On the Parameterized Complexity of Exact Satisfiability Problems

Joachim Kneis Daniel Mölle Stefan Richter Peter Rossmanith

Dept. of Computer Science, RWTH Aachen University, Germany

Abstract. For many problems, the investigation of their parameterized complexity provides an interesting and useful point of view. The most obvious natural parameterization for the maximum satisfiability problem—the number of satisfiable clauses—makes little sense, because at least half of the clauses can be satisfied in any formula. We look at two optimization variants of the exact satisfiability problem, where a clause is only said to be fulfilled iff exactly one of its literals is set to *true*. Interestingly, these variants behave quite differently. In the case of ResMaxExactSAT, where over-satisfied clauses are entirely forbidden, we show fixed parameter tractability. On the other hand, if we choose to ignore over-satisfied clauses, the MaxExactSAT problem is obtained. Surprisingly, it is W[1]-complete. Still, restricted variants of the problem turn out to be tractable.

1 Introduction

Satisfiability of boolean formulæ in conjunctive normal form — abbreviated to SAT — is one of the most preeminent problems in computer science to this day. In its most common form, it asks for an assignment to a set of boolean variables $x_1, \dots, x_n$ that satisfies the formula, which is a conjunction of disjunctive clauses of literals over $x_1, \dots, x_n$. However, there are many important variations, some of which will be discussed below. All of them are NP-complete just as the original problem [4, 7, 10, 14].

A simple restriction is to bound the length of the clauses by a natural number q, resulting in a problem called q-SAT. It is well-known that 2-SAT is in P whereas 3-SAT is NP-complete.

In the exact satisfiability problem — ExactSAT — a clause is only counted as satisfied if exactly one of its literals is fulfilled. In this case we say that the clause is *exact-satisfied*. This variety, often combined with restrictions on the length of clauses as above, has been investigated by many authors [1, 2, 8, 10, 13, 14].

MaxSAT, on the other hand, is the natural generalization of SAT as an optimization problem. That is, given a formula F, the question is *how many* clauses can be satisfied simultaneously. Here, too, restrictions on the length of clauses have been analyzed in addition to the problem itself [4, 5, 15].

Of course, ExactSAT can be transformed into an optimization problem as well. There are, however, two natural ways to do this. In the first case, over-satisfied clauses are ignored, and the derived problem is called MaxExactSAT.

In the second case, over-satisfied clauses are forbidden, yielding a problem called ResMaxExactSAT [9, 10]. The first variety turns out to be a generalization of the well-known problem MaxCut.

In this paper, we analyze both MaxExactSAT and ResMaxExactSAT in terms of parameterized complexity [3], where the number of clauses that can be satisfied simultaneously gives the most natural parameter. Note that, in opposition to MaxSAT, both problems are non-trivial for small values of this parameter. In MaxSAT, either an arbitrary assignment or its complement already satisfy at least half of the clauses. Hence, MaxSAT belongs to FPT [11].

There are formulæ with arbitrary many clauses, such that no clause can be exact-satisfied. Therefore, MaxExactSAT is not trivially solvable for small parameters. Similarly, we can easily construct an instance for ResMaxExactSAT such that all assignments are illegal.

The key results achieved in this paper are as follows. MaxExactSAT is complete for W[1], but in FPT when restricted to clauses of bounded length or to monotonicity (MonoMaxExactSAT). In contrast, ResMaxExactSAT is in FPT even on unrestricted clauses.

2 Preliminaries

In parameterized complexity every input instance has an associated natural number, called the *parameter*. Often, the parameter is part of the input, otherwise it is a — usually simple — function of the input. The time complexity of a parameterized problem is measured as a function in both the input length n and the parameter k. The idea behind parameterized complexity theory is that hard problems can be easy in practice if hard instances do not occur. The parameter measures how hard an instance is; the complexity explodes in terms of the parameter only. See the influential monograph by Downey and Fellows [3] for an introduction to parameterized complexity.

Definition 1. $\mathcal{L}$ is a parameterized language iff $\mathcal{L} \subseteq \Sigma^* \times \mathbf{N}$ for some alphabet Σ. Let $(x, k) \in \mathcal{L}$, then x is called instance and k is called parameter. The language $\mathcal{L}$ is fixed parameter tractable iff there is an algorithm that decides whether $(x, k) \in \mathcal{L}$ in $O\big(f(k)p(|x|)\big)$ for some function $f : \mathbf{N} \to \mathbf{N}$ and a polynomial p.

