BRICS RS-04-19 Madsen & Rossmanith: Maximum Exact Satisfiability: NP-completeness Proofs and Exact Algorithms

**Basic Research in Computer Science**

# Maximum Exact Satisfiability: NP-completeness Proofs and Exact Algorithms

**Bolette Ammitzbøll Madsen**
**Peter Rossmanith**

**BRICS Report Series** **RS-04-19**

**ISSN 0909-0878** **October 2004**

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

# Maximum Exact Satisfiability: NP-completeness Proofs and Exact Algorithms

Bolette Ammitzbøll Madsen
BRICS*, Department of Computer Science
University of Aarhus, Denmark
bolette@brics.dk

Peter Rossmanith
Department of Computer Science
RWTH, Aachen, Germany
rossmani@informatik.rwth-aachen.de

October 2004

**Abstract**

Inspired by the Maximum Satisfiability and Exact Satisfiability problems we present two Maximum Exact Satisfiability problems. The first problem called Maximum Exact Satisfiability is: given a formula in conjunctive normal form and an integer $k$, is there an assignment to all variables in the formula such that at least $k$ clauses have exactly one true literal. The second problem called Restricted Maximum Exact Satisfiability has the further restriction that no clause is allowed to have more than one true literal. Both problems are proved NP-complete restricted to the versions where each clause contains at most two literals. In fact Maximum Exact Satisfiability is a generalisation of the well-known NP-complete problem MaxCut. We present an exact algorithm for Maximum Exact Satisfiability where each clause contains at most two literals with time complexity $O(poly(L) \cdot 2^{m/4})$, where $m$ is the number of clauses and $L$ is the length of the formula. For the second version we give an algorithm with time complexity $O(poly(L) \cdot 1.324718^n)$, where $n$ is the number of variables. We note that when restricted to the versions where each clause contains exactly two literals and there are no negations both problems are fixed parameter tractable. It is an open question if this is also the case for the general problems.

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

# 1 Introduction

Much work on NP-complete problems has centered around SAT as the original NP-complete problem. Many satisfiability type problems have been introduced, and many exact, approximate and parameterised algorithms for these problems have been given. We introduce two new NP-complete satisfiability problems and give exact algorithms for both.

Traditionally algorithms for satisfiability problems have been given in the following three parameters: $n$ – the number of variables, $m$ – the number of clauses and $L$ – the number of literals or the length of the formula. For further introduction to these terms, see section 2.

## 1.1 Background

**MaxSat.** The Maximum Satisfiability problem (MaxSat), which is an important generalisation of SAT, is: Given a boolean formula $F$ in conjunctive normal form (CNF) with $m$ clauses and a positive integer $k \leq m$, is there a truth assignment to the variables in $F$ that simultaneously satisfies at least $k$ of the $m$ clauses? Thus a fourth parameter $k$ is introduced in maximisation/minimisation problems.

MaxSat is NP-complete even when restricted to Max2Sat, where each clause contains at most 2 literals [8]. The algorithms with the currently best known running times for MaxSat have time complexity $O(L \cdot 1.3247^m)$ [4], $O(L+k^2 \cdot 1.3695^k)$ [4] and $O(L \cdot 1.105729^L)$ [1]. For Max2Sat, the best known time complexities are $O(poly(L) \cdot 2^{(m/5)})$ and $O(poly(L) \cdot 2^{(L/10)})$, where $poly(L)$ is some polynomial in $L$ [9]. When looking at the variables, it is only recently that the trivial bound of $2^n$ was improved. The new algorithm for Max2Sat by Williams from 2004 has time complexity $O(m^3 2^{\omega n/3})$, where $\omega < 2.379$ is the matrix product exponent over a ring [18]. The improved time bound however comes at the expense of using exponential space, whereas all other algorithms mentioned in this paper only use polynomial space.

**XSAT.** The Exact Satisfiability problem XSAT (also called 1-in-$k$-SAT) is the following: Given a boolean formula in conjunctive normal form, is there an assignment to the variables satisfying exactly one literal in each clause? We call such an assignment exactly satisfying.

The Exact Satisfiability problem is NP-complete even when restricted to the case where each clause contains at most 3 literals and there are no negations [17]. This problem has also received renewed attention in recent years [11, 15, 5, 2].

## 1.2 New problems

**MaxExactSat.** The MaxExactSat problem is: given a boolean formula in CNF and an integer $k$, is there a truth assignment to the variables exactly satisfying at least $k$ clauses simultaneously? This problem is NP-hard even restricted to the special case MaxExact2Sat (each clause contains at most two literals) with no negations and no unit clauses, as this special case is equivalent to the well-known graph-problem MaxCut, which is known to be NP-complete [10]. We present an algorithm for MaxExact2Sat (with negations and unit clauses) which has a running time of $O(poly(L) \cdot 2^{m/4})$. This exponent matches that of the best known algorithm for MaxCut, which is $O(poly(m) \cdot 2^{m/4})$, where $m$ is the number of edges [7].

**ResMaxExactSat.** The second problem is Restricted MaxExactSat (ResMaxExactSat). The ResMaxExactSat problem is MaxExactSat with the restriction that no clauses may be 'over-satisfied', i.e. two literals in the same clause are not both allowed to be *true*. We show that the ResMaxExactSat problem is also NP-complete. Interestingly we can give a good algorithm for ResMaxExactSat in the number of variables, which has only recently been achieved for other MaxSat problems [18]. The time complexity of the algorithm is $O(poly(L) \cdot 1.324718^n)$.

## 1.3 Techniques and concepts

**Branching algorithms and bottlenecks.** Both the presented exact algorithms are branching algorithms (also called splitting algorithms). The main idea of a branching algorithm for a satisfiability problem is to 'branch' on setting a variable to either *true* or *false*, which gives us two smaller formulas to solve. The algorithm then reduces the parameter before branching again. This gives rise to a branching tree.

