Parameterized complexity of constraint satisfaction problems

Dániel Marx[*]
Department of Computer Science and Information Theory,
Budapest University of Technology and Economics
H-1521 Budapest, Hungary
dmarx@cs.bme.hu

30th September 2004

Abstract

We prove a parameterized analog of Schaefer's Dichotomy Theorem: we show that for every finite boolean constraint family $\mathscr{F}$, deciding whether a formula containing constraints from $\mathscr{F}$ has a satisfying assignment of weight exactly k is either fixed-parameter tractable (FPT) or W[1]-complete. We give a simple characterization of those constraints that make the problem fixed-parameter tractable. The special cases when the formula is restricted to be bounded occurrence, bounded treewidth or planar are also considered, it turns out that in these cases the problem is in FPT for every constraint family $\mathscr{F}$.

1 Introduction

A *dichotomy theorem* in computational complexity shows that every problem in a certain family of problems is either polynomial-time solvable or NP-complete. The first such result is *Schaefer's Dichotomy Theorem* [15], which considers boolean constraint satisfaction. Let $\mathscr{F}$ be a finite set of boolean constraints, each constraint is a boolean relation of some finite arity. In the $\mathscr{F}$-SAT problem we are given a formula that consists of a conjunction of clauses, where each clause is a constraint from $\mathscr{F}$ on the variables. Our task is to decide whether the given formula has a satisfying assignment. For example, if $\mathscr{F} = \{(x \lor y \lor z), (\bar{x} \lor y \lor z), (\bar{x} \lor \bar{y} \lor z), (\bar{x} \lor \bar{y} \lor \bar{z})\}$, then $\mathscr{F}$-SAT is equivalent to 3SAT, as every 3CNF formula is a conjunction of such clauses. For every constraint family $\mathscr{F}$, the $\mathscr{F}$-SAT problem is a separate problem. Schaefer [15] determines the complexity of each of these infinitely many problems: it turns out that for

[*]Research is supported in part by grants OTKA 44733, 42559 and 42706 of the Hungarian National Science Fund.

every finite constraint family $\mathscr{F}$, the $\mathscr{F}$-SAT problem is either polynomial-time solvable or NP-complete.

There are several extensions of Schaefer's theorem in the literature. Bulatov [6] proved a dichotomy theorem similar to Schaefer's theorem, but his result classifies the complexity of the satisfiability problem with three-valued variables. However, extending Schaefer's theorem to variables with arbitrary domain is an important open problem (see [6, 10] for partial results).

Optimization variants of the boolean constraint satisfaction problem were also considered in the literature. First, Creignou [7] classified the approximability of the $\mathscr{F}$-MAX-SAT problem, where the goal is to maximize the number of clauses satisfied. Khanna et al. [12] classified three other families of problems: $\mathscr{F}$-MIN-SAT (minimize the number of unsatisfied clauses), $\mathscr{F}$-MAX-ONES (find a satisfying assignment with maximum number of true variables), $\mathscr{F}$-MIN-ONES (minimize the number of true variables). Notice that $\mathscr{F}$-MAX-SAT and $\mathscr{F}$-MIN-SAT are the same problem, but due to their different formulations, their approximability might be different.

In parameterized complexity we are dealing with problems where each problem instance has a distinguished part called the *parameter*. For example, in the parameterized maximum clique problem the parameter k is the size of the clique to be found. A parameterized problem is *fixed-parameter tractable* (FPT) if it can be solved in polynomial time for every fixed value of the problem parameter k, and moreover, the degree of the polynomial in the time bound does not depend on k. That is, a problem is in FPT, if it has an $f(k)n^c$ time algorithm, where c is independent of k and n. Such an algorithm is called *uniformly polynomial*. It turns out that the parameterized versions of several NP-hard problems are fixed-parameter tractable: for example, there are uniformly polynomial algorithms for the parameterized minimum vertex cover, longest path, and minimum feedback vertex set problems. In some cases, these algorithms are highly nontrivial.

By showing that a problem is NP-complete, we give strong evidence that it does not have a polynomial-time algorithm. There is a similar completeness program in parameterized complexity that allows us to show that certain problems are unlikely to be in FPT. A *parameterized reduction* from problem A to problem B transforms an instance x of A with parameter k to an instance x' of B with parameter k' such that x is a yes instance of A if and only if y is a yes instance of B. The reduction has to be computed in time $f(k)|x|^c$ (for some function f and constant c) and the new parameter k' has to be a function of k only. It is easy to see that if A is reducible to B, and B is in FPT, then it follows that A is in FPT as well. The class W[1] contains the parameterized problems that can be reduced to the problem "Does the given nondeterministic Turing machine accepts input x in at most k steps?" It is believed that W[1]-complete problems are not fixed-parameter tractable. For more background on parameterized complexity theory, the reader is referred to the monograph of Downey and Fellows [8].

In this paper we investigate the parameterized complexity of boolean constraint satisfaction problems. The parameterized satisfiability problem corresponding to 3SAT is WEIGHTED 3SAT. Here we are given a 3CNF formula ϕ together with an integer parameter k, and it has to be determined whether ϕ has a satisfying assignment with exactly k true variables. Clearly, the problem is polynomial-time solvable for fixed k, since we have to consider at most $O(n^k)$ possible solutions. However, WEIGHTED 3SAT is one of the first problems that were proved W[1]-complete, which means that it unlikely that there is a uniformly polynomial-time algorithm for this problem. In fact, even WEIGHTED 2SAT is W[1]-complete, showing that parameterized satisfiability problems and their classical counterparts can have different hardness.

The main result of the paper is a parameterized complexity analog of Schaefer's Dichotomy Theorem. For every constraint family $\mathscr{F}$, we determine the parameterized complexity of the WEIGHTED $\mathscr{F}$-SAT problem. In WEIGHTED $\mathscr{F}$-SAT we are given a formula with constraints from $\mathscr{F}$, and it has to be decided whether the formula has a satisfying assignment with exactly k true variables. We prove that WEIGHTED $\mathscr{F}$-SAT is either in FPT or W[1]-complete for every constraint family $\mathscr{F}$. The precise statement can be found in Theorem 3.2. Moreover, as in Schaefer's theorem, the class of FPT constraints has a simple characterization. We note here that in this theorem the class of "easy" constraint families does not even remotely resembles the class of polynomial-time solvable families in Schaefer's theorem. It seems that very different properties are required to make WEIGHTED $\mathscr{F}$-SAT easy.

The paper is organized as follows. In Section 2 we introduce a new property called weak separability. Section 3 states our main theorem (Theorem 3.2). Section 4 handles 0-invalid constraints. Section 5 gives an algorithm for bounded occurrence formulae. The positive results (uniformly polynomial-time algorithms) are presented in Section 6. In Section 7 we introduce a W[1]-complete problem, which is used in Section 8 to obtain further hardness results. Section 9 deals with the special case where the formula has bounded treewidth, while Section 10 considers the case of planar formulae.

2 Weakly separable constraints

A *boolean constraint* is a function f: $\{0,1\}^r \to \{0,1\}$, where r is called the *arity* of f. The r-tuple $s \in \{0,1\}^r$ *satisfies* f if $f(s) = 1$. There are exactly 2^{2^r} different constraints of arity r, hence if a constraint family $\mathscr{F}$ contains only constraints with arity at most r, then $|\mathscr{F}| \leq r2^{2^r}$. We will call the ith variable of a constraint f the ith *position* in f (the word "variable" will be reserved for the variables appearing in a formula).

An r-tuple $s \in \{0,1\}^r$ can be thought of as a subset of $\{1,2,\dots,r\}$: let i be in the subset if and only if the ith component of s is 1. Therefore we can apply standard set theoretic notations (such as union, disjointness, and symmetric difference) to the assignments of a constraint. Moreover, a constraint f can be expressed as a set system over $\{1,2,\dots,r\}$ that contains exactly those sets that correspond to satisfying assignments of the constraint.

We introduce a new property that (to the best of our knowledge) has not been investigated in the literature. It turns out that this property plays a crucial role in the parameterized complexity of WEIGHTED $\mathscr{F}$-SAT.

Definition 2.1 (Weak separability) *A constraint R is* weakly separable *if*

1. *whenever $\mathbf{x}_1$ and $\mathbf{x}_2$ are two satisfying assignments of R such that their intersection is satisfying, then their union is also satisfying, and*
2. *whenever $\mathbf{x}_1 \subset \mathbf{x}_2 \subset \mathbf{x}_3$ are satisfying assignments of R, then $(\mathbf{x}_2 \setminus \mathbf{x}_1) \cup \mathbf{x}_3$ $(= \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \mathbf{x}_3)$ is also satisfying.*

Here $\oplus$ means symmetric difference. In the rest of the section, we show some properties of weak separability, and present examples of weakly separable constraints.

A constraint is 0-valid (0-invalid) if it is satisfied (not satisfied) by the all-zero assignment. 1-valid and 1-invalid are defined similarly. In most of the paper we consider only 0-valid constraints. If R is 0-valid, then the requirements of Definition 2.1 can be made somewhat simpler:

Lemma 2.2 *A 0-valid constraint R is weakly separable if and only*

1. *whenever $\mathbf{x}_1$ and $\mathbf{x}_2$ are two disjoint satisfying assignments of R, then their union is also satisfying, and*
2. *whenever $\mathbf{x}_1$ and $\mathbf{x}_2$ are satisfying assignments of R such that $\mathbf{x}_1$ is a proper subset of $\mathbf{x}_2$, then their difference is also satisfying.*

Proof The necessity of these two requirements follow directly from Definition 2.1, since the all-zero assignment satisfies R.