FPT is the complexity class consisting of all fixed parameter tractable languages. To prove the intractability of problems, we need a reduction:

Definition 2. Let $\mathcal{L} \subseteq (\Sigma^* \times \mathbf{N}), \mathcal{L}' \subseteq (\Sigma'^* \times \mathbf{N})$, be parameterized languages. A mapping $f : \mathcal{L} \to \mathcal{L}'$ is an fpt-reduction iff

1. $(x, k) \in \mathcal{L} \Leftrightarrow f(x, k) \in \mathcal{L}'$
2. there is a computable function $g \colon \mathbf{N} \to \mathbf{N}$, such that $k' \leq g(k)$ for every $(x, k) \in \mathcal{L}$ with $f(x, k) = (x', k')$
3. there is a constant c and a computable function f', such that $f(x, k)$ can be computed in time $f'(k)|x|^c$

In short, we write $\mathcal{L} \leq^{\text{fpt}} \mathcal{L}'$ if there is an fpt-reduction from $\mathcal{L}$ to $\mathcal{L}'$. Downey and Fellows used this reduction to establish a class hierarchy for parameterized languages. Each of the classes in this hierarchy is defined by a key language which is complete for its class. Results on membership and hardness of other languages are then shown by fpt-reduction. We obtain the classes

$$\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \ldots \subseteq \text{W}[\text{SAT}] \subseteq \text{W}[\text{P}].$$

Note that none of the inclusions is known to be strict. Problems that are hard for any of the classes above FPT are called intractable as they probably admit no FPT algorithm.

Some examples for parameterized languages are the well-known VERTEX-COVER, INDEPENDENTSET and DOMINATINGSET. The first is known to be in FPT, while the last is W[2]-complete. Later, we will use INDEPENDENTSET, which is complete for W[1], to prove our hardness results.

We use the following notation for boolean formulæ: Let V be a set of boolean variables. The negation of a variable x is denoted by $\bar{x}$. *Literals* can be variables or their negations. If l denotes a negated variable $\bar{x}$, then $\bar{l}$ denotes the variable x.

Algorithms for finding the exact solution of MAXSAT are usually designed for the unweighted MAXSAT problem. However, we represent formulæ by multisets to account for positive integer weights.

Definition 3. A CNF formula is a multiset of clauses. A clause is a multiset of literals.

We call a formula *monotone* if each of its variables appears only in positive or only in negative form. Monotone formulæ are uninteresting for traditional MAXSAT, but become challenging if at most one literal per clause may be satisfied.

Definition 4. An assignment *exact-satisfies* a clause C iff it sets exactly one literal in c to *true*. A clause C is *over-satisfied* by an assignment iff two or more of its literals are set to *true*.

This yields the two problems:

Definition 5.

1. The MAXEXACTSAT problem is the question whether there is an assignment that exact-satisfies at least k clauses in a given formula F.
2. The RESMAXEXACTSAT problem asks for an assignment that exact-satisfies at least k clauses under the restriction that no clause is over-satisfied.

Let us analyze the following example:

$$F = \big\{\{y\}, \{z\}, \{x, y\}, \{x, z\}, \{x, y, z\}\big\}$$

In the case of MAXSAT, we can satisfy every clause of F. An optimal assignment for MAXEXACTSAT satisfies four clauses $(x = 0, y = 1, z = 1)$ while RESMAXEXACTSAT only admits an assignment that satisfies three clauses $(x = 1, y = 0, z = 0)$.

Another difference between MAXSAT and these new problems is the way double occurences of variables are handled. The clause $C = \{x, x, y, z\}$ is equivalent to $\{x, y, z\}$ in traditional MAXSAT. In RESMAXEXACTSAT, the clause $C = \{x, x, y, z\}$ translates to $\{y, z\}$ and $x = 0$, because $x = 1$ is an invalid assignment. In MAXEXACTSAT, however, C is not equivalent to any shorter clause. All we can say is that if x is *true*, then C cannot be satisfied.

