



ELSEVIER



Discrete Applied Mathematics ■■■ (■■■) ■■■–■■■

DISCRETE
APPLIED
MATHEMATICS

www.elsevier.com/locate/dam

Counting truth assignments of formulas of bounded tree-width or clique-width

E. Fischer^{a,1}, J.A. Makowsky^{a,2}, E.V. Ravve^{b,3}

^aDepartment of Computer Science, Technion—Israel Institute of Technology, Haifa, Israel

^bTower Semiconductors Ltd., Migdal Haemek, Israel

Received 31 January 2004; received in revised form 7 June 2005; accepted 7 June 2006

Abstract

We study algorithms for \sharp SAT and its generalized version \sharp GENSAT, the problem of computing the number of satisfying assignments of a set of propositional clauses Σ . For this purpose we consider the clauses given by their incidence graph, a signed bipartite graph $SI(\Sigma)$, and its derived graphs $I(\Sigma)$ and $P(\Sigma)$.

It is well known, that, given a graph of tree-width k , a k -tree decomposition can be found in polynomial time. Very recently Oum and Seymour have shown that, given a graph of clique-width k , a $(2^{3k+2} - 1)$ -parse tree witnessing clique-width can be found in polynomial time.

In this paper we present an algorithm for \sharp GENSAT for formulas of bounded tree-width k which runs in time $4^k(n + n^2 \cdot \log_2(n))$, where n is the size of the input. The main ingredient of the algorithm is a splitting formula for the number of satisfying assignments for a set of clauses Σ where the incidence graph $I(\Sigma)$ is a union of two graphs G_1 and G_2 with a shared induced subgraph H of size at most k . We also present analogue improvements for algorithms for formulas of bounded clique-width which are given together with their derivation.

This considerably improves results for \sharp SAT, and hence also for SAT, previously obtained by Courcelle et al. [On the fixed parameter complexity of graph enumeration problems definable in monadic second order logic, *Discrete Appl. Math.* 108 (1–2) (2001) 23–52].

© 2007 Published by Elsevier B.V.

Keywords: Satisfiability (SAT); Counting problems; Fixed parameter tractable (FPT); Tree-width; Clique-width

1. Introduction and statement of result

1.1. The problem

We study algorithms for \sharp SAT, and \sharp GENSAT, the problem of computing the number of satisfying assignments of a set of (generalized) propositional clauses Σ . It was shown by Valiant [59] that \sharp SAT is \sharp P-complete. \sharp GENSAT is the counting problem associated with the generalized satisfiability problem introduced by Schaefer [54], who proved

E-mail addresses: eldar@cs.technion.ac.il (E. Fischer), janos@cs.technion.ac.il (J.A. Makowsky).

¹ Partially supported by the VPR-Fund—Dent Charitable Trust—non-military research fund of the Technion—Israel Institute of Technology.

² Partially supported by a Grant of the Fund for Promotion of Research of the Technion-Israel Institute of Technology.

³ Freely collaborating with J.A. Makowsky.

1 also a dichotomy theorem, classifying the problems into polynomially solvable cases and **NP**-complete cases, and
 2 nothing in between. Instances Σ of GENSAT(S) consist of generalized clauses $r_i(\bar{v}) : i \in \mathbb{N}$, where \bar{v} is a vector
 3 of $\rho(i)$ propositional variables. $\rho(i)$ is called the size or arity of r_i . The truth of r_i is given by the truth tables in
 4 $S = \{R_i : i \in \mathbb{N}\}$ over the variables of r_i . In [54] S is assumed to be finite. A dichotomy theorem for \sharp GENSAT was
 5 proven by Creignou and Hermann in [26], where only polynomial time computable and \sharp **P**-complete cases occur. For
 6 a unified treatment of these results, cf. the book by Creignou et al. [27]. For each version of GENSAT, the instances Σ
 7 can be expanded into a set of clauses Σ^{exp} of SAT, such that each satisfying assignment z makes Σ true for GENSAT iff
 8 z makes Σ^{exp} true for SAT. Note, however, that in general Σ^{exp} could be exponentially bigger than Σ . We shall introduce
 9 the formal framework and examples for GENSAT and \sharp GENSAT in Section 2.

