Better Algorithms for Satisfiability Problems for Formulas of Bounded Rank-width*

Robert Ganian, Petr Hliněný, and Jan Obdržálek

Faculty of Informatics, Masaryk University
Botanická 68a, Brno, Czech Republic
{xganian1,hlineny,obdrzalek}@fi.muni.cz

Abstract

We provide a parameterized polynomial algorithm for the propositional model counting problem #SAT, the runtime of which is single-exponential in the rank-width of a formula. Previously, analogous algorithms have been known – e.g. [Fischer, Makowsky, and Ravve] – with a single-exponential dependency on the clique-width of a formula. Our algorithm thus presents an exponential runtime improvement (since clique-width reaches up to exponentially higher values than rank-width), and can be of practical interest for small values of rank-width. We also provide an algorithm for the Max-SAT problem along the same lines.

Keywords and phrases propositional model counting; satisfiability; rank-width; clique-width; parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.73

1 Introduction

The satisfiability problem for Boolean formulas in conjunctive normal form (known as SAT) has been of great practical and theoretical interest for decades. It is known to be NP-complete, even though many instances are practically solvable using the various SAT-solvers. We focus on two well-known generalizations of this problem, namely #SAT and Max-SAT. In #SAT– otherwise known as the *propositional model counting* problem – the goal is to compute the number of satisfying truth assignments for an input formula ϕ, whereas in Max-SAT we ask for the maximum number of simultaneously satisfiable clauses of ϕ. It is known that computing #SAT is #P-hard [21] and that Max-SAT is already NP-hard to approximate within some constant [1].

In light of these hardness results, we may ask what happens if we restrict ourselves to some subclass of inputs. The parameterized algorithmics approach is suitable in such a case. Let k be some parameter associated with the input instance. Such a decision problem is said to be *fixed-parameter tractable (FPT)* if it is solvable in time $\mathcal{O}(n^p \cdot f(k))$ for some constant p and a computable function f. So the running time is polynomial in n, the size of the input, but can be e.g. exponential in the parameter k. Obviously the specific form of f plays an important role in practical applicability of any such algorithm – while FPT algorithms with single-exponential f can be feasible for non-trivial values of the parameter, a double-exponential f would make the algorithm impractical for almost all values of k.

But what are suitable parameters for satisfiability problems? In the particular case of Max-SAT, one can consider the desired number of satisfied or unsatisfied clauses as a

* This research has been supported by the Czech bilateral research grant GA 201/09/J021 and by the Institute for Theoretical Computer Science ITI, project 1M0545.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 73–83

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parameter of the input, such as in [4, 19], respectively. However such approach is not at all suitable for #SAT which is our prime interest in this paper.

Another approach used for instance by Fischer, Makowsky and Ravve [7] represents the formula ϕ as a formula graph F_ϕ (nodes of which are the clauses and variables of ϕ, see Definition 6), and exploits the fact that for graphs there are many known (and intensively studied) so-called *width parameters*. In [7] the authors presented FPT algorithms for the #SAT problem in the case of two well known width parameters – *tree-width* and *clique-width*. A similar idea was used by Georgiou and Papakonstantinou [10] also for the Max-SAT problem and by Samer and Szeider [20] for #SAT.

The latter algorithms work by dynamic programming on tree-like decompositions related to the width parameters (tree-decompositions and clique-decompositions – often called k-expressions – in the cases above). However, there is the separate issue of the complexity of computing the width of the formula graph and its decomposition. In the case of tree-width this can be done in FPT [2]. For the much more general clique-width (every graph of bounded tree-width also has bounded clique-width, while the converse does not hold) there exist no such algorithms and we rely on approximations or an oracle. In [20] the authors made the following statement on this issue:

> A single-exponential algorithm (for #SAT) is due to Fisher, Makowsky, and Ravve [7]. However, both algorithms rely on clique-width approximation algorithms. The known polynomial-time algorithms for that purpose admit an exponential approximation error [13] and are of limited practical value.

The exponential approximation error mentioned in this statement results by bounding the clique-width by a another, fairly new, width parameter called *rank-width* (Definition 2). Rank-width is bounded if and only if clique-width is bounded, but its value can be exponentially lower than that of clique-width (Theorem 3 a,b). And since clique-width generalizes tree-width, so does rank-width (Theorem 3 c). Moreover, for rank-width we can efficiently compute the related decomposition (Theorem 4), which is in stark contrast to the case for clique-width. Therefore an algorithm which is linear in the formula size and *single-exponential in its rank-width* challenges the claim quoted above, and can be of real practical value. In this paper we present such algorithms for the problems #SAT and Max-SAT (Theorems 9 and 17). Precisely we prove the following two results:

▶ **Theorem 1.** *Both the* #SAT *and* Max-SAT *problems have FPT algorithms running in time*

$$\mathcal{O}(t^3 \cdot 2^{3t(t+1)/2} \cdot |\phi|),$$

provided a rank-decomposition of the input instance (CNF formula) ϕ of width t is given on the input. If, on the other hand, no rank-decomposition is given along with the input ϕ, then the runtime estimate is

$$\mathcal{O}(2^{\Theta(t^2)} \cdot |\phi|^3)$$

where t is the rank-width of the input instance (CNF formula) ϕ.