However in one case, we are not able to reduce the parameter by very much. Such a worst case situation is often referred to as a 'bottleneck' instance. We show that this bottleneck situation can only arise once in any path from the root to a leaf in the branching tree, a technique also used by e.g. Robson [16], Gramm et al. [9] and Fedin et al. [7]. Thus it only has the effect of a constant on the running time of the algorithm, and we can omit the corresponding recurrence inequality from the analysis of the running time.

**Parameterised algorithms.** The classic MaxSat problem is fixed parameter tractable, which means that given a formula $F$, the question of whether $k$ clauses can be satisfied can be answered in time $poly(L) \cdot f(k)$, where $f(k)$ depends only on $k$.

It is not immediate that MaxExactSat or ResMaxExactSat is fixed parameter tractable. However MaxExact2Sat with no unit clauses and no negations is equivalent to MaxCut, which is known to be fixed parameter tractable [14, 3]. We give an algorithm that shows that ResMaxExact2Sat with no unit clauses and no negations is also fixed parameter tractable.

### 1.4 Outline

In section 2 we define the two problems and introduce the notation we will use. In section 3 we prove that both of these problems are NP-complete, and that a special variant of MaxExact2Sat, where each variable occur in at most two clauses of size two, is in P. In section 4 we present exact algorithms for both problems and look at parameterisation.

## 2 Preliminaries

In this paper a formula is a conjunction of $m$ clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ over $n$ variables. A clause $(l_1, l_2, \ldots, l_k)$ is a set of literals and possibly constants (*true* and *false*). A literal is either a variable $x$ or the negation of a variable $\bar{x}$. A *unique* variable is a variable that only occurs once in the formula. The length of a formula, $L$, is the number of literals (and constants) in the formula. A $k$-clause is a clause with $k$ literals and constants, and a unit clause is a clause with only one literal (or constant). For simplicity we will assume that all variables in a formula occur at least as many times unnegated as negated (otherwise replace literal $\bar{x}$ with a new variable $x_{not}$ and $x$ with $\bar{x}_{not}$). An assignment to a variable $x$, sets $x$ to either *true* or *false*; a full assignment is an assignment to all variables in a formula.

A clause is *exactly satisfied* if exactly one of its literals is *true*. In this paper 'satisfied' always means 'exactly satisfied'. If more than one literal in a clause are *true*, we say that the clause is *oversatisfied*.

We define clauses as sets, so it is not possible for the same literal to occur twice in the same clause. We also assume that a literal does not occur in a clause together with its negation. For 2-clauses however this is not a problem, as such clauses can easily be reduced. A 2-clause with two occurrences of the same literal can not be exactly satisfied, and a 2-clause with a literal and its negation will always be satisfied.

**Definition 1.** The MaxExactSat (Maximum Exact Satisfiability) problem is: given a formula in CNF and an integer $k$, is there a full assignment exactly satisfying at least $k$ clauses?

The MaxExact$k$Sat problem is the MaxExactSat problem, where all clauses are restricted to at most $k$ literals.

**Definition 2.** The ResMaxExactSat (Restricted Maximum Exact Satisfiability) problem is MaxExactSat with the restriction that no clause may be oversatisfied.

We note that in a ResMaxExactSat instance, we can eliminate all occurrences of the same variable twice in a clause. If the literal $x$ occurs twice in a clause, it must be set to *false*, as oversatisfying is not allowed, and if $x$ occurs together with $\bar{x}$ in a clause, one of these literals will satisfy the clause, so all other literals in the clause must be set to *false*, and the clause can be removed.

We shall also look at a few graph problems on undirected graphs. Given a graph $G = (V, E)$, we denote the number of vertices $n = |V|$ and the number of edges $m = |E|$.

The MaxCut problem is: given a graph $G = (V, E)$ and an integer $k$, is there a cut in $G$ of size at least $k$, i.e. a partition of $V$ into disjoint sets $V_1$ and $V_2$ such that the number of edges, that have one endpoint in $V_1$ and one endpoint in $V_2$ is at least $k$. This problem is NP-complete [10].

The Maximum Independent Set problem is: given a graph $G = (V, E)$ and an integer $k$, is there an independent set of size at least $k$ in $G$, i.e. a subset $V' \subseteq V$ of size at least $k$ such that no two vertices in $V'$ are joined by an edge in $E$. This problem is also NP-complete [10].

## 3 NP-completeness

We show that MaxExactSat and ResMaxExactSat are NP-complete even restricted to clauses with at most two literals, and we prove that the version of MaxExact2Sat in which all variables are restricted to occur in at most two 2-clauses, can be solved in polynomial time.

**Theorem 1.** *The* MaxExactSat *problem is NP-complete even when restricted to the special-case* MaxExact2Sat *with no negations and no unit clauses.*

*Proof.* The MaxExact2Sat problem with no negations and no unit clauses is NP-complete, as it is equivalent to MaxCut, which is known to be NP-complete.

Given a MaxExact2Sat instance $F$ with no negations and no unit clauses, construct a graph $G$ as follows: translate each variable to a vertex and each clause to an edge. We prove that there is a cut of size $k$ in $G$ if and only if there is a full assignment satisfying $k$ clauses in $F$.

Given an assignment to the variables in $F$, which satisfies $k$ clauses, let $V_1$ be the vertices corresponding to *true* variables in the assignment and $V_2$ the vertices corresponding to the *false* variables. $(V_1, V_2)$ is a cut of size $k$ in

$G$: the clauses, which are exactly satisfied, are the ones in which one of the variables is *true* and one is *false*. These clauses correspond to the edges in the cut.

Now given a cut $(V_1, V_2)$ of size $k$ in $G$, set all variables corresponding to vertices in $V_1$ to *true* and all variables corresponding to vertices in $V_2$ to *false*. Then every clause corresponding to an edge in the cut will be exactly satisfied.