Now assume that these two requirements hold. To see that the first requirement of Definition 2.1 holds for R, assume that $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_1 \cap \mathbf{x}_2$ satisfy R. If $\mathbf{x}_1 \subseteq \mathbf{x}_2$ or $\mathbf{x}_2 \subseteq \mathbf{x}_1$, then there is nothing to prove. Otherwise $\mathbf{x}_1 \setminus (\mathbf{x}_1 \cap \mathbf{x}_2) = \mathbf{x}_1 \setminus \mathbf{x}_2$ is a satisfying assignment by the second requirement of the lemma being proved. Assignments $\mathbf{x}_1 \setminus \mathbf{x}_2$ and $\mathbf{x}_2$ are disjoint, hence their union $\mathbf{x}_1 \cup \mathbf{x}_2$ is also satisfying by the first requirement.

To see that the second requirement of Definition 2.1 holds, let $\mathbf{x}_1 \subset \mathbf{x}_2 \subset \mathbf{x}_3$ be satisfying assignments. Now $\mathbf{x}_3 \setminus \mathbf{x}_2$ is also satisfying, and since it is disjoint from $\mathbf{x}_1$, it follows that $(\mathbf{x}_1 \setminus \mathbf{x}_2) \cup \mathbf{x}_3$ is satisfying, as required. $\square$

Another way of stating Lemma 2.2 is the following. If we consider two satisfying assignments as 0-1 vectors in $\mathbb{Z}^r$, and their sum (in $\mathbb{Z}^r$) is also a 0-1 vector, then the first property says that the sum is also satisfying. The second property says that the difference of two satisfying vectors is also satisfying if it is a 0-1 vector. Therefore Lemma 2.2 says that whenever the sum (difference) of the satisfying assignments is also a 0-1 vector, then the sum (difference) is also satisfying.

Definition 2.1 might seem to be a bit artificial, but as the following examples show, this class contains several interesting constraints.

Example 2.3 (Intersecting clutters) Consider the set system corresponding to the satisfying assignments of some constraint R. We say that the constraint is *intersecting* if every two non-empty sets in the system intersect each other. The constraint is a *clutter* if neither of the non-empty satisfying assignments is the proper subset of some other satisfying assignment.[1] If a 0-valid constraint R is an intersecting clutter, then it is weakly separable. Both requirements of Lemma 2.2 vacuously hold: there are no disjoint satisfying assignments and a satisfying assignment cannot be the subset of another satisfying assignment. For example, $R = \{00000, 11100, 00111, 01110\}$ is weakly separable. Moreover, for every r and $t > r/2$, the r-ary constraint that contains the all-zero assignment and all the assignments of weight exactly t is also weakly separable.

Example 2.4 (Affine constraints) A constraint of arity r is called *affine* if the subset of $\{0,1\}^r$ that corresponds to the satisfying assignments is an affine subspace of the r-dimensional space over GF[2]. It can be shown that a constraint is affine if and only if for every three satisfying assignments $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, the assignment $\mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \mathbf{x}_3$ also satisfies the constraint.

An affine constraint of arity r can be characterized by the equation $\mathbf{Ax} = \mathbf{b}$ over GF[2], where $\mathbf{A}$ is a matrix with r columns. If there are two satisfying assignments $\mathbf{x}_1$ and $\mathbf{x}_2$ such that their intersection $\mathbf{z}$ is also satisfying, then this means that $\mathbf{x}_1$, $\mathbf{x}_2$ can be written as $\mathbf{x}_1 = \mathbf{x}'_1 + \mathbf{z}$, $\mathbf{x}_2 = \mathbf{x}'_2 + \mathbf{z}$, where $\mathbf{x}'_1$ and $\mathbf{x}'_2$ are disjoint, and

$$\begin{aligned} \mathbf{Ax}_1 = \mathbf{A}(\mathbf{x}'_1 + \mathbf{z}) &= \mathbf{b}, \\ \mathbf{Ax}_2 = \mathbf{A}(\mathbf{x}'_2 + \mathbf{z}) &= \mathbf{b}, \\ \mathbf{Az} &= \mathbf{b}. \end{aligned}$$

Now the union of $\mathbf{x}_1$ and $\mathbf{x}_2$ is $\mathbf{x}'_1 + \mathbf{x}'_2 + \mathbf{z}$, which is also satisfying since

$$\begin{aligned} \mathbf{A}(\mathbf{x}'_1 + \mathbf{x}'_2 + \mathbf{z}) &= \mathbf{A}(\mathbf{x}'_1 + \mathbf{z}) + \mathbf{A}(\mathbf{x}'_1 + \mathbf{z}) - \mathbf{Az} \\ &= \mathbf{b} + \mathbf{b} - \mathbf{b} = \mathbf{b}. \end{aligned}$$

Moreover, if $\mathbf{x}_1 \subset \mathbf{x}_2 \subset \mathbf{x}_3$ are three satisfying assignments, then by a similar argument it can be shown that $\mathbf{x}_3 - \mathbf{x}_2 + \mathbf{x}_1$ is also a satisfying assignment. Thus we have shown that every affine constraint is weakly separable. In particular, the r-ary constraint EVEN$_r$ that requires that an even number of its variables are set to 1 is also weakly separable.

Example 2.5 (Integer lattices) An *integer lattice* L is a subset of $\mathbb{Z}^r$ that is generated by the integer linear combination of a finite number of vectors $\mathbf{a}_1, \dots, \mathbf{a}_k \in \mathbb{Z}^r$, that is, $L = \{\alpha_1\mathbf{a}_1 + \cdots + \alpha_k\mathbf{a}_k : \alpha_1, \dots, \alpha_k \in \mathbb{Z}\}$. An alternative definition is that L is an integer lattice if and only if for every two vectors in L their sum and their difference are also in L. This immediately

[1]Note that we use the notions intersecting and clutter in a slightly non-standard way. Here the empty set is allowed to be a member of a clutter or an intersecting set system.

implies that if we consider only the 0-1 vectors in L (the intersection of L with the hypercube $\{0,1\}^r$), then this yields a weakly separable constraint. Indeed, the sum and difference of every two satisfying assignments are in L, and if they happen to be 0-1 vectors, then they are also satisfying assignments.

The converse is not true: not every weakly separable constraint arises from an integer lattice this way. For example, consider the constraint R given in Example 2.3. If R is part of an integer lattice, then $11100 + 00111 - 01110 = 10101$ has to be in the lattice as well.

If $R(x_1, \dots, x_r)$ is a constraint of arity r, then for every $1 \le i \le r$ we define $R|_{(i,0)}(x_1, \dots, x_{r-1}) = R(x_1, \dots, x_{i-1}, 0, x_i, \dots, x_{r-1})$ to be a constraint of arity $r-1$. That is, $R|_{(i,0)}$ is obtained from R by restricting the ith position to 0. The constraint $R|_{(i,1)}$ is defined similarly. Applying these two operations repeatedly on R we can obtain 3^r (not necessarily distinct) constraints: each position can be forced to 0, forced to 1, or left unchanged. These constraints will be called the *restrictions* of R. Given a constraint family $\mathscr{F}$, we denote by $\mathscr{F}^*$ the set of those constraints that can be obtained from a member of $\mathscr{F}$ by repeated applications of these two operations. Clearly, if every constraint in $\mathscr{F}$ has arity at most r, then $|\mathscr{F}^*| \le 3^r |\mathscr{F}|$.

Weak separability is a hereditary property with respect to taking restrictions:

Lemma 2.6 *If R is weakly separable, then every restriction of R is also weakly separable.*

Proof Assume that R has a non-weakly separable restriction R'. Without loss of generality, it can be assumed that $R'(x_1, \dots, x_{r'}) = R(x_1, \dots, x_{r'}, \overbrace{0, \dots, 0}^{r_1}, \overbrace{1, \dots, 1}^{r_2})$. Abusing notations, if $\mathbf{x}$ is an r'-ary assignment of R', then we also consider $\mathbf{x}$ to be an r-ary assignment of R that assigns 0 to the last $r_1 + r_2$ positions. Let $\mathbf{z}$ be the r-ary assignment that assigns 1 to the last r_2 positions. An assignment $\mathbf{x}$ satisfies R' if and only if $\mathbf{x} \cup \mathbf{z}$ satisfies R.

If R' violates the first requirement of Definition 2.1, then there are assignments $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_1 \cap \mathbf{x}_2$ that satisfy R', but $\mathbf{x}_1 \cup \mathbf{x}_2$ is not satisfying. Therefore $\mathbf{x}_1 \cup \mathbf{z}$, $\mathbf{x}_2 \cup \mathbf{z}$, and their intersection $(\mathbf{x}_1 \cap \mathbf{x}_2) \cup \mathbf{z}$ satisfy R. Since R is weakly separable, thus $(\mathbf{x}_1 \cup \mathbf{z}) \cup (\mathbf{x}_2 \cup \mathbf{z}) = (\mathbf{x}_1 \cup \mathbf{x}_2) \cup \mathbf{z}$ also satisfies R, showing that $\mathbf{x}_1 \cup \mathbf{x}_2$ satisfies R', a contradiction. The case when R' violates the second requirement can be handled similarly. $\square$

Later we will need the following observation:

Lemma 2.7 *If R is a 0-invalid non-weakly separable constraint, then R has a 0-valid non-weakly separable restriction.*

Proof If R violates the first requirement of Definition 2.1, then there are assignments $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_1 \cap \mathbf{x}_2$ that satisfy R, but $\mathbf{x}_1 \cup \mathbf{x}_2$ is not satisfying. Consider the restriction R' of R where the positions that receive 1 in $\mathbf{x}_1 \cap \mathbf{x}_2$ are forced to 1. Clearly, R' is 0-valid, and based on $\mathbf{x}_1$ and $\mathbf{x}_2$ we can get two disjoint satisfying assignment whose union is not satisfying. If R violates the second requirement, then we force those positions to 1 that receive 1 in $\mathbf{x}_1$. Based on $\mathbf{x}_2$ and $\mathbf{x}_3$, we obtain two satisfying assignments such that one is the subset of the other, but their difference is not satisfying. $\square$

3 Weighted SAT

A *clause representing the constraint* f is a pair $\langle f, (x_1, \dots, x_r)\rangle$, where r is the arity of f and $x_1, \dots, x_r$ are variables. A 0-1 assignment of the variables satisfies this clause if $f(x_1, \dots, x_r) = 1$. If $\mathscr{F}$ is a finite family of constraints, then an *$\mathscr{F}$-formula* ϕ is a conjunction of clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each clause C_i represents some constraint f from $\mathscr{F}$. A variable assignment satisfies ϕ if it satisfies every clause of ϕ. A formula is *satisfiable* if it has at least one satisfying assignment. The *weight* of an assignment is the number of variables that are set to 1. Usually we denote by n the number of variables in the formula, and by m the number of clauses.