We refer to further Theorems 11 and 9, 17 for details and the proofs.

Note that our results present an *actual exponential runtime improvement* in the parameter over any algorithm utilizing the clique-width measure, including aforementioned [7]. This is since any parameterized algorithm $\mathcal{A}$ for a SAT problem has to depend at least exponentially on the clique-width of a formula (unless the Exponential Time Hypothesis fails). Then, considering typical instances ϕ as from Proposition 8 b, such an algorithm $\mathcal{A}$ runs in time

which is double-exponential in the rank-width of ϕ even if $\mathcal{A}$ is given an optimal clique-width expression on the input.

As for potential practical usefulness of Theorem 1, note that there are no "huge hidden constants" in the $\mathcal{O}$-notation. One may also ask whether there are any interesting classes of graphs of low rank-width. The answer is a resounding YES, since already for $t = 1$ we obtain the very rich class of distance-hereditary graphs. Rank-width indeed is a very general graph width measure.

The approach we use to prove both parts of Theorem 1 quite naturally extends the elaborated new algebraic methods of designing parameterized algorithms for graphs of bounded rank-width, e.g. [6, 3, 8], to the area of SAT problems. Yet, this is not a trivial extension—we remark that a straightforward translation of the algorithm of [7] from clique-width expressions to rank-decompositions (which is easily possible) would result just in a double-exponential runtime dependency on the rank-width.

The rest of the paper is organized as follows: In Section 2 we present the rank-width measure and some related technical considerations. This is applied to signed graphs of SAT formulas. Section 3 then presents our FPT algorithm for the #SAT problem (Theorem 9 and Algorithm 16), and Section 4 the similar algorithm for Max-SAT (Theorem 17). We conclude with some related observations.

2 Definitions

2.1 Rank-width

The usual way of defining rank-width [18] is via the branch-width of the cut-rank function (Definition 2). A set function $f : 2^M \to \mathbb{Z}$ is *symmetric* if $f(X) = f(M \setminus X)$ for all $X \subseteq M$. A tree is *subcubic* if all its nodes have degree at most 3. For a symmetric function $f : 2^M \to \mathbb{Z}$ on a finite ground set M, the branch-width of f is defined as follows:

A *branch-decomposition* of f is a pair (T, μ) of a subcubic tree T and a bijective function $\mu : M \to \{t : t \text{ is a leaf of } T\}$. For an edge e of T, the connected components of $T \setminus e$ induce a bipartition (X, Y) of the set of leaves of T. The *width* of an edge e of a branch-decomposition (T, μ) is $f(\mu^{-1}(X))$. The *width* of (T, μ) is the maximum width over all edges of T. The *branch-width* of f is the minimum of the width of all branch-decompositions of f.

▶ **Definition 2. (Rank-width [18])** For a simple undirected graph G and $U, W \subseteq V(G)$, let $\boldsymbol{A}_G[U, W]$ be the matrix defined over the two-element field GF(2) as follows: the entry $a_{u,w}$, $u \in U$ and $w \in W$, of $\boldsymbol{A}_G[U, W]$ is 1 if and only if uw is an edge of G. The *cut-rank* function $\rho_G(U) = \rho_G(W)$ then equals the rank of $\boldsymbol{A}_G[U, W]$ over GF(2) where $W = V(G) \setminus U$. A *rank-decomposition* (see Figure 1) and *rank-width* of a graph G is the branch-decomposition and branch-width of the cut-rank function ρ_G of G on $M = V(G)$, respectively.

As already mentioned in the introduction, rank-width is closely related to clique-width and more general than better known tree-width. Indeed:

▶ **Theorem 3.** *Let G be a simple graph, and* $\mathrm{tw}(G)$, $\mathrm{bw}(G)$, $\mathrm{cwd}(G)$ *and* $\mathrm{rwd}(G)$ *denote in this order the tree-width, branch-width, clique-width, and rank-width of G. Then*

a) [18] $\mathrm{rwd}(G) \leq \mathrm{cwd}(G) \leq 2^{\mathrm{rwd}(G)+1} - 1$,
b) [5] the clique-width $\mathrm{cwd}(G)$ *can reach up to* $2^{\mathrm{rwd}(G)/2-1}$,
c) [17] $\mathrm{rwd}(G) \leq \mathrm{bw}(G) \leq \mathrm{tw}(G) + 1$,
d) [folklore] $\mathrm{tw}(G)$ *cannot be bounded from above by a function of* $\mathrm{rwd}(G)$*, e.g. the complete graphs have rank-width* 1 *while their tree-width is unbounded,*