3 Maximum Exact Satisfiability

The upcoming lemma establishes W[1]-hardness for MAXEXACTSAT. Observe that the proof does not employ variables that occur twice or more in a clause. That is, the result does hold even for the restricted variant where clauses are sets rather than multisets.

Lemma 1. INDEPENDENTSET $\leq^{\text{fpt}}$ MAXEXACTSAT.

Proof. Let $G = (V, E)$ a graph with $V = \{v_1, \dots, v_n\}$. We construct a formula $F = \{C_1, \dots, C_n\}$ over variables $x_1, \dots, x_n, z$ such that C_i is satisfied iff v_i belongs to an independent set. Moreover, we ensure that C_i can only be exact-satisfied by $x_i = 1$ and $x_j = 0$ for all $v_j \in N(v_i)$, where $N(v_i)$ denotes the neighborhood of v_i, excluding v_i itself. For every node v_i, define the clause

$$C_i := \{z, \bar{z}, \bar{x}_i\} \cup \bigcup_{v_j \in N(v_i)} \{x_j\}.$$

Note that C_i is exact-satisfied if and only if $x_i = 1$ and $x_j = 0$ for every j with $v_j \in N(v_i)$. Let $F := \{C_1, \dots, C_n\}$. We now show that G contains an independent set of size k if and only if it is possible to exact-satisfy k clauses in F:

Let first $I = \{v_{i_1}, \dots, v_{i_k}\} \subseteq V$ be an independent set in G. Set $x_j = 1$ for every $v_j \in I$ and $x_j = 0$ otherwise. This assignment exact-satisfies C_j if $v_j \in I$ because only x_j is set to *true* in C_j.

Conversely, let $X = \{x_{a_1}, \dots, x_{a_l}\}$ be a set of variables set to *true*, such that at least k clauses are exact-satisfied. We obviously have $l \geq k$, because $x_i = 1$ holds for every exact-satisfied C_i. Now set $I = \{v_i \mid C_i \text{ is satisfied}\}$. By construction, $x_i = 1$ for every $v_i \in I$. Then I is an independent set, since if $v_i \in N(v_j)$ for some $v_i \in I$, then at least two literals are satisfied in

$$C_j = \{z, \bar{z}, \bar{x}_j, x_i\} \cup \bigcup_{\substack{v_r \in N(v_j) \\ r \neq i}} \{x_r\}.$$

□

For example, applying the reduction from the proof of Lemma 1 to the graph depicted in Figure 1 constructs the formula

$$\big\{\{z, \bar{z}, \bar{x}_1, x_2, x_3\}, \{z, \bar{z}, \bar{x}_2, x_1, x_3\}, \{z, \bar{z}, \bar{x}_3, x_1, x_2, x_4\}, \{z, \bar{z}, \bar{x}_4, x_3\}\big\}.$$

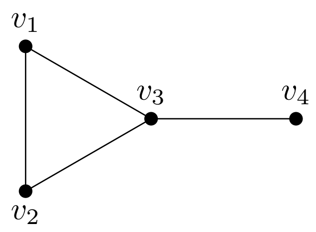

Fig. 1. A graph with an independent set of size two

It turns out that MaxExactSAT is hard even for monotone formulæ. However, our proof uses clauses that are multisets of literals rather than sets of literals. Later, we will see that this is unavoidable.

Lemma 2. IndependentSet $\leq^{\text{fpt}}$ MonoMaxExactSAT

Proof. Let $G = (V, E)$ a graph with $V = \{v_1, \ldots, v_n\}$. We construct a formula F over variables $x_1, \ldots, x_n$ such that C_i is satisfied iff v_i belongs to an independent set. Furthermore, we ensure that C_i can only be exact-satisfied by x_i. For every node v_i, define the clause

$$C_i := \{x_i\} \cup \bigcup_{v_j \in N(v_i)} \{x_j, x_j\}.$$

As in the proof of Lemma 1, C_i is exact-satisfied if and only if $x_i = 1$ and $x_j = 0$ for every $j \neq i$ with $v_j \in N(v_i)$. Let $F := \{C_1, \ldots, C_n\}$. It is easy to see that G has an independent set of size k if and only if k clauses in F can be satisfied simultaneously. □

If, for instance, we apply this reduction to the graph given in Figure 1, we obtain the formula

$$\big\{\{x_1, x_2, x_2, x_3, x_3\}, \{x_2, x_1, x_1, x_3, x_3\}, \{x_3, x_1, x_1, x_2, x_2, x_4, x_4\}, \{x_4, x_3, x_3\}\big\}.$$

Let us now further investigate the complexity of MaxExactSAT. In order to obtain completeness for W[1], we need to add containment to hardness.

