then
       $F = F \wedge (w)$ where $w$ is a new variable;
5. while ($F \neq \emptyset$) do
       if $F$ contains only $(3^-, 1^-)$ literals then
           if any of TransRule 1-3 is applicable then apply it to $F$
           else apply Case 4.0
       else if $F$ contains a closed subformula $H$ of size $\leq 10$ then
           solve $H$ and call Create-Low-Literal;
       else branch according to the first branch in Case 4.1-4.12 that applies;
6. return $\tau$;
```

Figure 3: An algorithm for Max-Sat

Case 4.0. F contains only $(3^-, 1^-)$ literals. In this case either a $(2^-, 1^-)$ literal exists, or the formula consists of only $(3, 1)$ literals. In the former case one of **TransRule 1-3** or one of the cases **Case 3.4** through **Case 3.6** must apply, hence branching with a worst case recurrence $C(m) \leq C(m-1) + C(m-5)$. In the latter case one of cases **Case 3.11** through **Case 3.13** applies. Now observing that the recurrence relation in **Subcase 3.13.2** becomes $C(m) \leq C(m-1) + C(m-7) + C(m-10)$ for the non-parameterized case (since when $x = 1$ the unit clause containing $\bar{x}$ will be eliminated), we conclude that in these cases we can branch with a worst case recurrence relation $C(m) \leq C(m-1) + C(m-5)$.

Case 4.1. Any of **TransRule 1** through **TransRule 3** is applicable. Apply the transformation rule and then **Create-Low-Literal** if necessary. $C(m) \leq 2C(m-3)$

Let $\tilde{y}$ be a $(2, 2)$ literal and $\tilde{x}$ be a $(2, 1)$ literal. Note that a $(2, 1)$ literal must exist at this point since the invariant guarantees the existence of a $(2^-, 1^-)$ literal, and **Case 4.1** takes care of all the other possibilities.

Case 4.2. y or $\bar{y}$ occurs as a unit clause. Assume, without loss of generality, that y occurs as a unit clause. Branch as $F[y][]$ and $F[][y]$. In both branches at least one occurrence of the literal $\tilde{x}$ remains, and hence, the invariant is satisfied. $C(m) \leq C(m-2) + C(m-3)$

Now assume that $\tilde{y}$ occurs with $\tilde{x}$. Without loss of generality, suppose that y occurs with $\tilde{x}$.

Case 4.3. y occurs only with $\bar{x}$. Branch as $F[y][]$ and apply **TransRule 1** to x, and then **Create-Low-Literal**, and $F[][y]$. Note that in the second branch a $(2^-, 1^-)$ literal remains (namely $\tilde{x}$). $C(m) \leq 2C(m-6) + C(m-2)$

Case 4.4. There is a literal $\tilde{r}$ that occurs with y such that $\tilde{r}$ occurs also outside y and $\tilde{x}$. Branch as $F[y][]$ and apply **TransRule 1** or **TransRule 3** to $\tilde{x}$, and $F[][y]$. In the first branch $\tilde{r}$ becomes a $(2^-, 1^-)$ literal, and in the second $\tilde{x}$. $C(m) \leq C(m-3) + C(m-2)$

Case 4.5. The other occurrence of y is outside $\tilde{x}$. Let $\tilde{r}$ be a literal that occurs with y outside $\tilde{x}$. Since **Case 4.4** was excluded, all other occurrences of $\tilde{r}$ have to be with $\tilde{x}$. Branch as $F[y][]$ and apply **TransRule 1** or **TransRule 3** to $\tilde{x}$ and $\tilde{r}$, and apply **Create-Low-Literal**, and $F[][y]$. In the second branch $\tilde{x}$ remains. $C(m) \leq 2C(m-6) + C(m-2)$

Now both occurrences of y are with $\tilde{x}$. Also, both occurrences of $\bar{y}$ are outside $\tilde{x}$, otherwise, by symmetry, we can apply one of the above cases with y exchanged with $\bar{y}$.