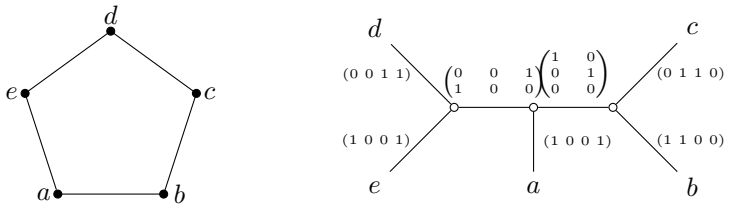

Figure 1 A rank-decomposition of the graph cycle C_5, showing the matrices involved in evaluation of its cut-rank function on the edges of the decomposition.

e) [15] $\mathrm{rwd}(G) = 1$ *if and only if* G *is a distance-hereditary graph.*

Although rank-width and clique-width are "tied together" by Theorem 3 a, one of the crucial advantages of rank-width is its parameterized tractability (note that it is not known how to efficiently test $\mathrm{cwd}(G) \leq k$ for $k > 3$):

▶ **Theorem 4.** [13] *There is an FPT algorithm that, for a fixed parameter* t *and a given graph* G*, either finds a rank-decomposition of* G *of width at most* t *or confirms that the rank-width of* G *is more than* t *in time* $O(|V(G)|^3)$*.*

With regard to our interest in precise runtime dependency on the parameter rank-width one should ask how the FPT algorithm of Theorem 4 depends on t. This issue is not at all addressed in [13], but a second look at the algorithm reveals a worst-case triple-exponential dependency on t (this is the currently best upper bounded on the number of forbidden minors for the matroids of branch-width t). One can do better while sacrificing the exact value of rank-width, such as using [18] or an improvement of the second algorithm of [16]:

▶ **Theorem 5.** [16] *There is an algorithm that, for a fixed parameter* t *and a given graph* G*, either finds a rank-decomposition of* G *of width at most* $3t$ *or confirms that the rank-width of* G *is more than* t *in time* $O(2^{\Theta(t^2)} \cdot |V(G)|^3)$*.*

2.2 Signed graphs of CNF formulas

There are several methods for converting formulas to graphs, the two most common and perhaps most natural approaches utilizing directed graphs (as seen e.g. in [20]) or so-called *signed graphs* (e.g. [7, 20, 12]). We employ the latter approach for technical reasons, although all the results also transfer straightforwardly to the former one. A signed graph is a graph G with two edge sets $E^+(G)$ and $E^-(G)$. We refer to its respective positive and negative subgraphs as to G^+ and G^-. Notice that G^+ and G^- are edge-disjoint and $G = G^+ \cup G^-$.

▶ **Definition 6.** The signed graph F_ϕ of a CNF formula ϕ is defined as follows:
- $V(F_\phi) = W \cup C$ where W is the set of variables of ϕ and C is the set of clauses of ϕ.
- For $w \in W$ and $c \in C$, it is $wc \in E^+(F_\phi)$ iff the literal 'w' occurs in c.
- For $w \in W$ and $c \in C$, it is $wc \in E^-(F_\phi)$ iff the literal '$\neg w$' occurs in c.

Since signed graphs have two distinct edge sets, the definition of rank-width needs to be modified to reflect this. It should be noted that simply using two separate, independent decompositions would not work – the bottom-up dynamic programming algorithm we are going to use will need information from both edge sets at every node to work properly. Instead, one may define, analogically to Definition 2, the *signed rank-width* of a signed graph G as the branch-width of the signed cut-rank function $\rho^{\pm}_G(U) = \rho_{G^+}(U) + \rho_{G^-}(U)$.

▶ **Definition 7. (Rank-width of formulas)** The *(signed) rank-width* $\mathrm{rwd}(\phi)$ of a CNF formula ϕ is the signed rank-width of the signed formula graph F_ϕ.

We remark that, although our signed rank-width is essentially equivalent to an existing concept of bi-rank-width of directed graphs as introduced by Kanté [14] (in the bipartite case, at least), the latter concept is not widely known and its introduction in the context of CNF formulas would bring only additional technical complications.

In our paper we propose signed rank-width as a way of measuring complexity of formulas that fares significantly better than previously considered signed clique-width of F_ϕ (e.g. [7]). *Signed clique-width* is the natural extension of clique-width having two separate operators for creating the 'plus' and the 'minus' edges. The main advantage of using rank-width as a parameter instead of clique-width comes from the following claim:

▶ **Proposition 8.** *Let ϕ be an arbitrary CNF formula and let $\mathrm{cwd}(\phi)$ denote the signed clique-width of F_ϕ. Then the following are true*

a) $\mathrm{rwd}(\phi) \leq 2\,\mathrm{cwd}(\phi)$,
b) there exist instances ϕ such that $\mathrm{cwd}(\phi) \geq 2^{\mathrm{rwd}(\phi)/4-1}$.