Lemma 3. MaxExactSAT $\leq^{\text{fpt}}$ IndependentSet

Proof. The idea of the proof is to construct a graph G for a formula F, such that the nodes corresponding to literal occurences that satisfy clauses in F form an independent set in G. The graph is built from subgraphs S_i representing clauses, where each literal occurence in a clause C_i is translated to a node in the respective subgraph S_i. Clauses C_i that are oversatisfied for every possible assignment can be detected easily and are ignored in the following reduction. A critical property of the construction is that the parameter does not change at all.

For the moment, assume that the clauses are but sets, rather than multisets. Given the formula $F = \{C_1, \ldots, C_m\}$ with $C_i = \{l_{i,1}, \ldots, l_{i,t_i}\}$, we construct the graph $G = (V, E)$ as follows:

$$V = \{\, v_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq t_i \,\}$$
$$E = E_1 \cup E_2 \cup E_3 \cup E_4$$

with

$$E_1 := \{\, \{v_{i,j}, v_{a,b}\} \mid i = a \,\} \tag{1}$$
$$E_2 := \{\, \{v_{i,j}, v_{a,b}\} \mid l_{i,j} = \bar{l_{a,b}} \,\} \tag{2}$$
$$E_3 := \{\, \{v_{i,j}, v_{a,b}\} \mid l_{i,j} = l_{a,t}, t \neq b \,\} \tag{3}$$
$$E_4 := \{\, \{v_{i,j}, v_{a,b}\} \mid l_{i,s} = \bar{l_{a,t}}, s \neq j, t \neq b\} \tag{4}$$

The edges from E_1 make every S_i a clique, thus guaranteeing that at most one node in S_i is selected for the independent set, preventing C_i from being over-satisfied. By E_2 it is impossible for l and $\bar{l}$ to satisfy clauses simultaneously. Further consistency between variables occuring together in a clause is ensured by E_3 and E_4.

In order to prove the claim $(F, k) \in$ MaxExactSAT $\Leftrightarrow (G = (V, E), k) \in$ IndependentSet, let first $\lambda = (\lambda_1, \ldots, \lambda_n)$ be an assignment exact-satisfying the clauses $C_1, \ldots, C_k$. Denote by l_{i,t_i} the unique literal satisfying C_i, and let $\mathcal{L}$ be the set of these literals. We want to show that $I = \{\, v_{i,t_i} \mid l_{i,t_i} \in \mathcal{L} \,\}$ is an independent set of size k. Note that I can be larger than $\mathcal{L}$, as $\mathcal{L}$ collects literals, versus nodes depicting literal occurences in I.

Clearly, $|I| = |L| = k$. It remains to show that no two nodes v_{i,t_i} and v_{j,t_j} from I are neighbors in G:

- Since l_{i,t_i} and l_{j,t_j} are from different clauses, we have $\{v_{i,t_i}, v_{j,t_j}\} \notin E_1$.
- Since l_{i,t_i} and l_{j,t_j} satisfy different clauses, we have $l_{i,t_i} \neq \bar{l_{j,t_j}}$ and thus $\{v_{i,t_i}, v_{j,t_j}\} \notin E_2$.
- Since l_{i,t_i} exact-satisfies C_i and l_{j,t_j} exact-satisfies C_j, l_{i,t_i} cannot appear in C_j and l_{j,t_j} cannot appear in C_i. It follows that $\{v_{i,t_i}, v_{j,t_j}\} \notin E_3$.
- Since l_{i,t_i} exact-satisfies C_i and l_{j,t_j} exact-satisfies C_j, there cannot be a third literal l such that l appears in C_i and $\bar{l}$ in C_j. It follows that $\{v_{i,t_i}, v_{j,t_j}\} \notin E_4$.

Conversely, let $I \subseteq V$ be an independent set in G, $|I| = k$. Let $\mathcal{L} = \{\, l_{i,j} \mid v_{i,j} \in I \,\}$ be the respective literals and $X = \{x_1, \ldots, x_{k'}\}$, $k' \leq k$ the respective variables.