Given a MaxCut instance $G$, construct a formula $F$ by translating each vertex to a variable and each edge to a clause. The proof that there is a cut of size $k$ in $G$ if and only if there is a full assignment satisfying $k$ clauses in $F$ is the same as above. □

**Theorem 2.** *If every variable in a* MaxExact2Sat *formula occurs in at most two 2-clauses, the problem can be solved in polynomial time.*

*Proof.* First we note that we can simplify a formula where all variables occur in at most two 2-clauses, such that all variables occur in exactly two 2-clauses and possibly in one unit clause. We use the following three simplifications, as long as one is applicable:

- If there are two clauses $(x)$ and $(\bar{x})$, no matter what $x$ is set to we will satisfy exactly one of these, so we can simply delete both and add one to the number of satisfied clauses.

- If there is a variable $x$ occurring in at least as many unit clauses as 2-clauses, we will satisfy at least as many clauses setting $x$ to satisfy the unit clauses as we would if we set $x$ to the opposite, so we set $x$ to satisfy the unit clauses, delete these and add the number of them to the number of satisfied clauses. Any 2-clause $(\mathit{true}, y)$ is replaced by the unit clause $(\bar{y})$ and any clause $(\mathit{false}, y)$ by $(y)$.

- If there is a unique variable $x$, it must be in a 2-clause or it would have been set by the above simplification. The clause in which $x$ occurs can always be satisfied by setting $x$ to the negation of the other variable in the clause, so we can delete this clause and add one to the number of satisfied clauses.

We note that checking if one of these simplifications apply and applying it can be done in polynomial time, and all three simplifications remove at least one clause, so at most $m$ simplifications are applied; thus the formula is simplified in polynomial time.

For simplicity we assume that the formula is connected; otherwise we solve the individual formulas separately. Since all variables now occur in exactly two 2-clauses, the 2-clauses constitute a cycle on the variables. We have four cases: we have either an odd or an even cycle and we have either an odd or an even number of negations.

If we have either an odd cycle and an odd number of negations or we have an even cycle and an even number of negations we can satisfy all 2-clauses. Note that since there are two 2-clauses and at most one unit clause with each variable, it will always be best to satisfy the two 2-clauses. This means that there are only 2 feasible solutions (pick a variable $x$ from one of the 2-clauses):

- set $x = true$ and set all other variables such that all the 2-clauses are satisfied.
- set $x = false$ and set all other variables such that all the 2-clauses are satisfied.

The result is then simply the one of the two possible solutions with the most satisfied clauses.

If we have an odd cycle and an even number of negations or an even cycle and an odd number of negations we can satisfy all but one of the 2-clauses. In this case we have $2t$ possibilities for a cycle of $t$ 2-clauses. We have $t$ possibilities of choosing which 2-clause is not satisfied and we have two possibilities of setting the variables as described in the case with odd/odd and even/even. So we solve this by picking a 2-clause, picking a truth value for the chosen variable $x$ and then setting the rest of the variables such that all but the chosen clause is satisfied (this is repeated for the $2t$ possibilities and the best solution is chosen). Note again it does not pay to satisfy a unit clause instead of a 2-clause as we have at least one 2-clause that is satisfied with each variable.

□

To see that the ResMaxExactSat problem is NP-hard, we first introduce the 'Independent Set Max Cover' problem.

**Definition 3.** The Independent Set Max Cover problem is: given an undirected graph and an integer $k$, is there an independent set with at least $k$ edges to the rest of the graph.

Independent Set Max Cover can also be described as MaxCut with the restriction that one side of the cut must be an independent set.

**Theorem 3.** *The* Independent Set Max Cover *problem is NP-complete.*

*Proof.* The problem is easily seen to be in NP.

The problem is NP-hard as Maximum Independent Set reduces to it. Given an instance of the Maximum Independent Set problem, i.e. a graph $G = (V, E)$, construct the instance $G'$ of the Independent Set Max Cover problem as follows: start with $G$. For each $v \in V$ construct a clique of size $n$ and attach this clique to $v$ by $n - d$ edges, where $d$ is the degree of $v$ in $G$. For simplicity assume that we have no vertices with degree zero. See example in figure 1.

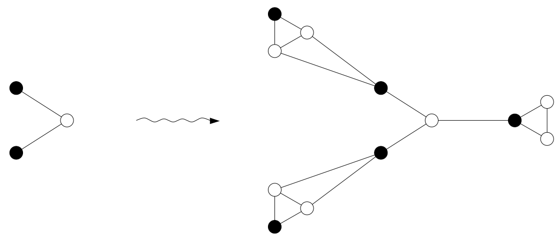

Figure 1: Reduction from an instance $G$ of MIS to an instance $G'$ of ISMC. The vertices constituting the maximum independent set in $G$ respectively the independent set max cover in $G'$ are marked black.

We now prove that the original graph $G$ has an independent set of size $k$ if and only if the graph $G'$ has an independent set which covers $(2n-1)k + n(n-k) = n^2 + (n-1)k$ edges.

In $G'$ each original vertex of $G$ has degree $n$, and each of the new vertices connected to an original vertex also has degree $n$, whereas the new vertices not connected to an original vertex have degree $n-1$. If we choose an 'original' vertex for our independent set, we can also choose an extra vertex from its clique (assuming the graph has no isolated vertices), which adds $2n-1$ covered edges; choosing a new vertex in the clique adds only $n$ covered edges.

If the original graph has an independent set of size $k$, choose this set plus an extra vertex from each vertex-clique in the new graph. This set covers $(2n-1)k + n(n-k)$ edges. If the new graph has an independent set, which covers $(2n-1)k+n(n-k)$ edges, it must be because $n-k$ of the vertices each cover $n$ edges, and the two vertices chosen in each of the remaining $k$ vertex-cliques cover $2n-1$ edges. The $k$ original vertices from the cliques where two vertices are chosen form an independent set in the original graph. □

We note that the above reduction is a polynomial-time transformation, but it is not a parameterised reduction. This is significant as Maximum Independent Set is W[1]-complete [6], whereas we shall later show that Independent Set Max Cover is fixed parameter tractable.