Proof. a) Assume a signed k-expression tree S for F_ϕ where $k = \mathrm{cwd}(\phi)$. Clearly, S gives ordinary k-expression trees for each of F^+_ϕ, F^-_ϕ. Now, analogically to Theorem 3 a [18], the rank-decompositions of F^+_ϕ and F^-_ϕ with the same underlying tree as S have widths $\leq k$ each, and hence $\mathrm{rwd}(\phi) \leq k + k = 2\,\mathrm{cwd}(\phi)$.

b) We define ϕ such that $F^+_\phi = G$ where, cf. Theorem 3 b, $\mathrm{cwd}(G) \geq 2^{\mathrm{rwd}(G)/2-1}$, and F^-_ϕ is arbitrary such that its rank-decomposition inherited from that of G has width $\leq \mathrm{rwd}(G)$. Then $\mathrm{rwd}(\phi) \leq 2\,\mathrm{rwd}(G)$ and the claim follows since $\mathrm{cwd}(\phi) \geq \mathrm{cwd}(G)$. ◀

3 Algorithm for Propositional Model Counting #SAT

This section proves our main result – the #SAT part of Theorem 1:

▶ **Theorem 9.** *Given a CNF formula ϕ and a (t^+, t^-)-labeling parse tree of the signed formula graph F_ϕ, there is an algorithm that counts the number of satisfying assignments of ϕ in time*

$$\mathcal{O}(t^3 \cdot 2^{3t(t+1)/2} \cdot |\phi|) \qquad \text{where } t = \max(t^+, t^-).$$

Note that the statement speaks about so called labeling parse trees (in place of rank-decompositions) which is a technical algebraic concept [6] suited for easier handling of rank-decompositions – this concept is briefly introduced in Section 3.1 to follow.

Our algorithm (see Algorithm 16) proving Theorem 9 applies the dynamic programming paradigm on the parse tree of the formula graph F_ϕ (constructed by Theorem 11). This is, on one hand, a standard approach utilized also by Fischer, Makowsky and Ravve [7]. On the other hand, however, in comparison to [7] we achieve an exponential runtime speedup in terms of rank-width, exploiting the nice linear-algebraic properties of rank-decompositions and their parse trees – see Section 3.2.

3.1 Parse trees for rank-decompositions

First of all we remark that, unlike for tree-width or clique-width, a bare rank-decomposition is not suitable for immediate design of algorithms. The remedy is provided by the notion of labeling parse trees [6, 8] which, informally saying, "enrich" a rank-decomposition with additional information captured in labelings of the vertices.

A *t-labeling* of a graph is a mapping $lab : V(G) \to 2^{[t]}$ where $[t] = \{1, 2, \ldots, t\}$ is the set of *labels*. Having a graph G with an associated t-labeling lab, we refer to the pair $\bar{G} = (G, lab)$ as to a *t-labeled graph*. A canonical operation associated with t-labeled graphs $\bar{G}_1 = (G_1, lab^1)$ and $\bar{G}_2 = (G_2, lab^2)$ is the *labeling join* $\bar{G}_1 \otimes \bar{G}_2$, defined on the disjoint union of G_1 and G_2 by adding all edges (u, v) such that $|lab^1(u) \cap lab^2(v)|$ is odd, where $u \in V(G_1), v \in V(G_2)$ (the resulting graph is unlabeled).

The labeling concept is similar to the definition of clique-width where every graph vertex carries one label (while here we allow a set of labels). It is actually more powerful to view a t-labeling of G equivalently as a mapping $V(G) \to \mathrm{GF}(2)^t$ into the *binary vector space* of dimension t, where GF(2) is the two-element finite field. (Then an edge $\{u, v\}$ is added in $\bar{G}_1 \otimes \bar{G}_2$ if and only if $lab^1(u) \cdot lab^2(v) = 1$ as a scalar product of vectors.)

As shown first by Courcelle and Kanté in [6], a rank-decomposition of width t can be equivalently characterized by a t-labeling parse tree (thereafter called an algebraic expression for rank-width). A *t-labeling parse tree* T for the t-labeled graph $\bar{G}$ is a finite rooted ordered subcubic tree such that

- the leaves of T form the vertex set of $\bar{G}$, and
- for each internal node z of T, the subtree $T_z \subseteq T$ rooted at z builds up a t-labeled graph $\bar{G}_z$ which (up to relabeling) equals the subgraph of $\bar{G}$ induced on the leaves of T_z.

To keep this paper simple and accessible, we skip a (rather long) formal definition of the algebraic operations taking place in the internal nodes of a t-labeling parse tree. Instead, we summarize their properties which will be used in our algorithm:

▶ **Property 10.** [8] *Let a t-labeled graph $\bar{G}$ have a t-labeling parse tree T. For every internal node z of T with a son (left or right) x, the following holds.*

a) The labels of the vertices of $V(G_x)$ in $\bar{G}_z$ are obtained from the corresponding labels in $\bar{G}_x$ by a linear transformation (relabeling) over the binary vector space $\mathrm{GF}(2)^t$.
b) For each vertex $v \in V(G_z)$, the $\bar{G}$-neighbours of v in the set $V(G) \setminus V(G_z)$ depend only on the label of v in $\bar{G}_z$. More precisely, there exists a t-labeled graph $\bar{H}$ such that unlabeled H is equal to the subgraph of G induced on $V(G) \setminus V(G_z)$, and $G = \bar{G}_z \otimes \bar{H}$.