Case 4.6. There is a literal $\tilde{r}$ that occurs simultaneously with $\tilde{x}$ and outside $\tilde{x}$. Branch as $F[y][]$ and apply **TransRule 1** to $\tilde{x}$, and $F[][y]$. In the first $\tilde{r}$ becomes a $(2^-, 1^-)$ literal, and in the second $\tilde{x}$ remains. $C(m) \leq C(m-3) + C(m-2)$

Case 4.7. There is a literal $\tilde{r}$ distinct from y that occurs with $\tilde{x}$. Since the previous case was excluded, $\tilde{r}$ can only occur with $\tilde{x}$, and hence it must be a $(2,1)$ literal. In this case a safe partial assignment that satisfies the three clauses containing $\tilde{x}$ and $\tilde{r}$ can be easily computed, and we can apply **Create-Low-Literal** to create a $(2^-, 1^-)$ literal in the resulting formula. $C(m) \leq 2C(m-5)$

Case 4.8. y is the only variable that occurs with $\tilde{x}$ and $\bar{x}$ occurs as a unit clause. By **BranchRule 2**, branch $F[x][y]$ and $F[][y,x]$ reduce to $F[x][y]$. Thus, we branch as $F[y][]$ and apply **TransRule 1** to $\tilde{x}$, then apply **Create-Low-Literal**, and $F[x][y]$ and apply **Create-Low-Literal**. $C(m) \leq 2C(m-5) + 2C(m-7)$

Now we can assume that no $(2,2)$ literal occurs with a $(2,1)$ literal.

Case 4.9. The $(2,1)$ literals form a closed subformula. Let H be the closed subformula consisting of the $(2,1)$ literals. Since closed subformulas of size ≤ 10 have been taken care of, we can assume that $|H| > 10$. Now we can apply one of the cases **Case 3.4** through **Case 3.6**. In this case we branch with recurrence relation $C(m) \leq C(m-1) + C(m-5)$, and the remaining formula of H will contain a $(2^-, 1^-)$ literal. The last statement is true since the number of clauses in H is chosen to be large enough so that none of the branches in **Case 3.4** through **Case 3.6** can eliminate all the clauses in H, thus leaving at least one $(2^-, 1^-)$ literal.

Now we must have a $(3,1)$ literal $\tilde{y}$ occurring with a $(2,1)$ literal $\tilde{x}$ or else the $(2,1)$ literals would form a closed subformula. Note that all $(3,1)$ literals have their negative occurrences as unit clauses.

Case 4.10. $\tilde{x}$ is dominated by $\tilde{y}$. Now branch $F[][y]$ contains $F[x][y]$. By **BranchRule 2**, $F[x][y]$ and $F[y][x]$ reduce to $F[x][y]$. Also $F[x][y]$ and $F[x,y][]$ reduce to $F[x][y]$. Thus, in this case we branch as $F[][y]$ and $\tilde{x}$ remains in the resulting formula. We get recurrence relation $C(m) = C(m-1)$.

Case 4.11. y and $\tilde{x}$ occur twice together. Note that if we do not have a literal $\tilde{r}$ that occurs with $\tilde{x}$ or $\tilde{y}$ and that has at least one occurrence outside the clauses containing $\tilde{x}$ or $\tilde{y}$, then the clauses containing $\tilde{x}$ or $\tilde{y}$ would form a closed subformula of F containing at most five clauses. So we can assume that such a literal $\tilde{r}$ exists. Branch as $F[y][]$ and apply **TransRule 1** to $\tilde{x}$, and $F[][y]$. In the first branch $\tilde{r}$ becomes a $(2^-, 1^-)$, and in the second $\tilde{x}$ remains. $C(m) \leq C(m-5)+C(m-1)$

Case 4.12. y occurs exactly once with $\tilde{x}$. We distinguish two subcases.

Subcase 4.12.1. y occurs with x. By **TransRule 2**, none of the occurrences of y can be in a unit clause. Now let $\tilde{r}$ be a literal different from $\tilde{x}$ that occurs with y. Since the previous two cases were eliminated, $\tilde{r}$ must occur outside y (if $\tilde{r}$ is a $(3, 1)$ literal, this condition is automatically satisfied). Branch as $F[y][]$ and apply **TransRule 3** to $\tilde{x}$, and $F[][y]$. In the first branch $\tilde{r}$ becomes a $(2^-, 1^-)$ literal, and in the second $\tilde{x}$ remains. $C(m) \leq C(m-5) + C(m-1)$

Subcase 4.12.2. y occurs with $\bar{x}$. Branch as $F[y][]$ and **TransRule 1** to $\tilde{x}$ then apply **Create-Low-Literal**, and $F[][y]$. In the second branch $\tilde{x}$ remains. $C(m) \leq 2C(m-8)+C(m-1)$

Theorem 4.2 *Given a boolean formula F of m clauses then in time $O(1.3247^m|F|)$ we can compute a truth assignment to F satisfying the largest number of clauses.*

Proof. It is easy to see that the above cases are comprehensive in the sense that they cover all possible situations. The size of the branching tree is the largest when we branch with the recurrence relation $C(m) \leq C(m-1) + C(m-5)$. This gives a tree size of $O(\alpha^m)$ where $\alpha \approx 1.3247$ is the unique positive root of the polynomial $x^5 - x^4 - 1$. Along each root-leaf path in the branching tree we spend $O(|F|)$ time. Hence, the running time of the algorithm is $O(1.3247^m|F|)$. $\square$

The above algorithm is an improvement on Bansal and Raman's $O(1.341294^m|F|)$ algorithm for the Max-Sat problem [7].