When defining constraint satisfaction problems some authors allow that a variable appears multiple times in a clause, while some others forbid this. In particular, Schaefer's original paper [15] allowed multiple variables, while Khanna et al. [12] does not. Disallowing multiple variables makes the constraint satisfaction problem less general, hence it makes obtaining hardness results more difficult. We present our results in the strongest possible form: we allow multiple variables when giving positive results, while on the negative side hardness is proved for the case when multiple variables are not allowed.

Formally, we will investigate the parameterized complexity of the following problem:

WEIGHTED $\mathscr{F}$-SAT

Input: An $\mathscr{F}$-formula ϕ (each variable can appear at most once in a clause) and an integer k.

Parameter: k

Question: Is there an assignment of weight exactly k that satisfies ϕ?

It can be shown that the problem WEIGHTED $\mathscr{F}$-SAT is in W[1] for every family $\mathscr{F}$.

In the rest of be paper we consider only parameterized problems, hence we will say $\mathscr{F}$-SAT instead of WEIGHTED $\mathscr{F}$-SAT for brevity. $\mathscr{F}$-SAT* denotes the more general problem where a variable can appear multiple times in a clause. If $\mathscr{F}$ contains only a single constraint R, then we abuse notation by writing R-SAT instead of $\{R\}$-SAT.

In some cases we allow that not only variables, but also the constants 0 and 1 can appear in the formula. This extension of the problem will be called

$\mathscr{F}$-SAT$_{01}$. In the problem $\mathscr{F}$-SAT$_0$ only the constant 0 is allowed. Problems $\mathscr{F}$-SAT$^*_{01}$ and $\mathscr{F}$-SAT$^*_0$ are defined similarly.

It is easy to see that the problem $\mathscr{F}$-SAT$_{01}$ is essentially the same as $\mathscr{F}^*$-SAT (recall that $\mathscr{F}^*$ contains all the restrictions of $\mathscr{F}$). If a clause of the formula contains constants, then the clause can be replaced by an appropriate constraint from $\mathscr{F}^*$, and vice versa. Therefore we obtain

Proposition 3.1 *For every constraint family $\mathscr{F}$, the problems $\mathscr{F}$-SAT$_{01}$ and $\mathscr{F}^*$-SAT have the same complexity.* □

Although the definition is somewhat technical, weak separability is precisely the property that separates the easy and the hard cases in the $\mathscr{F}$-SAT problem:

Theorem 3.2 (Main) *Let $\mathscr{F}$ be a finite set of constraints. If every constraint in $\mathscr{F}$ is weakly separable, then $\mathscr{F}$-SAT is in* FPT *otherwise $\mathscr{F}$-SAT is* W[1]-*complete.*

We prove Theorem 3.2 the following way. The special case when the formula is not satisfied by the all-zero assignment can be taken care of easily (Lemma 4.1). The next step is to prove that the problem is in FPT for *every $\mathscr{F}$* if the formula is bounded occurrence, that is, if every variable occurs at most d (constant) times. Theorem 5.3 gives a uniformly polynomial-time algorithm for the bounded occurrence case. The algorithm first collects a set of solutions that are “local” in some sense, then uses color coding to put together these assignments to obtain a solution of exactly the required weight.

If a variable occurs many times in the formula and every member of $\mathscr{F}$ is weakly separable, then we can use the sunflower lemma of Erdős and Rado to find a certain special structure in the formula. This structure allows us to reduce the problem to a shorter but equivalent form (Theorem 6.5). Repeating these reductions, eventually we arrive to a formula where each variable occurs a bounded number of times, proving the positive side of Theorem 3.2.

On the negative side, we use two hardness results as basis to our reductions. First, the parameterized maximum independent set problem is well-known to be W[1]-complete. Notice that the maximum independent set problem is in fact the same as $\mathscr{F}$-SAT with $\mathscr{F} = \{(\bar{x} \vee \bar{y})\}$: the constraint $(\bar{x} \vee \bar{y})$ (that is, NAND) expresses the requirement that either x or y should not be selected into the independent set. Moreover, we prove in Lemma 7.1 that the constraint $(x \rightarrow y)$ also makes weighted satisfiability W[1]-complete. It turns out that if a constraint is not weakly separable, then it can simulate one of $(\bar{x} \vee \bar{y})$ and $(x \rightarrow y)$, making the satisfiability problem W[1]-hard (Lemma 8.1). This proves the negative side of Theorem 3.2.

Besides bounding the number of occurrences, we investigate the effect of other structural restrictions on the formula. The incidence graph of a formula is a bipartite graph having the variables and clauses as vertices, where the edges represent the incidence relation. We prove that $\mathscr{F}$-SAT is in FPT for every $\mathscr{F}$ if the incidence graph of the formula has bounded treewidth (Theorem 9.4) or it is planar (Theorem 10.2). These results follow from standard algorithmic techniques of bounded treewidth graphs.

4 0-invalid constraints

The case when the formula contains 0-invalid constraints can be taken care of easily: the problem can be reduced to a constant number of 0-valid formulae.

Lemma 4.1 *Let $\mathscr{F}$ be a family of constraints with arity at most r. The $\mathscr{F}$-SAT problem can be reduced to at most r^k instances of the $\mathscr{F}^*$-SAT (or $\mathscr{F}$-SAT$_{01}$) problem such that the constructed instances contain only 0-valid constraints. Moreover, the reduction does not increase the number of occurrences for any of the variables and the parameter k' for the generated $\mathscr{F}^*$-SAT instances is not greater than the parameter k.*

Proof We use the method of bounded search trees. If the formula ϕ contains a 0-invalid clause C_i, then one of the variables in C_i has to be 1. Therefore the algorithm selects a variable in C_i and sets it to 1. Since there are at most r variables in C_i, thus we branch into at most r directions. Now there are constants in the formula, but we can get rid of these constants by replacing the clauses containing constants with appropriate constraints from $\mathscr{F}^*$ (Prop. 3.1). We repeat this procedure until there are no 0-invalid clauses. If we set k variables to 1 and there are still 0-invalid clauses, then this branch of the algorithm is unsuccessful and we stop. If the formula becomes 0-valid after setting c variables to 1, then we check whether it has a satisfying assignment of weight $k' := k - c$. If there is such an assignment, then it gives a satisfying assignment of weight k for the original formula. The search tree of the algorithm has height at most k, hence it has at most r^k leaves, implying that we generate at most r^k 0-valid formulae to check. $\square$

5 Bounded occurrences

In this section we give a uniformly polynomial-time algorithm for $\mathscr{F}$-SAT in the special case when every variable appears in a bounded number of clauses. The main idea is that we can generate a linear number of satisfying assignments such that every satisfying assignment of weight at most k can be obtained as the disjoint union of some these assignments. Now an algorithm based on color coding can be used to decide whether a satisfying assignment of weight exactly k can be put together from these selected assignments.

The vertex set of the *primal graph* $G(\phi)$ of formula ϕ is the set of variables in ϕ, and two variables are connected by an edge if they appear in a common clause. We say that a set of variables is *connected* in ϕ if they induce a connected subgraph of $G(\phi)$. A set of variables is *satisfying* in ϕ if setting these variables to 1 and all the other variables to 0 gives a satisfying assignment. The following lemma bounds the number of connected satisfying sets:

Lemma 5.1 *Let r be the maximum arity of the clauses in the 0-valid formula ϕ, and assume that every variable occurs at most d times in ϕ. There are at most $(rd)^{k^2} \cdot n$ connected satisfying sets of variables having size at most k. Moreover, we can enumerate all such sets in $2^{O(k^2 \log rd)} \cdot n$ time.*

Proof In $G(\phi)$ every vertex has degree at most $(r-1)d$. We give an upper bound on the number of connected subsets that contain variable x_i and have size at most k. If variable x_i and at most $k-1$ other vertices form a connected subgraph, then all these vertices are at distance at most $k-1$ from x_i. There are less than $((r-1)d)^k < (rd)^k$ vertices at distance less than k from x_i, therefore we have to consider only these vertices. One can form less than $(rd)^{k^2}$ different sets of size at most k from these vertices, this bounds the number of sets containing x_i. Considering all the n variables, we obtain the upper bound $(rd)^{k^2} \cdot n$.