In the context of signed graphs we, moreover, introduce the following two straightforward extensions:

- A signed graph G with associated pair of labelings $lab^+ : V(G) \to 2^{[t^+]}$ and $lab^- : V(G) \to 2^{[t^-]}$ is called a (t^+, t^-)-*labeled graph* $\widetilde{G} = (G, lab^+, lab^-)$. We use $\widetilde{G}^+$ as a shorthand for the t^+-labeled graph (G^+, lab^+), and $\widetilde{G}^-$ is defined analogically. The scope of the join operation $\otimes$ is then extended in a natural way: $\widetilde{G}_1 \otimes \widetilde{G}_2 = \big(\widetilde{G}_1^+ \otimes \widetilde{G}_2^+\big) \cup \big(\widetilde{G}_1^- \otimes \widetilde{G}_2^-\big)$.
- A (t^+, t^-)-*labeling parse tree* $T = (T^+, T^-)$ of a signed graph G is a pair (T^+, T^-) of labeling parse trees T^+ and T^- such that T^+ (T^-) is a t^+-labeling (t^--labeling) parse tree generating G^+ (G^-), and the underlying rooted trees of T^+ and T^- are identical.

The underlying idea is to separately handle the labelings for the positive and negative subgraphs of signed G, while keeping the same parse tree structure for both.

Finally, we shall use the following extension of Theorem 5 which makes restricted use of "partitioned" rank-width of vertex-partitioned graphs.

▶ **Theorem 11.** *There is an algorithm that, for a fixed parameter t and a given CNF formula ϕ, in time $O(2^{\Theta(t^2)} \cdot |\phi|^3)$ either finds a (t^+, t^-)-labeling parse tree for the formula graph F_ϕ where $t^+ \leq 3t$ and $t^- \leq 3t$, or confirms that the signed rank-width of ϕ is more than t.*

3.2 Recording partial assignments of a formula

The core of every dynamic programming algorithm is the specification of information which is then (exhaustively) processed on the input parse tree. Here we heavily apply the basic calculus of linear algebra (which is indeed natural in view of the definition of rank-width).

We say that labeling ℓ is *orthogonal* to a set of labelings X if ℓ has an even intersection with every element of X (i.e. the scalar product of the labeling vectors is 0 over GF(2)). Remember that for t-labeled graphs, in order for two vertices to become adjacent by the join operation $\otimes$, their labelings need to have an odd intersection, i.e. to be non-orthogonal. The power of orthogonality comes from the following rather trivial claim appearing already in [3, 8]:

▶ **Lemma 12.** [3, 8] *Assume t-labeled graphs $\bar{G}$ and $\bar{H}$, and arbitrary $X \subseteq V(\bar{G})$ and $y \in V(\bar{H})$. In the join graph $\bar{G} \otimes \bar{H}$, the vertex y is adjacent to some vertex in X if and only if the vector subspace spanned by the $\bar{G}$-labelings of the vertices of X is not orthogonal to the $\bar{H}$-labeling vector of y in* $\mathrm{GF}(2)^t$.

In view of Lemma 12, the following result will be useful in deriving the complexity of our algorithm.

▶ **Lemma 13.** [11] (cf. [8, Proposition 6.1]) *The number $S(t)$ of subspaces of the binary vector space* $\mathrm{GF}(2)^t$ *satisfies $S(t) \leq 2^{t(t+1)/4}$ for all $t \geq 12$.*

In the course of computation of our algorithm we need to remember some local information about all satisfying assignments for ϕ. The information to be remembered for each such assignment will be formally described in Definition 14. Before that, we would like to informally introduce its key idea of recording an "expectation" (when processing the parse tree of the input) – in addition to the information recorded about a partial solution processed so far, we also keep what is expected from a complementary partial solution coming from the unprocessed part of the input (cf. Definition 14. II).

The idea behind the algorithm is that the amount of information one has to remember about a partial solution shrinks a lot if one "knows" what the complete solution will look like. Such saving sometimes largely exceeds the cost of keeping an exhaustive list of all possible future shapes of complete solutions. This is also our case: we may exhaustively preprocess the values of some variables in advance.

The idea of using an "expectation" to speed up a dynamic programming algorithm on a rank-decomposition has first appeared in Bui-Xuan, Telle and Vatshelle [3] in relation to solving the dominating set on graphs of bounded rank-width. This concept has been subsequently formalized and generalized by the authors in [8] (in the so called PCE scheme formalism). Furthermore, it has also been shown [8, Proposition 5.1] that use of the "expectation" concept is unavoidable to achieve speed up for the dominating set problem which shares, in a sense, a similar background with our situation.