10 We associate with Σ three graphs. The graph $SI(\Sigma)$ is a signed bipartite graph with the variables and clauses of Σ
 11 as vertices, indicating whether variables occur positively or negatively in a clause. The graphs $P(\Sigma)$ (the primal graph
 12 of Σ) and $I(\Sigma)$ (the incidence graph of Σ) are unsigned graphs. $P(\Sigma)$ has only the variables as its vertices, and edges
 13 indicate that two variables occur in a common clause. $I(\Sigma)$ is obtained from $SI(\Sigma)$ by omitting the signs on the edges.

14 In the first part of the paper (Sections 2–5) we shall study the complexity of \sharp GENSAT under the assumption⁴ that
 15 the tree-width $tw(P(\Sigma))$ of $P(\Sigma)$ or the tree-width $tw(I(\Sigma))$ of $I(\Sigma)$ is bounded by a fixed number $k \in \mathbb{N}$. The exact
 16 definitions of these graphs and of tree-width are given in Section 3, where we also discuss examples of formulas of
 17 bounded and unbounded tree-width.

Let us note here already the observation of Gottlob and Pichler, [38]:

19 **Proposition 1.1.** *For every generalized clause set Σ we have*

$$tw(I(\Sigma)) \leq tw(P(\Sigma)) + 1.$$

21 It was pointed out in Courcelle et al. [23] that graph counting problems where the objects to be counted are definable
 22 in Monadic Second Order Logic, MSOL,⁵ are solvable in polynomial time when restricted to graphs of tree-width
 23 at most k , for some fixed $k \in \mathbb{N}$. In [23] no estimate of the constants involved is given, but using [44] one can get
 24 estimates which depend on the quantifier rank q of the defining formula and an upper bound of $\exp_q(c \cdot k) \cdot n^3$ with c
 25 small. Here $\exp_1(k)$ is the function 2^k and $\exp_{m+1}(k) = 2^{\exp_m(k)}$.

The method developed in [23] has various applications in the theory of graph polynomials, cf. [43–46].

27 To apply the methods of [23] to \sharp SAT, one notes that SAT is indeed definable in MSOL over $SI(\Sigma)$. A satisfying
 28 assignment can be identified with a subset of vertices V_0 (the variables which are assigned the value `true`), which
 29 has the property that every clause contains a literal $v \in V_0$ or it contains a literal $\neg v$ with $v \in V - V_0$. This is easily
 30 expressible as a formula $\phi_{\text{sat}}(V_0)$ of MSOL of quantifier depth 2. Clearly Σ is satisfiable iff

$$31 \quad SI(\Sigma) \models \exists V_0 \phi_{\text{sat}}(V_0).$$

Furthermore, the function

$$33 \quad \text{csat}(\Sigma) = |\{V_0 \subseteq V : SI(\Sigma), V_0 \models \phi_{\text{sat}}(V_0)\}|$$

counts the number of satisfying assignments.

35 So it follows from [23,44] for \sharp SAT that

37 **Theorem 1.2.** *csat can be computed, and hence also SAT can be solved, in time $O(n^3)$ for sets of clauses Σ with
 $tw(I(\Sigma)) \leq k$, where the constants depend on k only, but are at least doubly exponential in k .*

39 Also in [23,44] it is shown that a similar theorem holds for `csat` with formulas where $SI(\Sigma)$ is of bounded clique-
 width, provided the input is given together with a parse tree of the clique-width. In the second part of the paper (Section
 6) we discuss this extension in greater detail.

⁴ In [27, Chapter 8], other input-restricted satisfaction problems are considered, such as PLANAR-SAT, where $I(\Sigma)$ is assumed to be planar, or
 DENSE-SAT, where the number c of clauses over n variables is $\Omega(n^m)$, where $m = \max_i \{\rho(i)\}$. These restrictions leave the SAT **NP**-complete, but
 make PLANAR-MAX-SAT and DENSE-MAX-SAT approximable with polynomial time approximation schemes (PTAS).

⁵ This holds also if we replace MSOL by CMSOL, where we also allow all the modular counting quantifiers $C_{m,p}x\phi(x)$ which state that the
 number of elements satisfying $\phi(x)$ equals m modulo p .