5 Concluding remarks

In this paper we presented two exact algorithms for the Max-Sat problem. Both algorithms induce improvements on the previously best algorithms by Bansal and Raman for the problem. Basically the technique used in this paper is a variation of the search tree technique which has been widely employed in designing exact algorithms for NP-hard problems [6, 7, 11, 14, 15, 16, 22, 25, 33, 35]. Although case-by-case analysis seems unavoidable when using the search tree method to solve such problems, reducing the number of cases by introducing new techniques that either enable the classification of multiple cases into a general case, or exploit the structure of the combinatorial problem by looking more carefully at its nature, is always desirable. Such techniques like the "vertex folding" and the "iterative branching" introduced in [15], the "struction" in [11], and the **Create-Low-Literal** subroutine introduced in this paper, have reduced significantly the number of cases in the problems considered.

The general open question that is posed is to what extent we can keep improving these upper bounds? Using the results in [29], one can easily show that if the Max-Sat problem can be solved in sub-linear exponential time (i.e., $O(2^{o(m)}p(n))$, where m is the number of clauses, n the number of variables, and p a polynomial), then so can a family of problems including k-SAT, Independent Set, and Vertex Cover. This means that it is unlikely that the Max-Sat problem admits a sub-linear exponential time algorithm, and hence, it seems very likely that there exists a constant

$c > 0$, such that MAX-SAT has no algorithm of running time $O((1+c)^m p(n))$. How close are we to this constant c remains an open question.

Finally, we note that even though the algorithms for such NP-hard problems tend to be based on case-by-case analysis, these cases are easy to distinguish, and hence can be implemented easily. The remaining question would be how well these algorithms can perform in practice in comparison with their theoretical upper bounds and other existing heuristics for the problems.

References

[1] J. ALBER, H. L. BODLAENDER, H. FERNAU, T. KLOKS, AND R. NIEDERMEIER, Fixed parameter algorithms for dominating set and related problems on planar graphs, *Algorithmica* **33**, (2002), pp. 461-493.

[2] J. ALBER, H. FAN, M. R. FELLOWS, H. FERNAU, R. NIEDERMEIER, F. ROSAMOND, AND U. STEGE, Refined search tree technique for Dominating Set on planar graphs, in *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, *Lecture Notes in Computer Science* **2136**, (2001), pp 111-122.

[3] T. ASANO, K. HORI, T. ONO, AND T. HIRATA, A theoretical framework of hybrid approaches to MAX-SAT, in *Proceedings of the 8th International Symposium on Algorithms and Computation (ISAAC 1997)*, *Lecture Notes in Computer Science* **1350**, (1997), pp. 153-162.

[4] T. ASANO, AND D. P. WILLIAMSON, Improved Approximation Algorithms for MAX-SAT, in *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (2000), pp. 96-105.

[5] P. ASIRELLI, M. DE SANTIS, AND A. MARTELLI, Integrity constraints in logic databases, *Journal of Logic Programming* **3**, (1985), pp. 221-232.

[6] R. BALASUBRAMANIAN, M. R. FELLOWS, AND V. RAMAN, An improved fixed parameter algorithm for vertex cover, *Information Processing Letters* **65**, (1998), pp. 163-168.

[7] N. BANSAL AND V. RAMAN, Upper bounds for MAX-SAT further improved, in *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC)*, *Lecture Notes in Computer Science* **1741**, (1999), pp. 247-258.

[8] R. BATTITI AND M. PROTASI, Reactive research, a history base heuristic for MAX-SAT, *J. Exper. Algorithmics* **2**, No. 2, (1997).

[9] R. BATTITI AND M. PROTASI, Approximate algorithms and heuristics for MAX-SAT, in *Handbook of Combinatorial Optimization* **1**, D. Z. Du and P. M. Pardalos Eds., (1998), pp. 77-148.

[10] B. BORCHERS AND J. FURMAN, A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems, *J. Combinatorial Optimization* **2**, (1999), pp. 465-474.

[11] R. Beigel, Finding maximum independent sets in sparse and general graphs, in *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (1999), pp. 856-857.

[12] L. Cai and J. Chen, On fixed-parameter tractability and approximation of NP-hard optimization problems, *Journal of Computer and System Sciences* **54**, (1997), pp. 465-474.