Due to (1), we have $i \neq a$ for any two nodes $v_{i,j}$ and $v_{a,b}$ from I. Hence, for $\mathcal{C} = \{C_i \mid \exists j.\ v_{i,j} \in I\}$ we get $|\mathcal{C}| = k$. By (2), we furthermore have $l_{i,j} \neq \bar{l_{a,b}}$ for any two $l_{i,j}$ and $l_{a,b}$ from $\mathcal{L}$. It is thus possible to satisfy both $l_{i,j}$ and $l_{a,b}$. According to (3), no clause from $\mathcal{C}$ contains more than one literal from $\mathcal{L}$. Therefore, every clause from $\mathcal{C}$ is exact-satisfied by $\mathcal{L}$.

Altogether, the rules (1) through (3) guarantee a sound assignment to the variables from X that satisfies exactly one literal per clause from $\mathcal{C}$. It remains to show that there is an assignment to the rest of the variables that does not interfere with this goal.

For any literal $l \notin \mathcal{L}$ from any clause in $\mathcal{C}$, (4) guarantees that $\bar{l}$ does not appear in $\mathcal{C}$. This can easily be seen by contradiction: assume the opposite. Then, there are clauses C_i and C_a with $l \in C_i$ and $\bar{l} \in C_a$. By construction, C_i is satisfied by l_{i,t_i}, and C_a is satisfied by l_{a,t_a}. In that case, however, we have $\{v_{i,t_i}, v_{a,t_a}\} \in E_4$, implying that I is not an independent set. This, of course, means that we can assign a value to l such that l does not satisfy any of the clauses from $\mathcal{C}$.

When the clauses are allowed to be multisets, caution must be taken. Consider for instance the subgraph S_i generated from a clause $C_i = \{x_1, x_1, x_2\}$. While the first two nodes in S_i may contribute to an independent set, the assignment $x_1 = 1$ leads to over-satisfaction of C_i. Fortunately, there is an easy way out: Following the above reduction, for each i, remove the nodes from S_i that originate from literals that occur multiple times in C_i. □

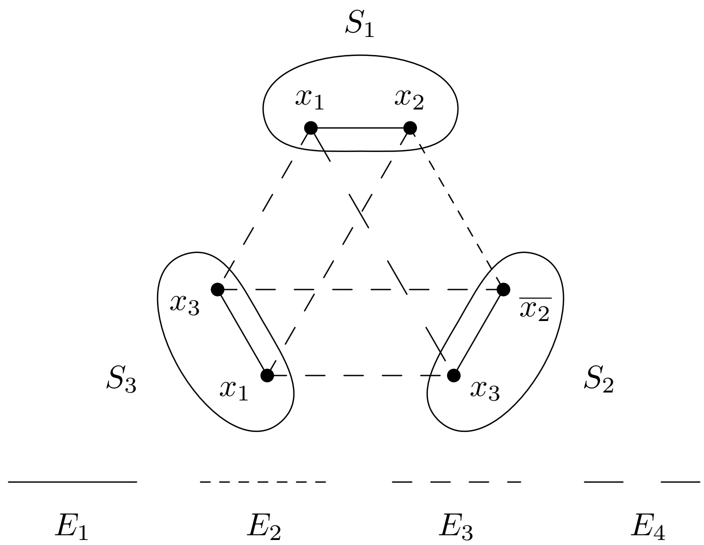

Fig. 2. Construction for $F = \Big\{\{x_1, x_2\}, \{\bar{x_2}, x_3\}, \{x_1, x_3\}\Big\}$

Figure 2 illustrates the construction on a small example. Combining Lemma 1 and Lemma 3 we obtain:

Theorem 1. MaxExactSAT *is complete for* W[1].

By Lemma 2 the problem remains W[1]-hard for monotone formulæ, if a literal can occur more than once in a clause. The complexity for monotone formulæ without repetition of literals will be dealt with in Section 4.

4 Monotone formulæ

Theorem 2. MonoMaxExactSAT $\in$ FPT, *if clauses are sets.*

Proof. Let F be a monotone formula. If F contains a variable x that occurs in k or more clauses, we can simply set x to *true* and all other variables to *false* in order to exact-satisfy at least k clauses. Also, if F consists of no more than k^2+k clauses, there are at most 2^{k^2+k} variables, and the problem can be solved using exhaustive search.