1 The rather bad estimate of the size of the constants in the algorithms presented in [23,44] is due to the general
 2 character of the method. The proof of Theorem 1.2 uses the Feferman–Vaught theorem for MSOL, and works for
 3 arbitrary counting functions given by MSOL-formulas $\phi(V_0)$ as above. The general running time will be a q -fold
 4 iterated exponential of k , where q is the quantifier depth of ϕ . Grohe and Frick [33], have shown that, unless $\mathbf{P} = \mathbf{NP}$,
 5 this is unavoidable for the general method.

1.2. Main results for bounded tree-width

7 Rather than using the general method of [23,44], we present here a method specially tailored for \sharp SAT, which reduces
 8 the size of the constants to be simply exponential in the tree-width $tw(I(\Sigma))$ of the incidence graph of Σ . We state our
 9 results in a model of computation where arithmetic operations of integers have unit cost. Addition cost of n -bit numbers
 10 in bits is $O(n)$, and of multiplication roughly $O(n \log_2(n))$, so the results have to be modified correspondingly, if the
 11 complexity is to be measured in bits. For details and optimal bounds cf. the classical monograph [1].

12 Our results are stated for given tree-decompositions of the incidence and primal graphs ($I(\Sigma)$ and $P(\Sigma)$) of Σ . There
 13 are algorithms that find a tree decomposition of bounded width, given a graph of treewidth at most some constant k ,
 14 and run in $O(n^2)$ time with constants simply exponential in k , cf. [5]. Proofs of our results are given in Sections 4
 15 and 5.

Theorem 1.3. *Given a k -tree decomposition of $I(\Sigma)$, $\text{csat}(\Sigma)$ can be computed, and hence also SAT can be solved,
 17 if restricted to Σ with $I(\Sigma)$ of tree-width at most k , using $4^k \cdot n$ arithmetic operations (or in time $4^k(n + n^2 \cdot \log_2(n))$
 18 if bit cost is applied).*

19 When considering \sharp GENSAT, Theorem 1.3 can be applied to Σ^{exp} , provided that both the size of Σ^{exp} and the
 20 tree-width $tw(I(\Sigma^{\text{exp}}))$ are polynomially bounded in the size of Σ , respectively, the tree-width of $I(\Sigma)$. For example
 21 this is the case, if the size of each clause is bounded by $\log_2(n)$, where n is the size of Σ . However, there are instances
 22 Σ of size n of GENSAT with $tw(I(\Sigma)) = 1$ and $tw(I(\Sigma^{\text{exp}})) = n$. In contrast we have:

Proposition 1.4. *For every instance Σ for GENSAT(S)*

- 23 (i) $tw(P(\Sigma^{\text{exp}})) = tw(P(\Sigma))$.
 24 (ii) *If the arities $\rho(i)$ of the clauses in S are bounded by m , $tw(I(\Sigma^{\text{exp}})) \leq tw(I(\Sigma)) \cdot m$.*

Using Proposition 1.4 gives immediately our main result for \sharp GENSAT:

27 **Theorem 1.5.** *Given a k_1 -tree decomposition of $P(\Sigma)$, a k_2 -tree decomposition of $I(\Sigma)$, let $m = \max_i\{\rho(i)\}$ (if it
 28 exists), and $k_3 = \max_i\{\rho(i), k_2\}$. Then \sharp GENSAT(S) can be computed*

- 29 (i) *with $4^{k_1} \cdot n^2$ arithmetic operations, provided the size of each clause is bounded by $\log_2(n)$;*
 30 (ii) *with $4^{k_1+m} \cdot n$ arithmetic operations, provided the size of each clause is bounded by $m \in \mathbb{N}$;*
 31 (iii) *with $4^{k_3 \cdot m} \cdot n$ arithmetic operations, provided the size of each clause is bounded by $m \in \mathbb{N}$.*

This includes the classical cases \sharp NOT-ALL-EQUAL 3SAT and \sharp ONE-IN-THREE 3SAT of [35, Problem list A9].