**Theorem 4.** *The* ResMaxExactSat *problem is NP-complete even when restricted to the* ResMaxExact2Sat *problem with no unit clauses and no negations.*

*Proof.* The ResMaxExact2Sat problem with no unit clauses and no negations is equivalent to Independent Set Max Cover, which we have just shown NP-complete. Given an instance $F$ of ResMaxExact2Sat (with no unit clauses and no negations), translate $F$ to an instance $G$ of Independent Set Max Cover by translating each variable to a vertex and each clause to an edge.

If there is an independent set $V'$ in $G$, which covers $k$ edges, set the variables in $F$ corresponding to vertices in $V'$ to *true* (and all other variables to *false*); the $k$ clauses corresponding to the covered edges will then be satisfied, and no clauses will be oversatisfied as $V'$ is an independent set. If there is an assignment to the variables in $F$ satisfying $k$ clauses, the vertices in $G$ corresponding to *true* variables constitute an independent set covering $k$ edges.

Given an instance $G$ of Independent Set Max Cover, simply translate each vertex to a variable and each edge to a clause to obtain an instance $F$ of ResMaxExact2Sat (with no unit clauses and no negations). The argument that there is an independent set covering $k$ edges in $G$ if and only if there is an assignment to the variables in $F$ satisfying $k$ clauses is exactly as before. $\square$

We note that the question of whether there is an allowed assignment to a ResMaxExactSat instance at all (i.e. an assignment that does not oversatisfy any clause) can be reduced to 2SAT and hence is in polynomial time. The assignments which are not allowed are the ones that assign *true* to two literals occurring in the same clause. This can be expressed in 2SAT as follows: if literal $x$ and literal $y$ occur in a clause together, construct the 2SAT-clause $(\bar{x}, \bar{y})$. The size of the 2SAT-formula is bounded by $(2n)^2$, which is polynomial, so as 2SAT is in P, so is the question of whether there is an allowed assignment to the ResMaxExactSat instance.

# 4 Algorithms

For MaxExact2Sat (allowing negations and unit clauses) we give an algorithm in the number of clauses with time complexity $O(poly(L) \cdot 2^{m/4})$, which is also the currently best known for MaxCut [7] in the number of edges.

For the general ResMaxExactSat problem, we give an algorithm in the number of variables with time complexity $O(poly(L) \cdot 1.324718^n)$.

Given a formula $F$, we branch on a variable $x$ by either setting $x$ to *true* (this formula is denoted $F[x]$) or setting $x$ to *false* ($F[\bar{x}]$) and calling recursively on these smaller formulas. The algorithm then reduces the parameter before branching again. This gives rise to a branching tree. The algorithm for MaxExactSat for instance is analysed with respect to the parameter $m$, and in most cases we can remove four clauses from each of the smaller formulas before branching again, so we get the branching vector $(4, 4)$ corresponding to a recurrence inequality for the running time of the form $T(m) \leq 2 \cdot T(m-4) + poly(L)$. For further introduction to branching algorithms see for instance the paper by Kullmann and Luckhardt [12]. However in one case, we are only able to get the recurrence inequality $T(m) \leq 2 \cdot T(m-3) + poly(L)$. We show that this bottleneck situation can only arise once in any path from the root to a leaf in

**SIMPLIFY1**$(F)$:

$/*$ returns a simplified formula $F'$ along with the number $k$ of removed exactly satisfied clauses $*/$

```
if there are clauses (x) and (x̄) then
  F ← (F with the two above clauses deleted)
  (F′, k) ← SIMPLIFY1(F)
  return (F′, k + 1)
end if
if some variable x occurs in at least as many unit clauses as 2-clauses then
  F ← (F with x set to satisfy the unit clauses and these deleted)
  (F′, k) ← SIMPLIFY1(F)
  return (F′, k + the number of unit clauses with x)
end if
if there is a unique variable x then
  F ← (F with the clause in which x occurs deleted)
  (F′, k) ← SIMPLIFY1(F)
  return (F′, k + 1)
end if
return (F, 0)
```

Figure 2: The simplification algorithm for MaxExact2Sat.

the branching tree. We can thus omit the corresponding recurrence inequality from the analysis of the running time.

## 4.1 Algorithm for MaxExact2Sat

We present an algorithm for the MaxExact2Sat problem with time complexity $O(poly(L) \cdot 2^{m/4})$. As we will never branch on a unit clause, we work with $m$ as the number of 2-clauses in the analysis of the algorithm. Since the number of 2-clauses is of course smaller than or equal to the total number of clauses, the result is also valid for $m$ equal to the total number of clauses. This algorithm also directly gives us the time bound $O(poly(L) \cdot 2^{L/8})$ in the length of the formula, as $L \geq 2m$, where $m$ is the number of 2-clauses.

When calling an algorithm recursively on a formula with a variable set to *true* or *false*, we replace any clause $(true, y)$ by the clause $(\bar{y})$ and any clause $(false, y)$ by $(y)$. Any unit clauses $(true)$ or $(false)$ are simply removed and the number of *true*-clauses is added to the total.

We first introduce an algorithm, which simplifies the formula; see figure 2.