It is not difficult to show that we can generate all these sets in time polynomial in d, r, and k per set (with appropriate data structures). Therefore the total time can be bounded by $2^{O(k^2 \log rd)}$. Moreover, selecting the satisfying sets can be also done within this time bound: for each set, we have to check at most kd clauses (those clauses that do not contain selected variables are automatically satisfied since the formula is 0-valid). $\square$

Two sets of variables V' and V'' are *nonadjacent* if there is no clause that contains variables from both V' and V''. The union of pairwise nonadjacent satisfying sets is also satisfying:

Lemma 5.2 *If V_1, V_2, ..., V_ℓ are pairwise nonadjacent satisfying sets of variables for the 0-valid formula ϕ, then $V_1 \cup \cdots \cup V_\ell$ also satisfies ϕ.*

Proof Assume that clause C_j is not satisfied by $V_1 \cup \cdots \cup V_\ell$. Since ϕ is 0-valid, hence C_j must contain one or more variables set to 1, denote these variables by V'. Since the sets V_1, V_2, ..., V_ℓ are pairwise nonadjacent, thus V' is contained in one of these sets, say V_i. Therefore C_j receives the same assignment as in V_i, contradicting the assumption that V_i is satisfying. $\square$

Now we are ready to present the algorithm for bounded occurrence formulae:

Theorem 5.3 *Let r be the maximum arity of the clauses in a formula ϕ, and assume that every variable occurs at most d times in ϕ. It can be decided in $2^{O(k^2 d \log r)} \cdot n \log n$ time whether ϕ has a satisfying assignment of weight k.*

Proof If the formula is not 0-valid, then Lemma 4.1 can be used to reduce the problem to at most r^k 0-valid instances. Therefore in the following we assume that the formula is 0-valid. For 0-invalid formulae, the running time obtained below has to be multiplied by r^k, which is dominated by the exponent.

Every satisfying assignment can be partitioned into pairwise nonadjacent connected satisfying assignments by taking its connected components in the underlying graph. Conversely, if we have pairwise nonadjacent connected satisfying assignments, then by Lemma 5.2, their union is also a satisfying assignment.

Therefore ϕ has a satisfying assignment of weight k if and only if there are pairwise nonadjacent connected satisfying assignments whose total size is k. Our algorithm tries to find such sets.

By Lemma 5.1, we can enumerate all the connected satisfying sets of size at most k, call these sets $V_1, \ldots, V_t$. For each such set V_i there corresponds a set of clauses $C[V_i]$ where the variables of V_i appear. Consider these sets $C[V_1]$, $C[V_2], \ldots, C[V_t]$, and associate a *weight* to each set. Let the weight of $C[V_i]$ be $|V_i|$, clearly the size of $C[V_i]$ is at most d times its weight. Notice that V_i and V_j are non-adjacent if and only if the corresponding sets $C[V_i]$ and $C[V_j]$ are disjoint. Therefore the observation of the previous paragraph can be restated as follows: ϕ has a satisfying assignment of weight k if and only if there are pairwise disjoint sets $C[V_{i_1}], \ldots, C[V_{i_\ell}]$ whose total weight is k. We use the method of color coding to decide whether such sets exist.

First we present the randomized version of the algorithm. Select a random coloring of the clauses using a set C of $c := kd$ colors. The algorithm uses dynamic programming to find a solution where the clauses covered by the sets $C[V_{i_1}], \ldots, C[V_{i_\ell}]$ have distinct colors. For every subset $C' \subseteq C$ of colors, every $0 \le i \le t$ and $0 \le k' \le k$ we set subproblem $S[C', i, k']$ to true if one can select pairwise disjoint sets from $C[V_1], \ldots, C[V_i]$ such that their total weight is k', the clauses covered by them have distinct colors, and they cover only clauses with color from C'. We are interested in $S[C, t, k]$, if it is true, then there is a weight k satisfying assignment.

It is trivial to solve the subproblems for $i = 0$. We can move from i to $i + 1$ as follows. If $S[C', i, k']$ is true, then $S[C', i+1, k']$ is also true, since any solution for i can be used for $i + 1$ as well. Moreover, let C_i be the set of colors appearing on the clauses of $C[V_i]$ (we assume that these colors are distinct, otherwise $C[V_i]$ cannot appear in a solution with this coloring). If $S[C' \setminus C_i, i, k' - |V_i|]$ is true, then we can set $S[C', i+1, k']$ to true as well: a solution to $S[C' \setminus C_i, i, k' - |V_i|]$ can be extended by the weight $|V_i|$ set $C[V_i]$ to obtain a solution that covers clauses only with color C'. Using these two rules, we can solve all the subproblems.

If there are pairwise disjoint sets $C[V_{i_1}], \ldots, C[V_{i_\ell}]$ whose total weight is k, then they cover at most $c = kd$ clauses (recall that the size of $C[V_i]$ is at most d times its weight). Therefore with probability at least $c!/c^c$, the clauses covered by $C[V_{i_1}], \ldots, C[V_{i_\ell}]$ have distinct colors, and the algorithm finds a solution. This means that if there is a weight k satisfying assignment, then on average we have to choose at most $c^c/c!$ random colorings to find a solution. We can derandomize the algorithm by using the standard technique of k-perfect hash functions [2, 8]. If there are m elements, then one can construct a family of $2^{O(c)} \log m$ c-colorings such that for each c-element subset X of the elements there is a coloring in the family where each element in X receives a different color. It is clear that the algorithm will work correctly if we modify it such that instead of repeatedly choosing random colorings we enumerate all the colorings in the family: eventually we select a coloring where all the at most c clauses covered by the solution are colored differently. Thus the algorithm considers $2^{O(c)} \log m \le 2^{O(c)} d \log n$ colorings. For each coloring, the dynamic programming algorithm solves at most $2^c kt \le 2^c k(rd)^{k^2} \cdot n$ subproblems. Each subproblem requires time polynomial in r, d, and k. Therefore the total running time is $2^{O(k^2 d \log r)} \cdot n \log n$. □

6 Fixed-parameter tractable cases

In this section we prove the positive part of Theorem 3.2: we show that if every constraint is weakly separable, then $\mathscr{F}$-SAT is in FPT. In fact, we show that even the more general problem $\mathscr{F}$-SAT$^*_{01}$ is fixed-parameter tractable. By Lemma 4.1, the 0-invalid clauses can be easily taken care of, therefore we assume that the formula is 0-valid. If every variable occurs at most d times (where d is a constant to be defined later), then the algorithm of Theorem 5.3 can be used. On the other hand, if a variable occurs more than d times, then we can find a large *sunflower* of weakly separable clauses, which allows us to simplify the formula.

The sunflower was defined in the context of set systems:

Definition 6.1 (Sunflower) *A* sunflower with p petals *is a collection of p sets $S_1, \ldots, S_p$ such that the intersection $S_i \cap S_j$ is the same for every $i \neq j$.*

In particular, p pairwise disjoint sets form a sunflower with p petals. The intersection of the sets will be called the *center* of the sunflower. The following lemma states that a sufficiently large set system necessarily contains a sunflower of given size:

Lemma 6.2 (Erdős and Rado, 1960, [9]) *If a set system has more than $(p-1)^\ell \ell!$ members and the size of each member is at most ℓ, then the set system contains a sunflower with p petals.*

We will use the notion of sunflower for clauses instead of sets. For clauses, we define the sunflower the following way:

Definition 6.3 (Sunflower) *A* sunflower with p petals *is a collection of p clauses $C_1, \ldots, C_p$ such that every clause represents the same constraint R of arity r, and for every $i = 1, \ldots, p$ and $j = 1, \ldots, r$*

- *either the same variable appears at the jth position of every clause, or*
- *the variable at the jth position of clause C_i appears only in C_i.*

For example, the clauses $R(x_1, x_2, x_3, x_4)$, $R(x_1, x_2, x_5, x_5)$, $R(x_1, x_2, x_6, x_7)$ form a sunflower with 3 petals. Here variables x_1 and x_2 form the center. It turns out that if a variable appears in many clauses, then there is a large sunflower in the formula:

Lemma 6.4 *Let $\mathscr{F}$ be a family of constraints with maximum arity r containing c constraints. If a variable x_i appears in more than $(r^r k)^r \cdot r! \cdot r^r \cdot c$ clauses* *of an $\mathscr{F}$-formula ϕ, then ϕ contains a sunflower with non-empty center and at least $k+1$ petals.*

Proof Among the clauses that contain variable x_i, at least $(r^r k)^r \cdot r! \cdot r^r$ of them have to represent the same constraint $R \in \mathscr{F}$. For each such clause, consider the set of variables contained in the clause. This way we obtain a family of $(r^r k)^r \cdot r! \cdot r^r$ sets, but a set can appear multiple times in the family. As a very rough estimate, we can say that there can be at most r^r different clauses on the same set of at most r variables (taking into account that a variable can appear multiple times in a clause), therefore if we retain only one copy of each set, then there remains at least $(r^r k)^r \cdot r!$ sets. Therefore by Lemma 6.2, this collection of sets contains a sunflower with $r^r k + 1$ petals. The center C of the sunflower is not empty, since it contains variable x_i. The clauses corresponding to the sets in the sunflower all use the variables in C, but these variables may appear in these clauses at different positions. We say that two clauses use the center C the same way if whenever the variable at the jth position of one clause is a variable in C, then the same variable appears in the other clause at the jth position. It is clear that there are at most r^r (rough upper bound) different ways of using C, thus there have to be more than k sets in the sunflower such that the corresponding clauses use the center C the same way. These clauses form a sunflower of size at least $k+1$: if the variable at the jth position of a clause is in C, then it appears in all the clauses at the jth position; if it is not in C, then it appears only in that clause. $\square$

The key idea of the algorithm for weakly separable constraints is to find a sunflower and reduce the formula by "plucking" the petals of the sunflower.

Theorem 6.5 *If every constraint in $\mathscr{F}$ is weakly separable, then $\mathscr{F}$-SAT$^*_{01}$ is fixed-parameter tractable.*

Proof By Prop. 3.1, $\mathscr{F}$-SAT$^*_{01}$ and $\mathscr{F}^*$-SAT* are equivalent, we give an algorithm for the latter problem. Note that by Lemma 2.6, every constraint in $\mathscr{F}^*$ is weakly separable. If the given $\mathscr{F}^*$-formula ϕ is not 0-valid, then we use Lemma 4.1 to reduce the problem to at most r^k 0-valid instances of $\mathscr{F}^*$-SAT*. Therefore in the following we can assume that the formula is 0-valid and every constraint is weakly separable.