By Definition 6 the signed graph F_ϕ of a formula ϕ has a vertex set $V(F_\phi) = W \cup C$ where W is the set of variables and C is the set of clauses of ϕ. An *assignment* is then a mapping $\nu : W \to \{0, 1\}$. We speak about a *partial assignment* if ν is a partial mapping from a specific subset of W (a situation that is typical in dynamic processing). For every partial assignment we then define:

▶ **Definition 14.** Consider an arbitrary (t^+, t^-)-labeling $\widetilde{F}_1 = (F_1, lab^+, lab^-)$ of a signed subgraph $F_1 \subseteq F_\phi$, and any partial assignment $\nu_1 : V(F_1) \cap W \to \{0, 1\}$. We say that ν_1 is an *assignment of shape* $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_1$ if

I. Σ^+ is the subspace of $\mathrm{GF}(2)^t$ generated by the set of labeling vectors $lab^+(\nu_1^{-1}(1))$ and Σ^- is the subspace of $\mathrm{GF}(2)^t$ generated by $lab^-(\nu_1^{-1}(0))$, and
II. Π^+, Π^- (the *expectation part* of the shape) are subspaces of $\mathrm{GF}(2)^t$ such that, for every clause $c \in V(F_1) \cap C$, at least one of the following is true
   - c is adjacent to some vertex from $\nu_1^{-1}(1)$ in F_1^+ or to one from $\nu_1^{-1}(0)$ in F_1^-, or
   - the label vector $lab^+(c)$ is not orthogonal to Π^+ or $lab^-(c)$ is not orthogonal to Π^- (cf. Lemma 12).

Very informally saying, I. states which true literals in F_1 (w.r.t. ν_1) are available to satisfy clauses of F_ϕ, and II. stipulates that every clause in F_1 is already satisfied by a literal in F_1 or is expected to be satisfied by some literal in $F_\phi \setminus V(F_1)$. Note that one partial assignment ν_1 could be of several distinct shapes, which differ just in the expectation part, i.e. in Π^+, Π^-. This is true even for complete assignments. Moreover, there is no requirement on Π^+ and Π^- to have an empty intersection with Σ^+, Σ^- and each other.

From the definition, considering Property 10 b and Lemma 12, one easily gets:

▶ **Proposition 15.** *We consider a CNF formula ϕ with the variable set W, and any assignment $\nu : W \to \{0,1\}$. Assume $\widetilde{F}_1, \widetilde{F}_2$ are (t^+,t^-)-labeled graphs such that $F_\phi = \widetilde{F}_1 \otimes \widetilde{F}_2$, and let ν_1, ν_2 denote the restrictions of ν to $\widetilde{F}_1, \widetilde{F}_2$.*

a) The assignment ν is satisfying for ϕ if, and only if, there exist subspaces Σ^+, Σ^-, Π^+, Π^- of $\mathrm{GF}(2)^t$ such that ν_1 is of shape $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_1$ and ν_2 is of shape $(\Pi^+, \Pi^-, \Sigma^+, \Sigma^-)$ in $\widetilde{F}_2$.
b) If, in $\widetilde{F}_1$, ν_1 is of shape $(\Sigma_0^+, \Sigma_0^-, \Pi_0^+, \Pi_0^-)$ and, at the same time, ν_1 is of shape $(\Sigma_1^+, \Sigma_1^-, \Pi_1^+, \Pi_1^-)$, then $\Sigma_0^+ = \Sigma_1^+$ and $\Sigma_0^- = \Sigma_1^-$.
c) The assignment ν_1 is satisfying for ϕ_1 – the subformula of ϕ represented by F_1 if, and only if, ν_1 is of shape $(\Sigma^+, \Sigma^-, \emptyset, \emptyset)$ for some subspaces Σ^+, Σ^-.

◀

3.3 The dynamic processing algorithm

We now return to the proof of our Theorem 9.

▶ **Algorithm 16.** (Theorem 9) *Given is a CNF formula ϕ and a signed (t^+,t^-)-labeling parse tree T_ϕ of the formula graph F_ϕ. Our task is to compute the total number of satisfying assignments of ϕ:*

At every node z such that the subtree of T_ϕ rooted at z parses a (t^+,t^-)-labeled graph $\widetilde{F}_z$, we create an integer-valued array $Table_z$ indexed by all the quadruples of subspaces of $\mathrm{GF}(2)^t$, where $t = \max(t^+,t^-)$. The value of the entry $Table_z[\Sigma^+, \Sigma^-, \Pi^+, \Pi^-]$ shall be equal to the total number of variable assignments in the subformula of ϕ corresponding to $F_z \subseteq F_\phi$ that are of the shape $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_z$ (cf. Definition 14).

The computation is (briefly, details in [9]) as follows.