33 We could also apply Theorem 1.2 to \sharp GENSAT, but not all versions of GENSAT are MSOL-definable, for example,
 34 HALFSAT, where we require that in each clause at least half of the literals are true. Let $\text{HALFSAT}_{f(n)}$ be like HALFSAT
 35 but with the size of the clauses bounded by $f(n)$, a function of the input size. If f is the constant function $b \in \mathbb{N}$, we
 36 write also HALFSAT_b . HALFSAT_b is MSOL-definable by a formula ϕ_b , but the quantifier rank $q = qr(\phi_b) \geq b + 2$.
 37 Theorem 1.5 includes also cases not covered by Theorem 1.2:

Corollary 1.6. *For every $f(n)$ the problem \sharp HALFSAT $_{f(n)}$ restricted to instances Σ with $tw(P(\Sigma)) \leq k$ can be com-
 39 puted with $4^k \cdot 2^{2 \cdot f(n)} \cdot n$ arithmetic operations.*

41 Finally, using the self-reducibility of SAT, cf. [53, Example 10.3, p. 228], we get also a generating algorithm with
 polynomial delay in the sense of [41]. These are algorithms which enumerate all instances of a problem where the

1 time elapsing between two such instances is polynomial in the size of the problem. Clearly, this allows an exponential
 2 number of instances to be produced.

3 In our situation we have:

4 **Corollary 1.7.** *Under the assumptions of Theorem 1.5, GENSAT, restricted to instances Σ with $tw(P(\Sigma)) \leq k$, has a
 5 generating algorithm with polynomial delay.*

1.3. Main results for bounded clique-width

7 The notion of clique-width was introduced in [21] and studied more systematically in [20,18,31,25]. In the last 10
 8 years, the study of graphs of bounded clique-width became very popular, cf. the work of A. Brandstaedt, B. Courcelle,
 9 V.V. Lozin, P. Seymour, J. Spinrad, and their many collaborators. Clique-width is a more general notion than tree-
 10 width and measures somehow how a graph can be built from smaller graphs by remembering only that certain nodes
 11 are coloured and the number of colours is fixed. The main difference is the important role of the complete bipartite
 12 subgraphs. If large bipartite subgraphs are excluded, then bounded clique-width yields bounded tree-width, cf. [19].
 13 Courcelle and Olariu in [25] showed that clique-width of graphs of tree-width k , is at most $2^{k+1} + 1$. Therefore,
 14 any class of graphs of bounded tree-width, is automatically of bounded clique-width. Moreover, Courcelle et al. [21]
 15 provided a complicated proof that any given context-free graph grammar based on vertex-replacement (Confluent NCE,
 16 or context-free VR grammar) generates graphs of bounded clique-width. Although an upper bound for the clique-width
 17 could be derived from their proof, it is not straightforward. In general, finding an explicit bound for the clique-width is
 18 a more complicated task than finding a bound for the tree-width. For explicit computations of clique-width, cf. [37,36].
 19 In contrast to tree-width, there is also a natural notion of clique-width for directed graphs or signed graphs, which is
 20 different from the undirected (unsigned) case. Recall that we denote by $SI(\Sigma)$ the signed version of the incidence graph
 21 of Σ where edges are labelled depending whether the variable occurs positively or negatively in a clause.

22 To get an analogue of Theorem 1.5 one needs a parse tree of the graph with respect to its clique-width. We denote
 23 by $der_{SI}(\Sigma)$ or $der_I(\Sigma)$ such a parse tree for the signed, respectively, unsigned case. Details are given in Section 6.

24 By a recent result of Oum and Seymour [51], described in more detail in Section 6, Theorem 6.2, this can be achieved
 25 in the following way, which suffices for our purposes. There is a function f , such that, for given k , there is a polynomial
 26 time algorithm that, with input a graph G , either concludes that its clique-width is larger than k or outputs an $f(k)$ -parse
 27 tree for G . By a straight inspection of their proof a similar theorem can be proven also for the clique-width of signed
 28 graphs where $f(k)$ is replaced by a function $g(k)$ of the same order of growth.

29 Using the parse tree obtained from the signed version of this theorem, we can now apply our result.

30 **Theorem 1.8.** *Given a set of clauses Σ and a signed parse tree $der_{SI}(\Sigma)$ for clique-width of up to k , it is possible to
 31 calculate $csat(\Sigma)$, with a number of algebraic operations that is linear in the size of the parse tree $der_{SI}(\Sigma)$, and simply
 32 exponential in k .*

33 This theorem can also be extended to solve \sharp GENSAT, but we leave this to the reader. We also believe that a
 34 corresponding theorem for unsigned clique-width is true, but we did not work out the details for this paper.