**Theorem 5.** *The algorithm SIMPLIFY1 is correct and runs in polynomial time.*

*Proof.* The three simplifications in the algorithm are the exact same ones used in the proof of Theorem 2. The proof that these simplifications are sound and

**ME2S**($F$):

$(F', k) \leftarrow$ SIMPLIFY1($F$)
**if** the formula is not connected **then**
  call recursively on the separate formulas and return the sum of the results
**end if**
**if** all variables occur in at most two 2-clauses **then**
  solve the instance in polynomial time and return the result
**end if**
**if** there is a variable $x$ occurring in at least four 2-clauses **then**
  /* branch on $x$ (1) – branching vector at least $(4, 4)$ */
  return $k+$ max(ME2S($F'[x]$), ME2S($F'[\bar{x}]$))
**end if**
**if** some variable $x$ occurs in three 2-clauses, and one of the variables it occurs with (say $y$) occurs in only two 2-clauses (and possibly a unit clause) **then**
  /* branch on $x$ (2) – branching vector at least $(4, 4)$ */
  return $k+$ max(ME2S($F'[x]$), ME2S($F'[\bar{x}]$))
**end if**
/* all variables occur in three 2-clauses and zero, one or two unit clauses */
pick a variable $x$
/* branch on $x$ (3) – branching vector at least $(3, 3)$ */
return $k+$ max(ME2S($F'[x]$), ME2S($F'[\bar{x}]$))

Figure 3: The main algorithm for MaxExact2Sat.

that the simplification algorithm runs in polynomial time is also given in the proof of Theorem 2. □

The main algorithm for MaxExact2Sat can be seen in figure 3.

**Theorem 6.** *The algorithm ME2S solves the* MaxExact2Sat *problem in time poly*$(L) \cdot 2^{m/4}$ *corresponding to the worst case branching vector* $(4, 4)$.

*Proof.* SIMPLIFY1 is sound and runs in polynomial time, and after calling SIMPLIFY1 on a formula, all variables occur in more 2-clauses than unit clauses. If a formula is not connected, we can simply solve the separate parts and add the results. Theorem 2 states that an instance where all variables occur in at most two 2-clauses can be solved in polynomial time. After this if-statement, some variable occurs at least three times in a 2-clause, so the algorithm covers all possible cases.

We argue that the branching vectors stated in the comments in the algorithm in figure 3 are correct. In the first type of branch (marked (1) in the comments), $x$ occurs in at least four 2-clauses. When setting $x$ to either *true* or *false*, we remove all 2-clauses in which $x$ occur (we create the same number of unit clauses, but we do not count these). So both when setting $x$ to *true* and to *false*, we remove at least four 2-clauses.

In the second type of branch, $x$ occurs only in three 2-clauses, so by the above argument we only remove these three clauses when branching. However the variable $y$, which occurs with $x$, occurs in only two 2-clauses (and possibly a unit clause). If $y$ occurs in two 2-clauses and no unit clauses, then after branching on $x$ it will occur in a unit clause and a 2-clause, and $y$ can then be set in the simplification step of the recursive call and the 2-clause removed, so we get a branching vector of $(4, 4)$. If $y$ occurs in two 2-clauses and one unit clause, then after branching on $x$ it will occur in two unit clauses and a 2-clause. If the two unit clauses are opposite they cancel out and $y$ becomes unique and the 2-clause can then be removed (in the next simplification step), giving us a branching vector of $(4, 4)$. If the two unit clauses are identical, $y$ can simply be set (again in the next simplification step) and the 2-clause can be removed, again giving us a branching vector of $(4, 4)$.

In the third type of branch, as no other case applies, all variables occur in exactly three 2-clauses and some may occur in one or two unit clauses. In this case we can only get a branching vector of $(3, 3)$, however by lemma 1 below this case can only happen once in any path in the branching tree, which is why we can omit this branching vector from the analysis.

Omitting the $(3, 3)$ branching vector, the worst case branching vector is $(4, 4)$, which gives us a running time of $O(poly(L) \cdot 2^{m/4})$. □

**Lemma 1.** *The formula can only become 3-regular (all variables occur in exactly three 2-clauses and zero, one or two unit clauses) one time in any path (from the root) in the branching tree of algorithm ME2S.*

*Proof.* When the formula has been 3-regular once, all variables occur in at most three 2-clauses in the rest of the algorithm, as no assignment makes a variable occur more times. If we look at the variables as vertices and the 2-clauses as edges, we have a 3-regular graph. As the formula is connected, so is the graph.

Now any assignment to some of the variables, will remove these variables (vertices) as well as the 2-clauses (edges) in which they occur. As the graph (formula) is connected, this means that we must have removed some edges to the rest of the graph (the unassigned variables), and there must now be some vertices with degree less than three, i.e. there must be some variables, that occur in less than three 2-clauses. □

**Corollary 1.** *The algorithm ME2S solves the* MaxExact2Sat *problem in time* $O(poly(L) \cdot 2^{L/8})$.

### 4.2 Algorithm for ResMaxExactSat

We give an algorithm for the ResMaxExactSat problem running in time $O(poly(L) \cdot 1.324718^n)$, where $n$ is the number of variables. This algorithm

**SIMPLIFY2**$(F)$:

**if** there are clauses $(x)$ and $(\bar{x})$ **then**
  $F \leftarrow$ ($F$ with the two above clauses deleted)
  $(F', k) \leftarrow$ SIMPLIFY2$(F)$
  **return** $(F', k+1)$
**end if**
**if** there is a variable $x$ occurring in $i$ unit clauses and no other clauses **then**
  $F \leftarrow$ ($F$ with these clauses deleted)
  $(F', k) \leftarrow$ SIMPLIFY2$(F)$
  **return** $(F', k+i)$
**end if**
**if** there are clauses $(x, y)$ and $(\bar{x}, y)$ **then**
  $F \leftarrow$ ($F$ with $y$ set to false and the two above clauses deleted)
  $(F', k) \leftarrow$ SIMPLIFY2$(F)$
  **return** $(F', k+1)$
**end if**
**if** there are clauses $(x, y)$ and $(\bar{x}, \bar{y})$ **then**
  $F \leftarrow$ ($F$ with $y$ set to $\bar{x}$ and the two above clauses deleted)
  $(F', k) \leftarrow$ SIMPLIFY2$(F)$
  **return** $(F', k+2)$
**end if**
**if** there is a variable $x$ occurring only with one other variable $y$ **then**
  /$*$ we can assume that $x$ and $y$ occur unnegated together (see proof) $*$/
  **if** there are at least as many $(\bar{x})$-clauses as $(x, y)$-clauses **then**
    $F \leftarrow$ ($F$ with $x$ set to *false*)
  **else**
    $F \leftarrow$ ($F$ with $x$ set to $\bar{y}$)
  **end if**
  **return** SIMPLIFY2$(F)$
**end if**

Figure 4: The simplification algorithm for ResMaxExactSat.

works for the general ResMaxExactSat problem, i.e. we have no restrictions on the clause size.