Let r be the maximum arity of the constraints in $\mathscr{F}$, and set $c := |\mathscr{F}^*| \le 3^r|\mathscr{F}| \le 3^r \cdot 2^{2^r} r$ and $d := r \cdot (r^r k)^r \cdot r! \cdot r^r \cdot c$. If every variable occurs at most d times in the 0-valid formula ϕ, then Lemma 5.3 can be used to solve the problem in $2^{O(k^2 d \log r)} \cdot n \log n = 2^{k^{r+2} \cdot 2^{2^{O(r)}}} \cdot n \log n$ time. Otherwise there is a variable that occurs more than d times. This means that this variable appears in at least d/r clauses, hence the formula contains a sunflower with $k+1$ petals (Lemma 6.4). Let $C_1, \ldots, C_{k+1}$ be the clauses of the sunflower and let C be its center. The clauses of the sunflower represent the same constraint R of arity $r' \le r$, it can be assumed without loss of generality that in each of these clauses, the first $\ell \ge 1$ variables are taken from C, and the remaining $r' - \ell$ variables are outside C.

We reduce the problem to a shorter formula by "plucking" the sunflower. In each clause $C_1, \ldots, C_{k+1}$ the variables of the center C are replaced by the constant 0, call C'_i these modified clauses. Furthermore, a new clause C'_0 is added to the formula: C'_0 can be obtained from any of the clauses C_i ($i = 1, \ldots, k+1$) by replacing the variables *not* in C by the constant 0. (Observe that by the definition of the sunflower, this gives the same clause C'_0 starting from any C_i). For example, plucking the sunflower

$$\begin{aligned} C_1 &= R(x_1, x_2, x_3, x_4), \\ C_2 &= R(x_1, x_2, x_5, x_5), \\ C_3 &= R(x_1, x_2, x_6, x_7) \end{aligned}$$

gives

$$\begin{aligned} C'_0 &= R(x_1, x_2, 0, 0), \\ C'_1 &= R(0, 0, x_3, x_4), \\ C'_2 &= R(0, 0, x_5, x_5), \\ C'_3 &= R(0, 0, x_6, x_7). \end{aligned}$$

We claim that this operation does not change the solvability of the instance with respect to weight k solutions.

Assume that the new formula ϕ' has a satisfying assignment $\mathbf{x}$ of weight k, but this assignment does not satisfy ϕ. This is only possible if one of the clauses C_i ($i = 1, \ldots, k+1$) is not satisfied, since all the other clauses of ϕ are present in ϕ' as well. Assume that clause C_i is not satisfied, thus $\mathbf{x}$ and C_i gives an r'-tuple $(\alpha_1, \ldots, \alpha_{r'})$ that does not satisfy the constraint R. However, $\mathbf{x}$ satisfies C'_i, hence $(0, \ldots, 0, \alpha_{\ell+1}, \ldots, \alpha_{r'})$ does satisfy R. Moreover, $\mathbf{x}$ satisfies C'_0, hence $(\alpha_1, \ldots, \alpha_\ell, 0, \ldots, 0)$ also satisfies R. Therefore we have two disjoint assignments satisfying R and since constraint R is 0-valid and weakly separable, the union of the assignments $(\alpha_1, \ldots, \alpha_\ell, \alpha_{\ell+1}, \ldots, \alpha_{r'})$ also satisfies R (Lemma 2.2), a contradiction.

Now assume that ϕ has a satisfying assignment $\mathbf{x}$ of weight k that does not satisfy ϕ'. There are at most k true variables outside C and by the definition of the sunflower, each such variable appears in at most one of the clauses $C_1, \ldots, C_{k+1}$. Thus there has to be a clause C_i that does not contain true variables outside C. Therefore the r'-tuple $(\alpha_1, \ldots, \alpha_\ell, 0, \ldots, 0)$ assigned by $\mathbf{x}$ to C_i satisfies the constraint R. This means that the clause C'_0 is satisfied in ϕ'. Assume therefore that for some clause C'_j ($1 \leq j \leq k+1$) the r'-tuple $(0, \ldots, 0, \alpha_{\ell+1}, \ldots, \alpha_{r'})$ assigned to C'_j does not satisfy R. However, $\mathbf{x}$ assigns the r'-tuple $(\alpha_1, \ldots, \alpha_\ell, \alpha_{\ell+1}, \ldots, \alpha_{r'})$ to C_j (observe that C_i and C_j use the variables of the center the same way), thus this r'-tuple satisfies R. Now from the weak separability of R (see also Lemma 2.2) and from the facts that $(\alpha_1, \ldots, \alpha_\ell, 0, \ldots, 0)$ and $(\alpha_1, \ldots, \alpha_\ell, \alpha_{\ell+1}, \ldots, \alpha_{r'})$ satisfy R it follows that the difference $(0, \ldots, 0, \alpha_{\ell+1}, \ldots, \alpha_{r'})$ also satisfies R, a contradiction.

Thus the formula ϕ' is equivalent to the original formula ϕ if we are only interested in weight k solutions. Formula ϕ' contains some constant zeros, but we can get rid of the constants by replacing the affected constraints with appropriate constraints from $\mathscr{F}^*$ (Prop. 3.1). Notice that plucking the sunflower strictly decreases the total number of occurrences of the variables. Therefore by repeating this operation at most as many times as the number of literals in the original formula ($\leq mr$), eventually we obtain a formula where every variable occurs at most d times. As noted above, in this case Lemma 5.3 can be used to solve the problem in uniformly polynomial time. $\square$

7 Hardness of implication

The negative part of Theorem 3.2 requires us to prove the W[1]-completeness of certain problems. All our completeness proofs are done by reduction from two problems, maximum independent set and IMPLICATIONS, where IMPLICATIONS is $\mathscr{F}$-SAT for $\mathscr{F} = \{(x \to y)\}$. Maximum independent set (which can be also thought of as $\mathscr{F}$-SAT for $\mathscr{F} = \{(\bar{x} \vee \bar{y})\}$) is a well-known W[1]-complete problem [8]. In this section we show that it is W[1]-complete to find a satisfying assignment of weight exactly k for a formula containing only implications of the form $(x \to y)$.

Notice that if $\mathscr{F} = \{(\bar{x} \vee \bar{y})\}$, then $\mathscr{F}$-SAT remains W[1]-hard even if we look for satisfying assignments of weight *at least* k instead of *exactly* k. On the other hand, the constraint $(x \to y)$ is 1-valid, thus it is trivial to find a satisfying assignment of weight at least k. Therefore the following hardness result has to rely on the fact that the weight of the satisfying assignment to be found is exactly k.

Lemma 7.1 *IMPLICATIONS is* W[1]*-complete.*

Proof We prove that the weighted version of the problem is W[1]-complete. In the weighted version each variable x_i is given a positive integer weight $w(x_i)$, and one has to find a satisfying assignment where the sum of the weights of the true variables is exactly k. If the weights are of constant size, then the weighted problem can be reduced to the unweighted problem in uniformly polynomial time. For each variable x_i, we add $w(x_i) - 1$ new variables $x_{i,1}, \dots, x_{i,w(x_i)-1}$, and the clauses $x_i \to x_{i,1}, x_{i,1} \to x_{i,2}, \dots, x_{i,w(x_i)-1} \to x_i$. These clauses form a cycle of implications, hence either all or none of these variables are true in a satisfying assignment. Thus these variables effectively act as one variable with weight $w(x_i)$, completing the reduction.

In the following, we show that weighted IMPLICATIONS is W[1]-hard. The proof is by a parameterized reduction from the maximum independent set problem. Let $G(V, E)$ be a graph, and let k be the number of independent vertices to be found. Set $k' = k + \binom{k}{2}$. We construct a formula where the variables are partitioned into k' sets $X_1, \dots, X_{k'}$. Each variable in X_i has weight $w_i = 2^{i-1} + 2^{2k'-i}$. The required weight of the solution is $k'' = \sum_{i=1}^{k'} w_i = 2^{2k'} - 1$.

We claim that any assignment with weight k'' sets to 1 exactly one variable from each set X_i. Suppose that i is the smallest index such that the claim does not hold. There are two cases. If X_i does not contain a variable with value 1, then consider the weight of the assignment modulo 2^i. The weight $w_{i'}$ is $2^{i'-1}$ modulo 2^i for $i' < i$, and it is 0 modulo 2^i for $i' > i$. By assumption, there is exactly one true variable in each X_i for $i' < i$, hence the weight is $\sum_{i'=1}^{i-1} 2^{i'-1} = 2^{i-1} - 1$ modulo 2^i. However, k'' is $2^i - 1$ modulo 2^i, a contradiction. Now assume that X_i contains at least two true variables. In this case the weight of the assignment is at least $\sum_{i'=1}^{i-1} w_{i'} + 2w_i \geq \sum_{i'=1}^{i-1} 2^{2k'-i'} + 2 \cdot 2^{2k'-i} > 2^{2k'} - 1 = k''$, again a contradiction.

In the following, we will rename the $k' = k + \binom{k}{2}$ sets X_i as Y_i for $1 \leq i \leq k$ and $Y_{i,j}$ for $1 \leq i < j \leq k$. Each set Y_i contains $|V|$ variables: there is a variable $y_{i,v}$ for each $v \in V$. Each $Y_{i,j}$ contains $\binom{|V|}{2} - |E|$ variables, that is, there is a variable $y_{i,j,u,v}$ for each non-edge $uv \notin E$ of the graph. Clauses are defined as follows: for every $1 \leq i < j \leq k$ and every non-edge $uv \notin E$, we add the two clauses $(y_{i,j,u,v} \to y_{i,u})$ and $(y_{i,j,u,v} \to y_{j,v})$.