35 1.4. Significance and applicability of the results

36 As pointed out by Downey and Fellows in [30] there is a long way to go from establishing that a problem is
 37 fixed parameter tractable, FPT, to feasible algorithms. In [23], it was first established that MSOL-definable counting
 38 problems are FPT, with constants being multiply exponential with tree-width k as the parameter k . We make the
 39 following significant improvements:

- 40 • In the case of SAT and \sharp SAT with tree-width k as parameter we bring the constants down to being simply
 41 exponential in k .
- 42 • In the case of SAT and \sharp SAT with clique-width k as parameter we also bring the constants down to being simply
 43 exponential in $g(k)$. We shall discuss in Section 6, how this can be further improved to be simply exponential
 in k .

- We show many versions of GENSAT and \sharp GENSAT to be FPT with the same parameter k and the constants simply exponential in k , even when they are not MSOL-definable.

In industrial applications of hardware and software verification, the problem is often presented in two steps. First, a labelled graph G is built for which a property Φ has to be verified. The labelled graph was generated by some graph grammar which takes into account that only a fixed number of labels are used and reflects the modularity of the hardware design or the well-structured character of the software, cf. [56]. As a result of this, cf. [36], the graphs are *a priori* of bounded tree-width or clique-width, depending on the particular grammar only. The tree-decompositions, respectively, the parse tree of the clique-width, can be explicitly computed from the parse tree in the graph grammar. In real-life applications of hardware verification, related methods using tree-width have been successfully implemented, cf. [11,60], and the references therein.

In a second step the verification of Φ on G is translated uniformly into an instance of SAT. If the latter translation can be expressed as an MSOL-transduction, it was shown by Courcelle and Engelfriet, cf. [20,18,31], that the resulting instance Σ of SAT has an incidence graph, the clique-width of which depends only on the tree-width or clique-width of G and Φ . For a detailed exposition, cf. [22]. It remains to be explored in detail, in which concrete situations this can be used.

In applications in Artificial Intelligence very large sets of clauses (rules and facts) have to be tested for satisfiability. But the clauses are often naturally partitioned into sets coming from different domains of discourse, where the shared variables are few. Amir has explored this in great detail [4]. In the course of his work he has shown that partitioning sets of clauses in this way is related to the tree-width of the clause graph $P(\Sigma)$. Low tree-width gives good partitions, and partitions with cyclefree overlapping of the variables give also tree-decompositions of low tree-width. To quote from [4, Section 5.2, p. 90]:

We believe that in domains that deal with engineered physical systems, many of the domain axiomatizations have these structural properties. Indeed, design of engineering artefacts encourages modularization with minimal interconnectivity, see [3,42,16]. More generally, we believe axiomatizers of large corpora of real-world knowledge tend to try to provide structured representations following some of these principles. Recent experiments with the HPKB knowledge base of SRI and a part of the Cyc knowledge base support this belief. Those experiments are reported in [4, Section 5.8].

So tree-width and clique-width turn out to be natural concepts in industrial applications of SAT, both in verification of software and hardware, and in automated reasoning.

1.5. Methods

The main ingredient of the algorithm is a Feferman–Vaught-type theorem, cf. [44], in form of a splitting formula for the number of satisfying assignments for a set of non-generalized clauses Σ where the incidence graph $I(\Sigma)$ is a union of two graphs G_1 and G_2 with a shared induced subgraph H of size at most k . This is given as Theorem 4.7 in Section 4. Such splitting formulas are well known for graph polynomials for $k = 0, 1$. In the case of H consisting of the empty set or only one vertex, many graph polynomials are multiplicative, e.g., the Tutte polynomial, the matching polynomials and others, cf. [14,44]. In the case of H consisting of two vertices, such a splitting formula was proven by Oxley and Welsh [52] for the Tutte polynomial. For H of arbitrary fixed size k , splitting formulas were established by Negami [49], Andrzejak [8], Noble [50] and Traldi [57] for various versions of the Tutte polynomial. In [44] a general existence theorem for such splitting formulas is given. Theorem 1.3 is the result of searching for a splitting formula for the function csat .