1. We initialize all entries of $Table_z$ for $z \in V(T_\phi)$ to 0.
2. We process all nodes of T_ϕ in the leaves-to-root order as follows.
   a) At a clause leaf c of T_ϕ, we set $Table_c[\emptyset, \emptyset, \Pi^+, \Pi^-] = 1$ (there is just one, empty, possible partial assignment in $\widetilde{F}_c$) for all pairs of expectation subspaces Π^+, Π^- that would satisfy this clause c.
   b) At a variable leaf ℓ of T_ϕ, we record one partial variable assignment with $\ell = 1$ and one with $\ell = 0$, both for all possible expectation pairs Π^+, Π^- (this is since there is no clause in $\widetilde{F}_\ell$, and so any expectation would work).

c) Consider an internal node z of T_ϕ, with the left son x and the right son y such that $Table_x$ and $Table_y$ have already been computed. We loop exhaustively over all "compatible" triples of indices to $Table_x$, $Table_y$, and $Table_z$ as follows.
   - We say that the indices in $Table_x[\Sigma_x^+, \Sigma_x^-, \Pi_x^+, \Pi_x^-]$, $Table_y[\Sigma_y^+, \Sigma_y^-, \Pi_y^+, \Pi_y^-]$, and $Table_z[\Sigma_z^+, \Sigma_z^-, \Pi_z^+, \Pi_z^-]$ are *compatible* if, e.g., the space Σ_z^+ spans the union of relabeled (cf. Property 10 a) spaces Σ_x^+, Σ_y^+, and the expectation space Π_x^+ is spanned by the (again relabeled accordingly) expectation space Π_z^+ and the space Σ_y^+ (which records the true literals of the corresponding $Table_y$ entry). Analogical conditions ought to be true for Σ_z^-, and $\Pi_x^-, \Pi_y^+, \Pi_y^-$.
   - Whenever such a compatible triple of table indices is found, we add up the product $Table_x[\Sigma_x^+, \Sigma_x^-, \Pi_x^+, \Pi_x^-] \cdot Table_y[\Sigma_y^+, \Sigma_y^-, \Pi_y^+, \Pi_y^-]$ to the entry $Table_z[\Sigma_z^+, \Sigma_z^-, \Pi_z^+, \Pi_z^-]$.

3. For the root r of T_ϕ, we sum up all the entries $Table_r[\Sigma^+, \Sigma^-, \emptyset, \emptyset]$ where Σ^+, Σ^- are arbitrary subspaces of $\mathrm{GF}(2)^t$. This is the resulting number of satisfying assignments of ϕ according to Proposition 15 c.

Correctness of Algorithm 16 routinely follows from Definition 14 of assignment shapes, Properties 10 of labeling parse trees, and Proposition 15. Our key novel contribution, on the other hand, is the runtime analysis of this algorithm in which we employ subspace orthogonality and the notion of *expectation* outlined in Section 3.2.

Runtime of Algorithm 16 is dominated by the calls to step (2c). Although we say that we exhaustively loop over all 12-tuples of subspaces of $\mathrm{GF}(2)^t$ there, actually half of the subspaces are determined by the others using linear algebra. Hence one call to this step takes time $\mathcal{O}\big(t^3 \cdot S(t)^6\big)$ where $S(t)$ bounds the number of subspaces as in Lemma 13. Altogether our Algorithm 16 takes time

$$\mathcal{O}\big(|V(T_\phi)| \cdot t^3 \cdot S(t)^6\big) = \mathcal{O}\big(|V(T_\phi)| \cdot t^3 \cdot 2^{6t(t+1)/4}\big) = \mathcal{O}\big(|\phi| \cdot t^3 \cdot 2^{3t(t+1)/2}\big).$$

4 Algorithm for the Max-SAT Problem

The same ideas as presented in Section 3 lead also to a parameterized algorithm for the Max-SAT optimization problem which asks for the maximum number of satisfied clauses in a CNF formula. We briefly describe this extension, though we have to admit that the importance of the Max-SAT algorithm on graphs of bounded rank-width is not as high as that of #SAT. The reason for lower applicability is that for "sparse" formula graphs (i.e. those not containing large bipartite cliques) their rank-width is bounded iff their tree-width is bounded, while for dense formula graphs the satisfiability problem is easier in general.

▶ **Theorem 17.** *There is an algorithm that, given a CNF formula ϕ and a (t^+, t^-)-labeling parse tree of the formula graph F_ϕ, solves the* Max-SAT *optimization problem of ϕ in time $\mathcal{O}(t^3 \cdot 2^{3t(t+1)/2} \cdot |\phi|)$ where $t = \max(t^+, t^-)$.*

In order to formulate this algorithm, we extend Definition 14 as follows. Recall $V(F_\phi) = W \cup C$ where W are the variables and C are the clauses of ϕ.

▶ **Definition 18.** Consider a (t^+, t^-)-labeling $\widetilde{F}_1 = (F_1, lab^+, lab^-)$ of a signed subgraph $F_1 \subseteq F_\phi$, and a partial assignment $\nu_1 : V(F_1) \cap W \to \{0, 1\}$. We say that ν_1 is an *assignment of defective shape* $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_1$ if there exists a set $C_0 \subseteq C \cap V(F_1)$ such that ν_1 is of shape $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_1 - C_0$. The value (the *defect*) of ν_1 with respect to this defective shape is the minimum cardinality of such C_0.

Informally, the defect equals the number of clauses in F_1 which are unsatisfied there and not expected to be satisfied in a complete assignment in F_ϕ.