Assume that there is a solution of weight exactly k''. We have seen that in such a solution, each set Y_i and $Y_{i,j}$ contains exactly one true variable. We construct an independent set of size k based on this solution: if variable $y_{i,v}$ is true, then let v be the ith vertex of the independent set. We claim that this results in k distinct independent vertices. To see that the ith and the jth vertex are not the same and not connected by an edge, assume that $y_{i,j,u,v}$ is the unique true variable in $Y_{i,j}$. The clauses imply that variables $y_{i,u}$ and $y_{j,v}$ are true, hence the ith vertex is u, and the jth vertex is v. By construction, uv is a non-edge in G, hence u and v are distinct vertices not connected by an edge.

To see the other direction, assume that $v_1, \ldots, v_k$ is an independent set of size k. It is easy to see that setting to 1 the variables y_{i,v_i} $(1 \leq i \leq k)$ and y_{i,j,v_i,v_j} $(1 \leq i < j \leq k)$ yields a satisfying assignment of weight exactly k''. $\square$

8 Hardness results

In this section we prove the negative side of Theorem 3.2: if $\mathscr{F}$ contains a non-weakly separable constraint, then $\mathscr{F}$-SAT is W[1]-complete. The following lemma shows a weaker claim: it needs a slightly stronger assumption ($\mathscr{F}$ contains a 0-valid non-weakly separable constraint) and it proves hardness for the more general problem $\mathscr{F}$-SAT$^*_0$. The proof contains all the important ideas, it shows what role (the lack of) weak separability plays in the complexity of the problem. A couple of technical tricks are required to prove hardness for the more restricted problem $\mathscr{F}$-SAT (Lemma 8.2, 8.3, and 8.4).

Lemma 8.1 *Let $\mathscr{F}$ be a finite constraint family. If $\mathscr{F}$ contains a 0-valid constraint that is not weakly separable, then $\mathscr{F}$-SAT$^*_0$ is* W[1]*-complete.*

Proof Assume that $R \in \mathscr{F}$ is a 0-valid constraint of arity r that is not weakly separable. Since R is 0-valid, it violates one of the requirements of Lemma 2.2. We consider two cases depending on which requirement is violated.

If there are two disjoint satisfying assignments of R whose union does not satisfy R, then we reduce the maximum independent set problem to R-SAT$^*_0$ as follows. Without loss of generality, it can be assumed that $(\overbrace{1,\dots,1}^{\ell_1},0,\dots,0)$ and $(\overbrace{0,\dots,0}^{\ell_1},\overbrace{1,\dots,1}^{\ell_2},0,\dots,0)$ satisfy R but $(\overbrace{1,\dots,1}^{\ell_1},\overbrace{1,\dots,1}^{\ell_2},0,\dots,0)$ does not. Now a clause $(\bar{x}_i \vee \bar{x}_j)$ of the maximum independent set problem can be simulated as $R(\overbrace{x_i,\dots,x_i}^{\ell_1},\overbrace{x_j,\dots,x_j}^{\ell_2},0,\dots,0)$. It is clear that this clause forbids that both of x_i and x_j is true at the same time, but the clause is satisfied if at most one of them is true.

If R violates the second requirement of weak separability, then we reduce IMPLICATIONS to R-SAT$^*_0$. In Lemma 7.1 we have shown that IMPLICATIONS is W[1]-complete. Without loss of generality, it can be assumed that $(\overbrace{1,\dots,1}^{\ell_1},0,\dots,0)$ and $(\overbrace{1,\dots,1}^{\ell_1},\overbrace{1,\dots,1}^{\ell_2},0,\dots,0)$ satisfy R but the difference $(\overbrace{0,\dots,0}^{\ell_1},\overbrace{1,\dots,1}^{\ell_2},0,\dots,0)$ does not. In this case a clause $(x_i \rightarrow x_j)$ of the IMPLICATIONS problem can be replaced by the clause $R(\overbrace{x_j,\dots,x_j}^{\ell_1},\overbrace{x_i,\dots,x_i}^{\ell_2},0,\dots,0)$. Clearly, x_i cannot be true without x_j being true as well, but every other combination of values is allowed. $\square$

A constraint R is *monotone* if whenever an assignment $\mathbf{x}$ satisfies R, then replacing any 0 in $\mathbf{x}$ by a 1 also gives a satisfying assignment. The following lemma states that a 0-invalid non-monotone constraint allows us to simulate constants.

Lemma 8.2 *If constraint family $\mathscr{F}$ contains a 0-invalid non-monotone constraint R of arity r, then $\mathscr{F}$-SAT$_{01}$ can be reduced to $\mathscr{F}$-SAT.*

Proof Let $r_{\max}$ be the maximum arity in $\mathscr{F}$. Given an $\mathscr{F}$-formula ϕ and an integer k, we construct a constant-free $\mathscr{F}$-formula ϕ' such that ϕ has a satisfying assignment of weight k if and only if ϕ' has a satisfying assignment of weight $k' := k + r_{\max}$. We introduce $r_{\max}$ new variables $X = \{x_1,\dots,x_{r_{\max}}\}$, and $r_{\max}+k$ new variables $Y = \{y_1,\dots,y_{r_{\max}+k}\}$. With some new clauses we ensure that if a satisfying assignment of ϕ' has weight k', then it assigns 1 to all the variables x_1, ..., $x_{r_{\max}}$, and 0 to y_1, ..., $y_{r_{\max}+k}$. Therefore the constants in the formula can be replaced by these variables. This gives a correct reduction, since a weight k' satisfying assignment of ϕ' sets to 1 exactly k original variables.

First we add clauses to ensure that every variable in X is set to 1. The new clauses are added as follows. Consider a minimum weight satisfying assignment having weight $0 < \ell \le r$. Without loss of generality, it can be assumed that $(\overbrace{1,\dots,1}^{\ell},0,\dots,0)$ satisfies R. We add the clauses $R(x_{i_1},x_{i_2},\dots,x_{i_\ell},y_{j_1},y_{j_2},\dots,y_{j_{r-\ell}})$ where i_1, ..., i_ℓ are distinct integers between 1 and $r_{\max}$, and j_1, ..., $j_{r-\ell}$ are distinct integers between 1 and $r_{\max}+k$. Considering all possibilities, there are $(r_{\max}!/(r_{\max}-\ell)!)\cdot((r_{\max}+k)!/(r_{\max}+k-r+\ell)!)$ such clauses. We claim that these clauses ensure that the variables x_i are true in every weight k' satisfying assignment. Notice first that among the $r_{\max}+k$ variables y_j, at least $r_{\max}$ of them (say y_{j_1}, ..., $y_{j_{r_{\max}}}$) are 0 in a weight k' assignment. Assume that some variable x_{i_1} is 0, then the clause $R(x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}, y_{j_1}, \ldots, y_{j_{r-\ell}})$ (where x_{i_2}, ..., x_{i_ℓ} are arbitrary distinct variables different from x_{i_1}) has an assignment of weight less than ℓ. But R has no satisfying assignment with weight less than ℓ, thus this clause is not satisfied, a contradiction.

Constraint R is not monotone, hence there is a satisfying assignment α of weight $0 < \ell' < r$ such that setting the pth position to 1 (for some p) makes this assignment unsatisfying. We add new clauses to ϕ' based on assignment α: replace every 1 in α with a distinct variable from X, and replace every 0 with a distinct variable from Y. Selecting the variables in every possible way gives $(r_{\max}!/(r_{\max}-\ell')!)\cdot((r_{\max}+k)!/(r_{\max}+k-r+\ell')!)$ clauses. We have seen in the previous paragraph that in a satisfying assignment of weight k', each variable of X is 1, and at least r variables of Y are 0. Assume that a variable y_j has value 1. There has to be a clause where y_j appears at the pth position, but every other variable from Y in the clause has value 0. Thus this clause receives the assignment α, but with the pth position set to 1, which does not satisfy R. $\square$

We say that the pth position of a constraint is *useful* if there is a satisfying assignment that sets this position to 1. The pth position is *satisfying* if the weight 1 assignment that sets to 1 only the pth position is satisfying. We consider two cases depending on whether every useful position is satisfying or not. If every useful position is satisfying, then we give a direct proof of W[1]-completeness (Lemma 8.3). Otherwise we show that $\mathscr{F}$-SAT$_0^*$ can be reduced to $\mathscr{F}$-SAT (Lemma 8.4), that is, allowing variables occurring multiple times in a clause does not make the problem harder.

Lemma 8.3 *Let R be a 0-valid constraint of arity r such that every useful position is satisfying. If R is not weakly separable, then the R-SAT problem is* W[1]*-complete.*

Proof The first observation is that R violates the first requirement of weak separability in Lemma 2.2. Otherwise R would be satisfied by every assignment that has value 1 only at useful positions, since these assignments can be obtained as the disjoint union of weight 1 satisfying assignments. Therefore the second requirement of weak separability would be also satisfied, contradicting the assumption that R is not weakly separable. Consider the counterexample to the first requirement where the weight ℓ of the union of the two disjoint sets is minimal. Without loss of generality, it can be assumed that the first $\ell \geq 2$ positions are useful, $(\overbrace{1,\ldots,1}^{\ell},0,\ldots,0)$ does not satisfy R, but every subset of this assignment is satisfying.

We reduce the maximum independent set problem to R-SAT as follows. There is a variable x_v for each vertex v, and additionally there is a set Y of

$r+k$ variables $y_1, \ldots, y_{r+k}$. Set $k' := k$, we assume that $k \geq r$. First we add clauses to ensure that the variables in Y are 0 in every satisfying assignment of weight k'. We add the clause $R(z_1, \ldots, z_r)$ where the variables are distinct, at least one of $z_1, \ldots, z_\ell$ is in Y, and all of $z_{\ell+1}, \ldots, z_r$ are from Y. Considering all possibilities gives $O((n+k+r)^r)$ clauses. Assume that variable y_i is true in a weight k' satisfying assignment. Let $q_1, \ldots, q_{\ell-1}$ be $\ell-1$ other true variables (we can assume that $k \geq \ell$), they can be in Y or not in Y. Since at most k' variables are set to 1 in Y, thus there are variables $y_{i_1}, \ldots, y_{i_{r-\ell}}$ in Y with value 0. Now the clause $R(y_i, q_1, \ldots, q_{\ell-1}, y_{i_1}, \ldots, y_{i_{r-\ell}})$ is not satisfied, since there is 1 on the first ℓ positions and 0 after that, a contradiction. On the other hand, note that if every variable in Y is set to 0, then all the clauses are satisfied: each of them receives an assignment of weight at most $\ell-1$ that is the proper subset of $(\overbrace{1, \ldots, 1}^{\ell}, 0, \ldots, 0)$.