When calling this algorithm recursively on a formula with a variable set to *true* or *false*, we give all variables occurring in a clause with *true* an assignment (if the variable occurs unnegated, it is set to *false* otherwise to *true*), as oversatisfying is not allowed. Any occurrences of *false* are simply removed, and any unit clauses (*true*) or (*false*) are removed and the number of *true*-clauses is added to the total. If we get two occurrences of *true* in the same clause, this branch does not lead to an allowed assignment; for the purpose of the algorithm below, we will just assume that $-\infty$ is returned.

We first introduce an algorithm, which simplifies the formula; see figure 4.

**Theorem 7.** *The algorithm SIMPLIFY2 is correct and runs in polynomial time.*

*Proof.* We argue that the simplifications are correct. If we have two opposite unit clauses $(x)$ and $(\bar{x})$, no matter what $x$ is set to we will satisfy exactly one of these, so we can simply delete both and add one to the total. If there is a variable $x$ occurring only in $i$ unit clauses, we can simply satisfy all of these, without affecting any other variables.

If there are clauses $(x, y)$ and $(\bar{x}, y)$, $y$ must be *false*, or we would oversatisfy one of the clauses. So we set $y$ to *false* and delete both clauses, one of which will be exactly satisfied, regardless of the assignment to $x$. If there are clauses $(x, y)$ and $(\bar{x}, \bar{y})$, we must set $y$ equal to $\bar{x}$, or one of the two clauses would be oversatisfied. We have then exactly satisfied both clauses, and we can delete them and add two to the total. Now if two variables $x$ and $y$ occur together in two clauses, the clauses must be identical (otherwise one of the two above rules would have been used), thus we can assume that any clause with $x$ and $y$ is of the form $(x, y)$ (otherwise rename the literals).

If a variable $x$ occurs only together with one other variable $y$ in one or more clauses $(x, y)$, these clauses can always be satisfied simply by setting $x = \bar{y}$. If $x$ also occurs unnegated in a number of unit clauses or negated in some unit clauses, but less than there are 2-clauses $(x, y)$, we shall still set $x = \bar{y}$, as if $y$ is later set to *true*, this forces $x$ to be *false* and if $y$ is later set to *false*, we can satisfy all the 2-clauses with $x$ by setting $x$ to *true*; the unit clauses with $x$ will then also be satisfied; whereas the unit clauses with $\bar{x}$ will not, but we prefer to satisfy the 2-clauses, as there are more of these. If on the other hand $x$ also occurs negated in at least as many unit clauses as there are 2-clauses $(x, y)$, we set $x =$ *false*, as if $y$ is later set to *true* this is a must and in any case we satisfy as least as many clauses setting $x$ to *false* as we could setting it to *true*.

To prove that the algorithm runs in polynomial time we simply note that each if-statement can be checked and performed in polynomial time in the length of the formula and removes either a clause or a variable. Thus at most $m + n$ simplifications can be performed, and SIMPLIFY2 runs in polynomial time. □

The main algorithm can be seen in figure 5.

**Theorem 8.** *The algorithm RMES solves the* ResMaxExactSat *problem in time $O(poly(L) \cdot 1.324718^n)$ corresponding to the worst case branching vector $(5, 1)$.*

*Proof.* The correctness of the algorithm follows from first observing that SIMPLIFY2 is sound and polynomial, and after simplification no variable occurs with only one other variable, so one of the four branching cases apply. If a

**RMES**$(F)$:
$(F', k) \leftarrow$ SIMPLIFY2$(F)$
**if** the formula contains four or less different variables **then**
  solve the problem directly (try all assignments) and return the result
**end if**
**if** the formula is not connected **then**
  call recursively on the separate formulas and return the sum of the results
**end if**
**if** there is a variable $x$ occurring with at least 4 other variables **then**
  /* branch on $x$ (1) – branching vector at least $(5, 1)$ */
  return $k+$ max(RMES$(F'[x])$, RMES$(F'[\bar{x}])$)
**end if**
**if** there is a variable $x$ occurring with 3 other variables, one of which occurs only with $x$ and one other variable **then**
  /* branch on $x$ (2) – branching vector at least $(4, 2)$ */
  return $k+$ max(RMES$(F'[x])$, RMES$(F'[\bar{x}])$)
**end if**
**if** there is a variable $x$ occurring with 3 other variables **then**
  /* branch on $x$ (3) – branching vector $(4, 1)$ */
  return $k+$ max(RMES$(F'[x])$, RMES$(F'[\bar{x}])$)
**end if**
/* all variables occur with 2 other variables */
pick a variable $x$
/* branch on $x$ (4) – branching vector at least $(3, 3)$ */
return $k+$ max(RMES$(F'[x])$, RMES$(F'[\bar{x}])$)

Figure 5: The main algorithm for ResMaxExactSat.

formula contains only four different variables, it can be solved directly in polynomial time. If a formula is not connected, we can simply solve the separate parts and add the results.

Secondly we argue that the branching vectors stated in the comments are correct. If a variable $x$ occurs with at least four other variables, just branching on this variable immediately yields a branching vector of at least $(5, 1)$, as setting $x$ to *true* forces the literals occurring with the literal $x$ to be *false*, and setting $x$ to *false* forces the literals occurring with the literal $\bar{x}$ to be *false*. This means that if $x$ occurs with exactly four other variables, we get branching vectors $(3, 3)$ if $x$ occurs negated with two other variables, $(4, 2)$ if $x$ occurs negated with one other variable and $(5, 1)$ if $x$ does not occur negated with another variable.