1.6. Related work

The study of \sharp SAT on formulas with $I(\Sigma)$ of bounded tree-width and clique-width was initiated in [23]. SAT on various presentations of the clauses as graphs and restricted to inputs of tree-width at most k was previously studied, among others, by Dechter and Pearl [28] and Feder and Vardi [32]. More recent work was presented by Gottlob and Pichler [38], Amir and McIlraith [7,6], Alekhnovich and Razborov [2], and Szeider [55].

1 Most of the previous results are stated for $P(\Sigma)$ having bounded tree-width. In the case of [2] branch-width of the
 3 clause hypergraph is studied. Here the vertices are the variables, and the hyperedges are the clauses as sets of variables
 (disregarding negations). Our results are in general much stronger, as we only require that the tree-width of $I(\Sigma)$ or
 the clique-width of $SI(\Sigma)$ is bounded.

5 1.7. Outline of the paper

In Section 2 we define the general framework of the satisfiability problems which we consider. In Section 3 we give
 7 the necessary background concerning tree-width and clause graphs $P(\Sigma)$ and $I(\Sigma)$. In Section 4 we derive the splitting
 formula which allows us to count satisfying assignments for H -sums of instances of SAT. This is one of the main new
 9 algorithmic ingredients of the paper. In Section 5 we prove the main theorems for the case of bounded tree-width. In
 Section 6 we give the necessary background concerning clique-width and extend the results to the case of bounded
 11 clique-width. In Section 7, finally, we draw some conclusions and discuss further research.

2. Generalized satisfiability

13 We follow closely [54,26].

Let $S = \{R_i : i \in \mathbb{N}\}$ be an infinite set of logical relations of rank $\rho(i)$. A logical relation R_i of rank $\rho(i)$ is a
 15 subset of $\{0, 1\}^{\rho(i)}$. An S -formula Σ is a set of (generalized) clauses of the form $r_i(\bar{v})$ where $\bar{v} = v_{j_1}, \dots, v_{j_{\rho(i)}}$ are any
 propositional variables.

17 The size of an S -formula Σ is the sum of the sizes of its generalized clauses, irrespective of the choice of S . We
 denote the set of propositional variables by Var and the set of variables occurring in Σ by $\text{Var}(\Sigma)$.

19 The S -satisfiability decision problem $\text{GENSAT}(S)$ is the problem of deciding whether for a given S -formula Σ there
 is an assignment $z : \text{Var} \rightarrow \{0, 1\}$ such that for each clause $r_i(\bar{v})$ in Σ , $z(\bar{v}) \in R_i$, i.e., all clauses are simultaneously
 21 satisfiable using the semantics given by the S . The S -satisfiability counting problem $\sharp\text{GENSAT}(S)$ counts the number
 of satisfying assignments for Σ . If S is not explicitly mentioned we speak of an instance of GENSAT or $\sharp\text{GENSAT}$
 23 rather than of $\text{GENSAT}(S)$, respectively, $\sharp\text{GENSAT}(S)$.

The classical satisfiability problem SAT usually is formulated with literals rather than variables only. When formu-
 25 lating SAT as an instance of GENSAT this amounts to having different r_i 's for each distribution of the negation symbols
 among the literals. If the size of the clauses is bounded by a fixed number then S can be assumed finite.

27 All instances of $\text{GENSAT}(S)$ are in \mathbf{NP} and all instances of $\sharp\text{GENSAT}(S)$ are in $\sharp\mathbf{P}$.

Schaefer et al., cf. [54,26], give a complete classification for which the corresponding instances given by S are
 29 \mathbf{NP} -hard, respectively, $\sharp\mathbf{P}$ -hard. They prove a Dichotomy Theorem which states that all the other cases are solvable in
 polynomial time.

31 For our purpose it suffices to note that if $\text{GENSAT}(S)$ is \mathbf{NP} -complete, then $\sharp\text{GENSAT}(S)$ is $\sharp\mathbf{P}$ -complete. In other
 words, there is an abundance of $\sharp\mathbf{P}$ -complete instances of GENSAT .