If there is an edge between vertices u and v, then we add the clauses $R(x_u, x_v, x_{i_1}, \ldots, x_{i_{\ell-2}}, y_1, \ldots, y_{r-\ell})$ where $x_{i_1}, \ldots, x_{i_{\ell-2}}$ are distinct variables not in Y. If one of x_u and x_v is 0 in a weight k' assignment, then all of these clauses are satisfied since they receive an assignment with weight less than ℓ, and 1 appears only on the first ℓ positions. On the other hand, if both x_u and x_v are 1, then one of these clauses is not satisfied: if we take $x_{i_1}, \ldots, x_{i_{\ell-2}}$ to be variables with value 1, then the clause $R(x_u, x_v, x_{i_1}, \ldots, x_{i_{\ell-2}}, y_1, \ldots, y_{r-\ell})$ is not satisfied. Therefore the constructed R-formula has a satisfying assignment of weight k' if and only if the graph has an independent set of size k, proving the correctness of the reduction. We note that r and ℓ are constants independent of k and n, hence the reduction is a uniformly polynomial-time parameterized reduction. □

Lemma 8.4 *Assume that $\mathscr{F}$ contains a 0-valid constraint R of arity r such that the pth position is useful but not satisfying. In this case $\mathscr{F}$-SAT$^*_0$ can be reduced to $\mathscr{F}$-SAT.*

Proof Let $r_{\max}$ be the maximum arity in $\mathscr{F}$. Given an $\mathscr{F}$-formula ϕ and an integer k, we construct an $\mathscr{F}$-formula ϕ' such that every clause of ϕ' contains every variable at most once and ϕ has a satisfying assignment of weight k if and only if ϕ' has a satisfying assignment of weight $k' := kr_{\max}$. Each variable x_i of ϕ is replaced by $r_{\max}$ new variables $x_{i,1}, \ldots, x_{i,r_{\max}}$. We also create a set Y of $k+r_{\max}$ new variables $y_1, \ldots, y_{k+r_{\max}}$. We add clauses to the formula to ensure that in every weight k' satisfying assignment of ϕ' the $r_{\max}$ variables $x_{i,1}, \ldots, x_{i,r_{\max}}$ have the same value, and the variables $y_1, \ldots, y_{k+r_{\max}}$ are set to 0. Now each clause of ϕ can be modified such that if the clause contains a variable x_i more than once, then we can use the variables $x_{i,1}, \ldots, x_{i,r_{\max}}$ to assign distinct variables for each occurrence of x_i in the clause. A constant 0 can be replaced by an arbitrary variable from Y. Clearly, there is a one-to-one correspondence between the weight k satisfying assignments of ϕ and the weight k' satisfying assignments of ϕ', proving the correctness of the reduction.

The new clauses are added as follows. Without loss of generality, it can be assumed that $(1, 0, \dots, 0)$ does not satisfy R, but $(\overbrace{1, \dots, 1}^{\ell}, 0, \dots, 0)$ satisfies R, and it has minimal weight among the satisfying assignments that have 1 at the first position. Add to the formula in every possible way a clause whose variables are taken from Y, there are $(r_{\max} + k)!/(r_{\max} + k - r)!$ such clauses. We claim that in every satisfying weight k' assignment the variables in Y have value 0. Assume that y_j is 1. Since only k' variables are set to 1, there have to be $r_{\max}$ variables $y_{j_1}, \dots, y_{j_{r_{\max}}}$ in Y with value 0, implying that the clause $R(y_j, y_{j_1}, \dots, y_{j_{r-1}})$ is not satisfied, a contradiction.

For each variable x_i of ϕ, we add clauses $R(x_{i,h_1}, \dots, x_{i,h_\ell}, y_1, \dots, y_{r-\ell})$ where $x_{i,h_1}, \dots, x_{i,h_\ell}$ are distinct variables. Considering all possibilities, this results in $r_{\max}!/(r_{\max}-\ell)!$ clauses for a variable x_i. We show that these clauses ensure that the variables $x_{i,1}, \dots, x_{i,k+r_{\max}}$ have the same value. Assume without loss of generality that $x_{i,1}$ is 1 and $x_{i,2}$ is 0 in a weight k' satisfying assignment of ϕ'. We have seen that every variable in Y is 0 in such an assignment, thus at most $\ell - 1$ variables are set to 1 in the clause $R(x_{i,1}, x_{i,2}, \dots, x_{i,\ell}, y_1, \dots, y_{r-\ell})$. However, there is 1 at the first position, but we assumed that every satisfying assignment with 1 at the first position has weight at least ℓ, a contradiction. Therefore the variables $x_{i,1}, \dots, x_{i,r}$ have the same value, as required. □

Now we are ready to put together the previous results and prove the negative side of Theorem 3.2.

Theorem 8.5 *Let $\mathscr{F}$ be a finite constraint family. If $\mathscr{F}$ contains a constraint that is not weakly separable, then $\mathscr{F}$-SAT is* W[1]*-complete.*

Proof Assume first that $\mathscr{F}$ contains a 0-valid constraint R_1 that is not weakly separable. We consider two cases depending on whether every useful position of R_1 is satisfying or not. If every useful position in R_1 is satisfying, then R_1-SAT is W[1]-complete by Lemma 8.3. On the other hand, if R_1 has a useful but not satisfying position, then by Lemma 8.4, R_1-SAT$^*_0$ can be reduced to R_1-SAT. By Lemma 8.1, R_1-SAT$^*_0$ is W[1]-complete, hence R-SAT is W[1]-complete in this case as well.

Assume now that $\mathscr{F}$ contains a 0-invalid non-weakly separable constraint R_2. By Lemma 2.7, R_2 has a 0-valid non-weakly separable restriction R'_2. We have seen in the previous paragraph that in this case R'_2-SAT is W[1]-complete. Furthermore, the constraint R_2 cannot be monotone: the restriction of a monotone constraint is also monotone, and a 0-valid monotone constraint is trivially weakly separable. Therefore Lemma 8.2 can be used to reduce $\mathscr{F}$-SAT$_{01}$ to $\mathscr{F}$-SAT. By Prop. 3.1, $\mathscr{F}$-SAT$_{01}$ is equivalent to $\mathscr{F}^*$-SAT, and $\mathscr{F}^*$ contains R'_2, thus the following series of reductions show that $\mathscr{F}$-SAT is W[1]-complete as well:

$$R'_2\text{-SAT} \preceq \mathscr{F}^*\text{-SAT} \overset{\text{Prop. 3.1}}{\preceq} \mathscr{F}\text{-SAT}_{01} \overset{\text{Lemma 8.2}}{\preceq} \mathscr{F}\text{-SAT}$$

□

9 Bounded treewidth

The *incidence graph* $I(\phi)$ of formula ϕ is a bipartite graph whose vertices are the variables and clauses of ϕ, and a clause is connected to those variables that appear in the clause. We show that certain structural assumptions on the incidence graph allows us to solve the $\mathscr{F}$-SAT problem in uniformly polynomial time for every constraint family $\mathscr{F}$.

Treewidth is a well-studied parameter of graphs. It is important from the algorithmic point of view, since a large number of hard problems becomes easy on bounded treewidth graphs (cf. [13]).

Definition 9.1 (Tree decomposition) *A* tree decomposition *of graph $G(V,E)$ is a rooted tree $T(U,F)$ together with a set $B_x \subseteq V$ for each node $x \in U$ such that*

1. *For every $v \in V$, the set of nodes in T that contain v induce a connected subgraph of T (a subtree of T).*
2. *For every edge $e = uv$ of G, there is a node x of T such that $u, v \in B_x$.*

Definition 9.2 (Treewidth) *The* treewidth *of a tree decomposition is* $\max_{x \in U} |B_x| - 1$*. The treewidth $w(G)$ of graph G is the smallest treewidth that its tree decomposition can have.*

The only reason for the -1 in the definition of treewidth is to ensure that graphs with treewidth 1 are exactly the forests.

A useful algorithmic trick is to consider only tree decompositions that have some nice properties [13]. Working with such tree decompositions makes the presentation of the algorithm considerably simpler (see [13]).

Definition 9.3 (Nice tree decomposition) *A tree decomposition $T(U,F)$, $\{B_x : x \in T\}$ is a* nice tree decomposition *of $G(V,E)$ if every node x of T has at most two children and it satisfies the following requirements:*

1. *If x has no children (x is a* leaf node*), then $B_x = \emptyset$.*
2. *If x has one child y, then either $B_x = B_y \cup \{v\}$ (x is an* add node*) or $B_x = B_y \setminus \{v\}$ (x is a* forget node*) for some $v \in V$.*
3. *If x has two children y and z, then $B_x = B_y = B_z$ (x is a* join node*).*

It turns out that bounded treewidth makes the problem easy in our case as well. Using the standard dynamic programming technique of tree decompositions, we can solve $\mathscr{F}$-SAT in uniformly polynomial time for every constraint family $\mathscr{F}$ if the incidence graph of the formula has bounded treewidth.