[13] J. Chen, D. K. Friesen, W. Jia, and I. A. Kanj, Using nondeterminism to design efficient deterministic algorithms, in *proceedings of the 21st annual conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Lecture Notes in Computer Science* **2245**, (2001), pp. 120-131.

[14] J. Chen and I. A. Kanj, On constrained minimum vertex covers of bipartite graphs: Improved Algorithms, in *Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Lecture Notes in Computer Science (LNCS)* **2204**, (2001), pp. 55-65.

[15] J. Chen, I. A. Kanj, and W. Jia, Vertex cover, further observations and further improvements, *Journal of Algorithms* **41**, (2001), pp. 280-301.

[16] J. Chen, L. Liu, and W. Jia, Improvement on Vertex Cover for low-degree graphs, *Networks* **35**, (2000), pp. 253-259.

[17] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning, A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search, *Theoretical Computer Science* **289-1**, (2002), pp. 69-83.

[18] M. Davis and H. Putnam, A computing procedure for quantification theory, *Journal of the ACM* **7**, (1960), pp. 201-215.

[19] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, New York: Springer, (1999).

[20] R. G. Downey, M. R. Fellows, and U. Stege, Parameterized complexity: A framework for systematically confronting computational intractability, in *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, F. Roberts, J. Kratochvíl, and J. Nešetřil, eds., *AMS-DIMACS Proceedings Series* **49**, AMS, (1999), pp. 49-99.

[21] D. Eppstein, Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction, in *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (2001), pp. 329-337.

[22] H. Fernau and R. Niedermeier, An efficient exact algorithm for constraint bipartite vertex cover, *Journal of Algorithms* **38**, (2001), pp. 374-410.

[23] H. Gallaire, J. Minker, and J. M. Nicolas, Logic and databases: A deductive approach, *Computing Surveys* **16**, No. 2, (1984), pp. 153-185.

[24] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.

[25] J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith, New worst-case upper bounds for Max-2-Sat with application to Max-Cut, to appear in *Discrete Applied Mathematics*. A preliminary version appeared as technical report **TR00-037** in the *Electronic Colloquium on Computational Complexity (ECCC)* **7 (37)**, (2000).

[26] J. Gramm, and R. Niedermeier, Faster exact solutions for Max-2-Sat, in *Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC)*, *Lecture Notes in Computer Science* **1767**, (2000), pp. 174-186.

[27] P. Hansen and B. Jaumard, Algorithms for the maximum satisfiability problem, *Computing* **44**, (1990) pp. 279-303.

[28] F. Hayes, D. A. Waterman, and D. B. Lenat, *Building Expert Systems*, Reading Massachusetts: Addison Wesley, (1983).

[29] R. Impagliazzo, R. Paturi, and F. Zane, Which Problems Have Strongly Exponential Complexity?, *Journal of Computer and System Sciences* **63-4**, (2001), pp. 512-530.

[30] O. Kullmann and H. Luckhardt, Deciding propositional tautologies: Algorithms and their complexity, submitted for publication, available at `http://cs-svr1.swan.ac.uk/ csoliver/papers.html`.

[31] M. Mahajan and V. Raman, Parameterizing above guaranteed values: Max-Sat and Max-Cut, *Journal of Algorithms* **31**, (1999), pp. 335-354.

[32] T. A. Nguyen, W. A. Perkins, T. J. Laffey, and D. Pecora, Checking an expert systems knowledge base for consistency and completeness, *IJCAI'85*, Arvind Joshi Ed., Los Altos, CA, (1985), pp. 375-378.

[33] R. Niedermeier and P. Rossmanith, New upper bounds for maximum satisfiability, *Journal of Algorithms* **36**, (2000), pp. 63-88.

[34] R. Niedermeier and P. Rossmanith, A general method to speed up fixed-parameter-tractable algorithms, *Information Processing Letters* **73**, (2000), pp. 125-129.

[35] J. M. Robson, Algorithms for maximum independent set, *Journal of Algorithms* **6**, (1987), pp. 425-440.

[36] R. J. Wallace, Enhancing maximum satisfiability algorithms with pure literal strategies, in *Proceedings of the 11th Canadian Conference on Artificial Intelligence (AI)*, *Lecture Notes in Artificial Intelligence* **1081**, (1996), pp. 388-401.

[37] R. J. Wallace and E. C. Feuder, Comparative studies of constraints satisfaction and Davis-Putnam algorithms for maximum satisfiability problems, *"Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenges"*, D. S. Johnson and M. A. Tricks Eds.,

DIMACS Series on Discrete Mathematics and Theoretical Computer Science **26**, American Mathematical Society, Providence, RI, (1996).