33 3. Tree-width of clause graphs and H -sums

We assume the reader is familiar with some basic graph theory and the notion of a graph minor. A graph H is a minor
 35 of a graph G if H can be obtained from G by deleting or contracting edges and deleting vertices. General background
 on minors and tree-width may be found in [29].

37 3.1. Tree-width of graphs

Definition 3.1. A k -tree decomposition of G is given as follows:

- 41 (i) We have a (not necessarily rooted) tree $\mathcal{T} = \langle T, f \rangle$, where T is a set and f is a function mapping nodes onto their
 father.
- (ii) The vertex set $V(G)$ of the graph is the union of sets A_t , with $t \in T$ and $|A_t| \leq k + 1$.
- 43 (iii) For every edge $e = (x, y) \in E(G)$ there is a $t \in T$ such that $x, y \in A_t$.
- (iv) For each $x \in V$ the set $T(x) = \{t \in T : x \in A_t\}$ is a connected subgraph of \mathcal{T} (Fig. 1).

45 If the tree \mathcal{T} is a path (no branching) we speak of a k -path decomposition.

Q1

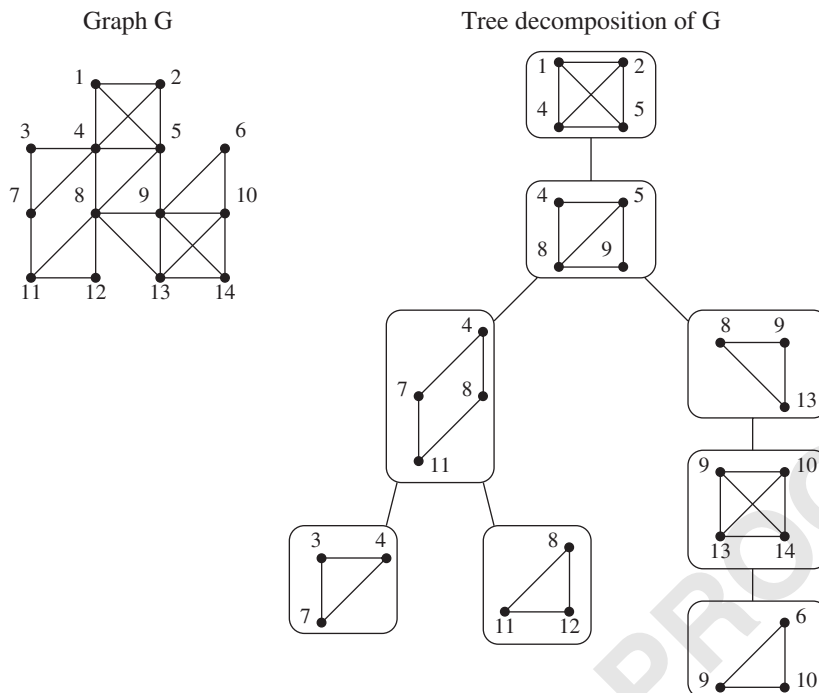


Fig. 1. Example of a 3-tree decomposition of a graph.

1 **Remark 3.2.** Under conditions (i)–(iv) is equivalent to: For every connected subgraph H of G , the set $\{t \in T : V(H) \cap A_t \neq \emptyset\}$ is a connected subtree of \mathcal{T} .

3 **Definition 3.3.** (i) G is of tree-width (resp. path-width) at most k , if there exists a k -tree decomposition (resp. k -path decomposition) of G . In the literature such graphs are sometimes also called *partial k -trees*.

5 (ii) The tree-width (or path-width) of a signed (edge coloured) graph is, by definition, the same as its tree-width without the colouring.

7 Given a graph G , finding its tree-width is NP-complete, cf. [9], but for fixed k , checking whether G has tree-width at most k (and if so, finding a witnessing tree decomposition), can be done in polynomial time, cf. [13]. For the most
9 advanced approximation algorithms to compute the tree-width, cf. [5].

Example 3.4. (i) The tree-width of a tree is 1.

11 (ii) The tree-width of C_n , the cycle with n vertices, is 2.

13 (iii) The tree-width of K_n , the complete graph on n vertices, is $n - 1$, and of $K_{n,n}$, the complete bipartite graph on twice n vertices, is n .