If there is a variable $x$ occurring with three other variables, one of which occurs only with $x$ and one other variable, we have one of the following situations. The variable $x$ can occur in three 2-clauses. Let $\tilde{x}$ denote either $x$ or $\bar{x}$. So we have $(\tilde{x}, y)$, $(\tilde{x}, z)$, $(x, w)$ and possibly some unit clauses with $x$. Say $y$

occurs only with one other variable in $(\tilde{y}, u)$. When setting $x$ to *true* we also remove $w$, and when $\tilde{x}$ is set to *true* we also remove $y$ and $z$. When $\tilde{x}$ is set to *false*, $y$ occurs only with $u$ and can be removed in the following simplification procedure, so we remove $x$ and $y$ in both branches and $z$ and $w$ in one branch and get a branching vector of at least $(4, 2)$ (notice that a negation on the second $x$ will result in $(3, 3)$). Otherwise $x$ can occur in a 3-clause and a 2-clause: $(\tilde{x}, y, z)$, $(\tilde{x}, w)$ (and possibly some unit clauses with $x$). If $y$ occurs with no other variables, setting $\tilde{x}$ to *false*, will leave $y$ occurring with only one variable, and we can remove it giving us a branching vector of at least $(4, 2)$ again. The same is the case if $w$ occurs with only one other variable. Notice that $x$ can not occur in a 4-clause in this case, as all of the variables which it occurs with would then also occur with three variables.

If there is still a variable $x$ occurring with three other variables, all of these three variables must also occur with exactly two other variables besides $x$: if one of them occurred with four or more different variables it would have been caught by (1), if one of them occurred with only $x$ and one other variable it would have been caught by (2) and if one of them occurred only with $x$ it would have been caught by SIMPLIFY2. Branching on $x$ only yields a branching vector of $(4, 1)$. However by lemma 2 below this case can only happen once in any path in the branching tree, which is why we can omit this branching vector from the analysis.

If we reach branch (4), all variables occur with only two other variables, and there can only be 2-clauses and unit clauses, as any 3-clauses would be disconnected from the rest of the graph, and would have been caught by the second and subsequently the first if-statement. So we have the case where $x$ occurs in $(x, y)$ and $(\tilde{x}, z)$ and possibly some unit clauses, and $y$ and $z$ must occur with another variable as we could otherwise have removed them. If $y$ and $z$ occur together, we again have a closed sub-formula, which would have been removed by the second and first if-statements, so assume they occur in $(\tilde{y}, u)$ and $(\tilde{z}, v)$. Now setting $x$ to *true* will force $y$ to be *false*, and setting $x$ to *false* will leave $y$ occurring with only one other variable, so also in this branch it can be removed. If $x$ is not negated, the situation is the same for $z$, and if $x$ is negated the opposite occurs, but in both cases $z$ is also removed in both branches. This gives us a branching vector of $(3, 3)$.

We remark that when all variables occur with only two other variables we can actually solve ResMaxExactSat in polynomial time as the 2-clauses form a cycle. This is, however, not necessary for the analysis of the algorithm.

Since the branch resulting in branching vector $(4, 1)$ can only happen once in any path in the branching tree, it only has the effect of a constant on the running time. The worst case branching vector is then $(5, 1)$, which gives us the running time of $O(poly(L) \cdot 1.324718^n)$. $\square$

**Lemma 2.** *The formula can only become 3-regular (all variables occur with exactly three other variables) one time in any path (from the root) in the branching tree of algorithm RMES.*

*Proof.* When the formula has been 3-regular once, all variables occur with at most three other variables in the rest of the algorithm, as no assignment makes a variable occur with more other variables. Remember also that the formula is connected.

Now any assignment to some of the variables, will remove these variables as well as the clauses in which they occur. As the formula is connected, this means that we must have removed at least one clause with an unassigned variable, and there must now be at least one variable that occurs with less than three other variables. □

### 4.3 Parameterisation

The classic MaxSat problem is fixed parameter tractable (FPT). The currently best known parameterised algorithm for MaxSat has time complexity $O(L + k^2 \cdot 1.3695^k)$ [4]. That MaxSat is FPT is not surprising as we know that we can always satisfy at least half of the clauses (try an assignment; if this does not satisfy half the clauses, then the bitwise complement will), so in the interesting case $k > m/2$, which means that an algorithm in $m$ can be transformed to an algorithm in $k$. Of course such an algorithm is not very effective, but nevertheless shows that MaxSat is FPT. Mahajan and Raman [13] show that if we change the parameter by asking if $m/2 + k$ clauses can be satisfied instead, the MaxSat problem is still FPT.

For MaxExactSat it is not immediate, that any certain part of the clauses can always be satisfied, or even if the problem is fixed parameter tractable. However MaxExact2Sat with no unit clauses and no negations is equivalent to MaxCut, which is known to be fixed parameter tractable [14, 3].

The ResMaxExactSat problem has instances, where no clauses can be satisfied (a simple example is $(x, y) \wedge (x, \bar{y}) \wedge (\bar{x}, y)$), as any assignment oversatisfies some clause; thus there is not a certain part of the clauses that can always be satisfied and a parameterised algorithm for this problem is not immediate. We however give such an algorithm for the Independent Set Max Cover problem, which is equivalent to ResMaxExact2Sat with no unit clauses and no negations.

**Theorem 9.** *The* Independent Set Max Cover *problem is fixed parameter tractable.*

*Proof.* Given an instance $G = (V, E)$ of Independent Set Max Cover and a parameter $k$, the question is if $k$ edges can be covered by an independent set.

If some vertex has degree $k$ or more, we can simply pick this vertex as the independent set, and the answer is yes. Otherwise all vertices have degree at most $k-1$. Then the following simple branching-algorithm will work: Pick some vertex $v$ and branch on including either $v$ or one of its neighbours in the independent set. As $v$ has degree at most $k-1$, there are at most $k$ branches, and in each branch we include a vertex in the independent set, so we reduce the parameter by one, and we get at most $k$ recursive calls.