In the following, we thus assume that no variable is used k times, and that the instance has more than k^2+k clauses. In this case, the algorithm stated below always finds an assignment exact-satisfying at least k clauses. Hence, any instance with more than k^2+k clauses is in MonoMaxExactSAT.

1. Initially, construct the hypergraph $G(F)$ by introducing a node for each clause and a hyperedge for each variable, where each such hyperedge contains exactly the clauses in which the respective variable occurs.
2. Find a maximal matching M greedily. If the union of the hyperedges in M consists of k or more nodes, assign *true* to the respective variables. This yields a solution exact-satisfying at least k clauses, and we are done.
3. Otherwise, we distinguish two cases. In the first case, $G(F)$ contains a hyperedge of size at least three. Then, we assign *false* to all the variables represented by the edges in M and remove all the clauses contained therein. As a consequence, since M is a matching, every remaining hyperedge shrinks by one. Go to (2).
4. In the second case, $G(F)$ is but a graph. Then, the nodes not covered by M form an independent set I. Step (3) was performed at most $k-3$ times, and each time, we erased at most $k-1$ clauses. Therefore, less than k^2-k of the clauses have been removed, and more than $2k$ remain. On the other hand, M spans less than k nodes, and thus, we have that $|I|>k$. Because the clauses represented by I are disjoint, we can easily find an assignment that exact-satisfies all of them. □

The above result also affects another problem. While HittingSet is W[2]-complete, ExactHittingSet is in FPT when the size of the hitting set is the parameter. If the number of exact-hit sets is chosen instead, we get MaxExactHittingSet: Given a family $\mathcal{S}=\{S_1,\ldots,S_m\}$ of sets from a universe S, is there an $H \subseteq S$ such that at least k sets from $\mathcal{S}$ share exactly one element with H? Now MaxExactHittingSet $\in$ FPT, for it is isomorphic to MonoMaxExactSAT.

5 Clauses of bounded length

There is an FPT algorithm for MaxExactSAT if it is restricted to clauses with length two and no negated literals are allowed [10]. In the following we will improve this result to arbitrary formulæ with clause length bounded by a fixed q.

Lemma 4. *Let F be a formula in q-CNF. If some literal x occurs in at least $k \cdot 2^{q-1}$ clauses, an assignment of the variables exists that exact-satisfies at least k clauses.*

Proof. Let X be the set of clauses containing x, and let X^- denote the set of clauses derived from X by removing x from each clause. Observe that k clauses in X can be exact-satisfied by x iff there is an assignment of the variables that leaves k clauses from X^- unsatisfied. By linearity of expectation, the expected number of clauses from X^- exact-satisfied by a random assignment to the variables is

$$E \leq \left(1 - \frac{1}{2^{q-1}}\right) \cdot |X^-|.$$

By the first moment method [12], there is an assignment that satisfies at most E clauses. Thus, there are at least

$$|X^-| - E \geq \left(1 - \left(1 - \frac{1}{2^{q-1}}\right)\right) \cdot |X^-| \geq \frac{1}{2^{q-1}} \cdot k \cdot 2^{q-1} = k$$

clauses not fulfilled by this assignment. □

We can use this lemma to prove that every formula with more than $f(k)$ clauses has an assignment that exact-satisfies at least k clauses.

Lemma 5. *Let F be a formula in q-CNF with $|F| \geq k^2 q 2^q$. Then there is an assignment that exact-satisfies at least k clauses.*

Proof. If a literal occurs in at least $k2^{q-1}$ clauses, k clauses can be exact-satisfied according to Lemma 4.

Otherwise, choose an arbitrary clause $(l_1, \ldots, l_q)$ to exact-satisfy. Remove all clauses containing any of l_1 through l_q. Each literal occurs in no more than $k \cdot 2^{q-1}$ clauses. Thus, each variable is in at most $k \cdot 2^q$ clauses, so we remove no more than $q \cdot k \cdot 2^q$ of the clauses.

Iterating this process k times yields an assignment that exact-satisfies at least k clauses. □

Theorem 3. *There is a linear problem kernel for* MaxExact-q-SAT.