Theorem 9.4 *For every finite constraint family $\mathscr{F}$, the $\mathscr{F}$-SAT problem can be solved in $f(\mathscr{F}, w)k^2(n+m)$ time if the incidence graph of the formula has n variables, m clauses and treewidth at most w.*

Proof Consider a width w nice tree decomposition of G. For a node $x \in U$ of the tree decomposition, denote by C_x the set of clauses that appear in B_x (the set of x) or in the set of a descendant of x. Similarly, V_x denotes the variables that appear in the set of x or a descendant of x. We say that a variable is *active* at x if either it is contained in B_x, or it appears in a clause contained in B_x. For each node x, there can be at most $r(w+1)$ active variables, where r is the maximum arity of the constraints in $\mathscr{F}$. Denote by A_x the active variables at x and set $V'_x := V_x \cup A_x$. Clearly, a variable is in V'_x if and only if it appears in a clause of C_x.

We solve several subproblems for each node x of the tree. Each subproblem is characterized by an integer $0 \leq k' \leq k$ and an assignment to the active variables of x. Thus there are at most $k2^{r(w+1)}$ subproblems per node. For each subproblem we determine whether this assignment can be extended to an assignment of V'_x that has weight exactly k' and satisfies all the clauses in C_x. The problems are solved by bottom up dynamic programming: we start with the leaf nodes, and when we consider a non-leaf node, it is assumed that the subproblems are already solved for all its children. Below we describe what has to be done for the different types of nodes.

Leaf node x. Since B_x is empty, the problem is trivial.

Add node x. Given an assignment α of A_x and an integer k', we solve the problem as follows. Notice that if y is the child of x, then $A_y \subseteq A_x$, $V'_y \subseteq V'_x$ and $A_x \setminus A_y = V'_x \setminus V'_y$. Assignment α induces an assignment β of A_y. Denote by c the number of variables in $A_x \setminus A_y$ that receive 1 in α. Now α can be extended to a weight k' assignment of V'_x satisfying C_y if and only if β can be extended to a weight $k'-c$ assignment of V'_y also satisfying C_y. The answer to the latter problem was already determined when we considered node y. However, what we have to determine is whether α can be extended to an assignment that satisfies every clause in C_x, not only those in C_y. The set C_x can be larger than C_y only if the vertex added by the add node x is a clause. In this case all the variables of this new clause is in A_x, hence α itself determines whether this clause is satisfied or not.

Forget node x. If the child of x is y, then $V'_x = V'_y$, $C_x = C_y$ and $A_x \subseteq A_y$. Therefore if α is an assignment of A_x, then it can be extended to a weight k' assignment of V'_x that satisfies the clauses in C_x if and only if α can be extended to an assignment β of A_y such that β can be extended to a weight k' assignment of V'_y that satisfies the clauses in C_y. The existence of such a β can be easily determined if all the subproblems for node y are already solved. We enumerate all the assignments β for A_y, and check whether there is a β that induces α on A_x, and has the required extension.

Join node x. Let y and z be the children of x. It is easy to see that $A_x = A_y = A_z$, $C_x = C_y \cup C_z$, $V'_x = V'_y \cup V'_z$, and $V'_y \cap V'_z = A_x$. An assignment α of A_x can be extended to an assignment of V'_x satisfying C_x if and only if α can be extended to an assignment of V'_y satisfying C_y, and to an assignment of V'_z satisfying C_z. Having solved the subproblems for y and z, we can determine whether α has such extensions, hence we can answer whether it can be extended to V'_x. However, we have to find an extension of weight exactly k'. Assume that

α has weight c on A_x. If α has a weight k_1 extension to V'_y and a weight k_2 extension to V'_z, then this gives a weight $k_1 + k_2 - c$ extension of α to V'_x. Therefore it is not enough to check whether α can be extended to V'_y and V'_z, we have to find two extensions such that the sum of their weight is exactly $k' + c$. For each α and k', at most k different values of k_1 have to be tried: for each k_1 it has to be checked whether α has a weight k_1 extension to V'_y and a weight $k_2 = k' + c - k_1$ extension to V'_z. Given the solutions to the subproblems of y and z, this can be done without any difficulty.

Time complexity. The incidence graph has $n + m$ vertices, hence a tree decomposition of width w can be found in $f_1(w)(n+m)$ time [4]. Furthermore, the tree decomposition can be transformed into a nice tree decomposition in linear time.

For each node we solve at most $k2^{r(w+1)}$ subproblems. We can store the solutions to the subproblems in a lookup table, thus they can be accessed in constant time. As noted above, if x is an add node, then a subproblem for x can be solved in constant time if the subproblems for the child of x are already solved. If x is a forget node, the solutions for x can be easily obtained by enumerating the solutions for the child of x. If x is a join node, then a subproblem can be solved by checking at most k cases. Therefore the time spent at a node is k^2 times a constant (assuming that w and r are constants). Thus the total number of steps required by the algorithm is $f(w)k^2(n+m)$, for an appropriate function $f(w)$ independent of n and m. □

10 Planar formulae

A formula is *planar* if its incidence graph is a planar graph. The complexity of the satisfiability problem restricted to planar formulae was investigated in [14]: it was show that the problem remains NP-complete even with this restriction. The NP-completeness of planar SAT was used to determine the complexity of several planar and geometric problems. It turns out that for problems like maximum independent set, minimum dominating set, minimum vertex cover, etc., the planar version is as hard as the general problem.

However, in the world of parameterized complexity the situation is very different. The planar versions of maximum independent set and minimum dominating set are fixed-parameter tractable while the unrestricted problems are W[1]-hard [1]. In general, we show that $\mathscr{F}$-SAT is in FPT for every constraint family $\mathscr{F}$. The proof uses standard techniques: using the layering method of Baker [3], we can reduce the problem to bounded outerplanarity instances. Graphs with bounded outerplanarity have bounded treewidth, hence the algorithm of Theorem 9.4 can be used.

Definition 10.1 (t-outerplanar) *An embedding of graph $G(V, E)$ is* 1-outerplanar *(or simply* outerplanar*), if it is planar, and all vertices lie on the exterior face. For $t \geq 2$, an embedding of a graph $G(V, E)$ is* t-outerplanar*, if it is planar, and when all vertices on the outer face are deleted, then a $(t-1)$-outerplanar* *embedding of the resulting graph is obtained. A graph is* t-outerplanar*, if it has a t-outerplanar embedding. A t-outerplanar embedding divides the vertices into* t layers*: layer L_1 contains the vertices on the outer face, while for $i \geq 2$, layer L_i contains those vertices that are on the outer face after deleting layers L_1, $\dots$, L_{i-1}.*

Theorem 10.2 *For every finite constraint family $\mathscr{F}$, the $\mathscr{F}$-SAT problem can be solved in time $f(\mathscr{F}, k)(n+m)$ if the formula has n variables, m clauses, and a planar incidence graph.*

Proof A planar embedding of $I(\phi)$ can be found in linear time [11]. The embedding is t-outerplanar for some integer t, we can determine the layers L_1, $\dots$, L_t. The variables are partitioned into $k+1$ sets: let X_i $(0 \leq i \leq k)$ contain the variables in layer $L_{3(k+1)j+3i+\ell}$ for $j = 0, 1, \dots$ and $\ell = 1, 2, 3$. Clearly, every variable belongs to exactly one of these sets. Given a weight k satisfying assignment, in at least one of the $k+1$ sets all the variables are set to 0. For $i = 0, 1, \dots, k$, we check whether there is a weight k assignment where every variable in X_i is set to 0. If there is a weight k satisfying assignment, then we eventually find one for some i.

For a given i we proceed as follows. Replace every variable in X_i with the constant 0, and delete the corresponding vertices from the graph. Now all the vertices in layer $L_{3(k+1)j+3i+2}$ represent clauses. Moreover, since the variables appearing in such a clause have to be in layer $L_{3(k+1)j+3i+1}$, $L_{3(k+1)j+3i+2}$, or $L_{3(k+1)j+3i+3}$, all these variables were replaced by 0. If this assignment does not satisfy the clause (it is not 0-valid), then there is no satisfying assignment where the variables in X_i are zero. On the other hand, if the clause is 0-valid, then it is automatically satisfied in every such assignment, hence we can delete it from the formula and the graph. Thus for every $j = 0, 1, \dots,$ all the vertices in layer $L_{3(k+1)j+3i+2}$ are deleted, which means that the remaining graph is the disjoint union of $(3(k+1)-1)$-outerplanar graphs, which is also $(3(k+1)-1)$-outerplanar. A theorem of Bodlaender [5, Theorem 83] assures that a t-outerplanar graph has treewidth at most $3t-1$, therefore we have to solve the problem on a graph with treewidth at most $9(k+1)-4$, which can be done in linear time by Theorem 9.4. $\square$

Acknowledgments

I'm grateful to Katalin Friedl for her suggestions that greatly improved the presentation of the paper.

References

[1] J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: Exponential speed-up for planar graph problems. In *ICALP 2001*, volume 2076 of *Lecture Notes in Comput. Sci*, pages 261–272. Springer, Berlin, 2001.

[2] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[3] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. Assoc. Comput. Mach.*, 41(1):153–180, 1994.

[4] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[5] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1-2):1–45, 1998.

[6] A. A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proc. 43th Symp. Foundations of Computer Science*, pages 649–658. IEEE, November 2002.

[7] N. Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. System Sci.*, 51(3):511–522, 1995. 24th Annual ACM Symposium on the Theory of Computing (Victoria, BC, 1992).

[8] R. G. Downey and M. R. Fellows. *Parameterized complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.

[9] P. Erdős and R. Rado. Intersection theorems for systems of sets. *J. London Math. Soc.*, 35:85–90, 1960.

[10] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1999.

[11] J. Hopcroft and R. Tarjan. Efficient planarity testing. *J. Assoc. Comput. Mach.*, 21:549–568, 1974.

[12] S. Khanna, M. Sudan, L. Trevisan, and D. P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2001.

[13] T. Kloks. *Treewidth*, volume 842 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1994. Computations and approximations.

[14] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[15] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. 1978.