▶ **Algorithm 19.** (Theorem 17) *Given is a CNF formula ϕ and a signed (t^+, t^-)-labeling parse tree T_ϕ of the formula graph F_ϕ. Our task is to compute the maximum number of satisfied clauses over all variable assignments of ϕ. A brief sketch follows, while the details can be found in [9]:*

We process the parse tree T_ϕ of F_ϕ similarly to Algorithm 16, but this time the value of the entry $Table_z[\Sigma^+, \Sigma^-, \Pi^+, \Pi^-]$ shall be equal to the minimum defect over all partial assignments in $\widetilde{F}_z$ that are of defective shape $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$. Within the framework of Algorithm 16 we, in an internal node z of T_ϕ with the sons x, y, determine the defect of any partial assignment in $\widetilde{F}_z$ (with respect to a particular shape, cf. Definition 18) as the sum of the defects of the corresponding (inherited) partial assignments in $\widetilde{F}_x$ and $\widetilde{F}_y$. The implementation is similar to the former.

At the end of processing, in the root r of T_ϕ, we find the minimum m over all the entries $Table_r[\Sigma^+, \Sigma^-, \emptyset, \emptyset]$ where Σ^+, Σ^- are arbitrary subspaces of $\mathrm{GF}(2)^t$. An optimal solution to Max-SAT of ϕ then has $|C| - m$ satisfied clauses.

5 Conclusions

We have presented new FPT algorithms for the #SAT and Max-SAT problems on formulas of bounded rank-width. Our algorithms are single-exponential in rank-width and linear in the size of the formula (cubic if a rank-decomposition has to be computed first), and they do not involve any "large hidden constants". This is a significant improvement over previous results, for several reasons. In the case of tree-width this follows from the fact that rank-width is much less restrictive than tree-width. If a graph has bounded tree-width it also has bounded rank-width, but there are classes of graphs with arbitrarily high tree-width and small rank-width (e.g. cliques, complete bipartite graphs, or distance hereditary graphs).

As for clique-width (which is bounded iff rank-width is bounded), we have obtained two significant improvements over the existing algorithms such as [7]. Firstly, rank-width can be exponentially smaller than clique-width, and therefore we obtain an exponential speed-up over the existing algorithms in the worst case. Secondly, there is an FPT algorithm for computing the rank-width of a graph (and the associated rank-decomposition) exactly, or a faster one providing a factor-3 approximation, whereas in the case of clique-width we have to rely on an approximation by an exponential function of rank-width.

Finally, our paper shows that many of the recent ideas and tricks of parameterized algorithm design on graphs of bounded rank-width smoothly translate to certain SAT-related problem instances, a fact which may also provide new inspiration for related research.

References

1 S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45:70–122, 1998.

2 H. Bodlaender. Treewidth: Algorithmic techniques and results. In *MFCS'97*, volume 1295 of *LNCS*, pages 19–36. Springer, 1997.

3 B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discrete Appl. Math.*, 158(7):809–819, 2010.

4 J. Chen and I. A. Kanj. Improved exact algorithms for Max-Sat. *Discrete Appl. Math.*, 142(1-3):17–27, 2004.

5 D. Corneil and U. Rotics. On the relationship between cliquewidth and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

6 B. Courcelle and M. Kanté. Graph operations characterizing rank-width. *Discrete Appl. Math.*, 157(4):627–640, 2009.

7 E. Fischer, J.A. Makowsky, and E. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Appl. Math.*, 156:511–529, 2008.

8 R. Ganian and P. Hliněný. On parse trees and Myhill–Nerode–type tools for handling graphs of bounded rank-width. *Discrete Appl. Math.*, 158:851–867, 2010.

9 R. Ganian, P. Hliněný, and J. Obdržálek. Better algorithms for satisfiability problems for formulas of bounded rank-width. arXiv:1006.5621v1 [cs.DM], June 2010.

10 K. Georgiou and P.A. Papakonstantinou. Complexity and algorithms for well-structured k-SAT instances. In *SAT'08*, pages 105–118, 2008.

11 J. Goldman and G.-C. Rota. The number of subspaces of a vector space. In W.T. Tutte, editor, *Recent Progress in Combinatorics*, pages 75–83. Academic Press, 1969.

12 G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving MAX-2-SAT above a tight lower bound. arXiv:0907.4573, July 2009.

13 P. Hliněný and S. Oum. Finding branch-decomposition and rank-decomposition. *SIAM J. Comput.*, 38:1012–1032, 2008.

14 M. Kanté. The rank-width of directed graphs. arXiv:0709.1433v3, March 2008.

15 S. Oum. Rank-width and vertex-minors. *J. Comb. Theory Ser. B*, 95(1):79–100, 2005.

16 S. Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):1–20, 2008.

17 S. Oum. Rank-width is less than or equal to branch-width. *J. Graph Theory*, 57(3):239–244, 2008.

18 S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

19 I. Razgon and B. O'Sullivan. Almost 2-SAT is fixed-parameter tractable. *J. Comput. System Sci.*, 75(8):435–450, 2009.

20 M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Alg.*, 8:50–64, 2010.

21 L. G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.