Proof. Consider the following algorithm. If $|F| \geq k^2 q 2^q$, return true. Otherwise, exhaustively search for a solution. This takes $O(2^{q^2 k^2 2^q} \cdot |F|)$ time, which is linear in $|F|$. □

6 Restricted Maximum Exact Satisfiability

We will now prove that ResMaxExactSAT is fixed parameter tractable. Our approach is based on an idea that was originally used in an algorithm for Set-Packing [6]. We will see in the end that the only critical case is monotone formulæ. Let us begin with the concept of an *approximator*.

Parameters: A monotone formula F, sets of clauses $M_1, \ldots, M_l$,
numbers $k_1, \ldots, k_l$

$P_i := M_i$ for every $i = 1, \ldots, l$;
for $i = 1, \ldots, l$ **do**
  if there is a variable x such that
    - x is in no clause of P_j for any $j \neq i$
    - x is in every clause of M_i
    - x occurs in exactly k_i clauses
  then $P_i := \mathit{clauses}(x)$;
  else guess a clause $C \in P_1 \cup \cdots \cup P_l - (M_1 \cup \cdots \cup M_l)$;
    $M_i := M_i \cup \{C\}$;
    return;
  fi
od;
$M_i := P_i$ for every $i = 1, \ldots, l$;
return;

Fig. 3. Subroutine *Improve*

Definition 6. Let F be a monotone formula. For any variable v_i, let O_i denote the set of clauses from F satisfied by v_i. We say that $(M_1, \ldots, M_l)$ is a $(k_1, \ldots, k_l)$-*approximator* for F if there is a set of variables $v_1, \ldots, v_l$, such that $O_1, \ldots, O_l$ are pairwise disjoint and for all $1 \leq i \leq l$ we have that $|O_i| = k_i$ and $M_i \subseteq O_i$. If $M_i = O_i$ for all $1 \leq i \leq l$, we call the approximator *saturated.*

The following lemma shows how the subroutine from Figure 3 is capable of improving approximators.

Lemma 6. *Let F be a monotone formula. If $(M_1, \ldots, M_l)$ is an unsaturated $(k_1, \ldots, k_l)$-approximator for F, then Improve$(F, M_1, \ldots, M_l, k_1, \ldots, k_l)$ adds elements to $M_1, \ldots, M_l$ such that $(M_1, \ldots, M_l)$ is still a $(k_1, \ldots, k_l)$-approximator for F, in at least one nondeterministic branch.*

Proof. There are two ways in which *Improve* adds elements to the M_i's, namely in the lines preceding the two **return** statements.

In the case that the second **return** statement is used, it is easy to see that $(P_1, \ldots, P_l)$ is a saturated $(k_1, \ldots, k_l)$-*approximator* for F.

If the first **return** statement is used in the i-th iteration of the **for**-loop, we get the following situation:

As $(M_1, \ldots, M_l)$ is a $(k_1, \ldots, k_l)$-approximator for F, there exist variables $v_1, \ldots, v_l$ and respective disjoint sets $O_1, \ldots, O_l$ of clauses, such that $M_i \subseteq O_i$. Observe that v_i cannot fulfill the three conditions of the **if**-statement, because the subroutine has entered the **else**-branch. However, v_i obviously fulfills the second and third condition and thus fails the first one. Therefore, there is a clause C that is in both O_i and some P_j, $j \neq i$, but not in $M_i = P_i$, because $P_1, \ldots, P_l$ are pairwise disjoint.

Input: A monotone formula F, numbers $k_1, \ldots, k_l$
Output: A set $\{v_1, \ldots, v_l\}$ of variables that corresponds to a saturated $(k_1, \ldots, k_l)$-approximator for F

$M_i := \emptyset$ for every $i = 1, \ldots, l$;
while $|M_1| + \cdots + |M_l| < k_1 + \cdots + k_l$ **do**
  Improve$(F, M_1, \ldots, M_l, k_1, \ldots, k_l)$
od;
if there are variables $v_1, \ldots, v_l$ such that v_i occurs exactly in the clauses M_i
then return $\{v_1, \ldots, v_l\}$
else return *no* **fi**

Fig. 4. Algorithm A

If the guess is correct, this C is added to M_i. Since we still have $M_i \subseteq O_i$, $(M_1, \ldots, M_l)$ remains a $(k_1, \ldots, k_l)$-approximator for F. □

The above subroutine may be profitably used in a nondeterministic algorithm that finds a feasible set $\{v_1, \ldots, v_l\}$ of variables such that each v_i satisfies exactly k_i many clauses, unless no such set exists. As before, no two of these variables are allowed to occur in the same clause.