This gives us a time complexity of $O(poly(L) \cdot k^k)$, and Independent Set Max Cover is FPT. □

We note that for the ResMaxExact2Sat problem without unit clauses we can also easily handle negations. If a variable occurs both unnegated and negated, $k$ is decreased by at least one, both when this variable is set to *true* and to *false* (remember that oversatisfying is not allowed, so if we set a literal in a clause to *true*, we must satisfy this clause). This means that we can simply start by branching on the variables, which occur both negated and unnegated, and when there are no more of these, we can continue with the above algorithm. This is still a fixed parameter algorithm, and the time complexity is still bounded by $O(poly(L) \cdot k^k)$.

## 5 Conclusion

We have introduced two new maximum exact satisfiability problems, and proved them NP-complete. We have also given exact algorithms for both problems. Interesting open problems are if new algorithms with improved time complexities, perhaps in different parameters, can be devised, and if the two full problems are fixed parameter tractable?

## Acknowledgements

We would like to thank Daniel Mölle and Stefan Richter for their help on proofs and algorithms.

## References

[1] N. Bansal and V. Raman. Upper bounds for MaxSat: further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC 1999)*, volume 1741 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 1999.

[2] J. M. Byskov, B. A. Madsen, and B. Skjernaa. New algorithms for exact satisfiability. Report Series RS-03-30, BRICS, Department of Computer Science, Aarhus University, October 2003.

[3] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *J. Comput. Syst. Sci.*, 54(3):465–474, 1997.

[4] J. Chen and I. Kanj. Improved exact algorithms for MAX-SAT. In *Proc. of the 5th Latin American Symposium on Theoretical Informatics (LATIN'02)*, pages 341–355, Cancun, Mexico, 2002.

[5] V. Dahllöf, P. Jonsson, and R. Beigel. Algorithms for four variants of the exact satisfiability problem. Theoretical Comput. Sci. In press.

[6] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theoretical Comput. Sci.*, 141(1-2):109–131, 1995.

[7] S. S. Fedin and A. S. Kulikov. A $2^{|E|/4}$-time algorithm for MAX-CUT. *Zapiski nauchnyh seminarov POMI*, 293:129–138, 2002. Available at `http://logic.pdmi.ras.ru/~kulikov/maxcut_e.ps.gz`.

[8] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Comput. Sci.*, 1:237–267, 1976.

[9] J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith. Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. *Discrete Appl. Math.*, 130(2):139–155, 2003.

[10] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[11] A. S. Kulikov. An upper bound $O(2^{0.16254n})$ for exact 3-satisfiability: A simpler proof. *Zapiski nauchnyh seminarov POMI*, 293:118–128, 2002. English translation is to appear in Journal of Mathematical Sciences. Available at `http://logic.pdmi.ras.ru/~kulikov/x3sat_e.ps.gz`.

[12] O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity, 1997. Available at `http://www.cs.swan.ac.uk/~csoliver/tg.ps.gz`.

[13] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

[14] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computing and System Sciences*, 43:425–440, 1991.

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

[16] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

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

[18] R. Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Automata, Languages and Programming. 31st International Colloquium, ICALP 2004.*, volume 3142 of *Lecture Notes in Computer Science*. Springer, 2004.

# Recent BRICS Report Series Publications

**RS-04-19** **Bolette Ammitzbøll Madsen and Peter Rossmanith.** ***Maximum Exact Satisfiability: NP-completeness Proofs and Exact Algorithms*****. October 2004. 20 pp.**

**RS-04-18** **Bolette Ammitzbøll Madsen.** ***An Algorithm for Exact Satisfiability Analysed with the Number of Clauses as Parameter*****. September 2004. 4 pp.**

**RS-04-17** **Mayer Goldberg.** ***Computing Logarithms Digit-by-Digit*****. September 2004. 6 pp.**

**RS-04-16** **Karl Krukow and Andrew Twigg.** ***Distributed Approximation of Fixed-Points in Trust Structures*****. September 2004. 25 pp.**

**RS-04-15** **Jesús Fernando Almansa.** ***Full Abstraction of the UC Framework in the Probabilistic Polynomial-time Calculus ppc*****. August 2004.**

**RS-04-14** **Jesper Makholm Byskov.** ***Maker-Maker and Maker-Breaker Games are PSPACE-Complete*****. August 2004. 5 pp.**

**RS-04-13** **Jens Groth and Gorm Salomonsen.** ***Strong Privacy Protection in Electronic Voting*****. July 2004. 12 pp. Preliminary abstract presented at Tjoa and Wagner, editors,** ***13th International Workshop on Database and Expert Systems Applications*****, DEXA '02 Proceedings, 2002, page 436.**

**RS-04-12** **Olivier Danvy and Ulrik P. Schultz.** ***Lambda-Lifting in Quadratic Time*****. June 2004. 34 pp. To appear in** ***Journal of Functional and Logic Programming*****. This report supersedes the earlier BRICS report RS-03-36 which was an extended version of a paper appearing in Hu and Rodríguez-Artalejo, editors,** ***Sixth International Symposium on Functional and Logic Programming*****, FLOPS '02 Proceedings, LNCS 2441, 2002, pages 134–151.**

**RS-04-11** **Vladimiro Sassone and Paweł Sobociński.** ***Congruences for Contextual Graph-Rewriting*****. June 2004. 29 pp.**

**RS-04-10** **Daniele Varacca, Hagen Völzer, and Glynn Winskel.** ***Probabilistic Event Structures and Domains*****. June 2004. 41 pp. Extended version of an article to appear in Gardner and Yoshida, editors,** ***Concurrency Theory: 15th International Conference*****, CONCUR '04 Proceedings, LNCS, 2004.**