Lemma 7. *Algorithm A returns "no" in all nondeterministic branches, if no solution exists. Otherwise, it returns correct solutions in at least one branch and "no" in all other branches.*

Proof. The algorithm always checks the correctness of a solution before returning it. Therefore, even on computation paths with wrong guesses no incorrect answer is possible.

By Lemma 6, the fact that $(M_1, \ldots, M_l)$ is a $(k_1, \ldots, k_l)$-approximator for F obviously constitutes an invariant of the **while**-loop, as long as all guesses are correct. □

In order to check whether exactly k clauses can be exact-satisfied, we need to test all possible partitions of the number k using Algorithm A. Unfortunately, there are formulæ that allow for satisfying, say, $k + 1$ clauses, but not exactly k. The following observation helps to overcome this difficulty in the final theorem.

Proposition 1. *Let F be a monotone formula in which no variable occurs in more than k clauses. If there is an assignment that exact-satisfies at least k clauses of F, then there is an assignment that exact-satisfies between k and $2k$ clauses.*

Theorem 4. *There is an algorithm that decides* ResMaxExactSAT *in time $O((2k)^{4k} \cdot poly(|F|))$. Hence,* ResMaxExactSAT *is in* FPT.

Proof. If F is not monotone, we can branch on a variable that occurs as a positive and a negative literal. This satisfies at least one clause in every step. The hard case is monotone formulæ, which we assume in what follows.

Try Algorithm A on F and all combinations of $k_1, \ldots, k_l$, such that $k \leq k_1 + \ldots + k_l \leq 2k$, $k \leq l \leq 2k$, and $k_i > 0$ for all $1 \leq i \leq l$. The number of these combinations is bounded by $(2k)^{2k}$. By Proposition 1, this approach suffices to find the solution as long as all guesses are correct.

For up to $2k$ times, Algorithm A guesses a clause from a set of at most $2k$ elements. Resolving the nondeterminism thus involves an additional factor of at most $(2k)^{2k}$ in the running time. □

Acknowledgements. We would like to thank several anonymous referees for helpful comments, especially regarding the final section.

References

1. J. M. Byskov, B. A. Madsen, and B. Skjernaa. New algorithms for exact satisfiability. Technical Report RS-03-30, BRICS, Oct. 2003.
2. V. Dahllöf, P. Johnson, and R. Beigel. Algorithms for four variants of the exact satisfiability problem. *Theoretical Comp. Sci.*, 320(2-3):373–394, 2004.
3. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
4. M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
5. J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith. New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. *Discrete Applied Mathematics*, 130(2):139–155, 2003.
6. W. Jia, C. Zhang, and J. Chen. An efficient parameterized algorithm for m-set packing. *Journal of Algorithms*, 50(1):106–117, 2004.
7. R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Communications*, pages 85–103. Plenum Press, 1972.
8. A. S. Kulikov. An upper bound $O(2^{0.16254n})$: A simpler proof. *Zapiski nauchnyh seminarov POMI*, 293:118–128, 2002.
9. B. Madsen. An algorithm for exact satisfiability analysed with the number of clauses as parameter. Technical Report RS-04-18, BRICS, 2004.
10. B. A. Madsen and P. Rossmanith. Maximum exact satisfiability: NP-completeness proofs and exact algorithms. Technical Report RS-04-19, BRICS, Oct. 2004.
11. M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31:335–354, 1999.
12. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
13. S. Porschen, B. Randerath, and E. Speckenmeyer. Exact 3-satisfiability is decidable in time $O(2^{0.16254n})$. In *Proc. 5th Int. Symp. on the Theory and Appl. Satisfiabilty testing (SAT2002)*, pages 231–235, 2002.
14. C. P. Schnorr. Satisfiability is quasilinear complete in NQL. *J. ACM*, 25:136–145, 1978.
15. R. Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Proc. of 31st ICALP*, number 3142 in LNCS, pages 1227–1237. Springer, 2004.