(iv) The tree-width of the two-dimensional square grid $Grid_{n,n}$ on n^2 vertices is n .

15 3.2. Tree-width of clause graphs

The incidence graph $I(\Sigma)$ of an S -formula Σ with variable set $Var(\Sigma)$ is the bipartite simple graph $I(\Sigma) = (\Sigma, Var(\Sigma), E_I)$ where for each generalized clause $C = r_i(\bar{v}) \in \Sigma$ we have that $(v, C) \in E_I$ iff $v \in C$.

17 The primal graph $P(\Sigma)$ of an S -formula Σ is the simple graph $P(\Sigma) = (Var(\Sigma), E_P)$ where for each $v_1, v_2 \in Var(\Sigma)$
19 the pair $(v_1, v_2) \in E_P$ iff there is a clause $C \in \Sigma$ where both v_1 and v_2 occur.

21 Recall that Proposition 1.1 in the Introduction stated that for every generalized clause set Σ we have $tw(I(\Sigma)) \leq tw(P(\Sigma)) + 1$.

1 A natural example of formulas of bounded tree-width can be obtained as follows. Let $V = \{v_0, v_1, \dots, v_m\}$ be a set
of propositional variables and $\Sigma = \{C_0, C_1, \dots, C_n\}$ a set of clauses over V .

3 **Proposition 3.5.** *Assume that there is $d \in \mathbb{N}$ such that, if v_i or $\neg v_i$ occurs in C_j , then $|i - j| \leq d$. Then $I(\Sigma)$ has
path-width at most d .*

5 This example is related to the cut-width of the hypergraph representing the clauses and has been used successfully
in very large real-life applications, cf. [60].

7 3.3. Tree-width of Horn, pigeon-hole and Tseitin formulas

To illustrate the concept of the tree-width of formulas, we look at three classical examples: Horn formulas, Tseitin
9 formulas and pigeon-hole formulas.

Horn clauses are clauses with at most one literal non-negated. Checking satisfiability of Horn formulas can be done
11 in linear time [40].

Proposition 3.6. *The tree-width of Horn-formulas is unbounded.*

13 **Proof.** Take the grid $Grid_{2n, 2n}$. It is bipartite, with equal number of variables and clauses. Each clause contains at most
four variables. For each clause we choose r_i to be a disjunction where exactly one variable occurs positively. This gives
15 us Σ with $Grid_{2n, 2n}$ as its underlying graph. Hence its tree-width is $2n$. \square

Tseitin showed that the difficulty of proving inconsistency of the Tseitin formulas $\Sigma(H)$ using regular resolution
17 only depends on the properties of the underlying graph H viewed as an expander graph, cf. [58,34]. The regularity
assumption for resolution was later removed by Haken. Haken also showed that the formulas PHP_n^{n+1} have long proofs
19 of inconsistency using resolution, cf. [39,15]. This is no accident, as it is shown by Alekhovich and Razborov, that
for sets of inconsistent clauses Σ with $tw(I(\Sigma))$ bounded by k , there are resolution proofs of polynomial length [2].
21 Atserias and Dalmau [10] give further interpretations of this phenomenon.

We now compute the tree-width of $I(\Sigma)$ for these examples of sets of (non-generalized) clauses which are natural or
23 occur in the literature. When no proofs are given, it is straightforward to verify the statements. The tree-width of $P(\Sigma)$
can be easily estimated using Proposition 1.1.

25 Tseitin formulas are formulas obtained as follows: Let $H = (V, E)$ be a graph. Let $\mu : V \rightarrow \{0, 1\}$ be a marking of
the vertices, with $\sum_{v \in V} \mu(v) = 1 \pmod{2}$. We define $\Sigma(H, \mu)$ in the following way: The variables of the formula are
27 represented by the edges in E , whereas the formula is the conjunction of all the clauses $F_v, v \in V$, where

$$F_v = \begin{cases} e_1(v) \oplus \dots \oplus e_d(v) & \text{if } \mu(v) = 1, \\ \neg(e_1(v) \oplus \dots \oplus e_d(v)) & \text{if } \mu(v) = 0 \end{cases}$$

29 and $e_1(v), \dots, e_d(v)$ are the edges incident with v . It is straightforward to bring this into clausal form, which we denote
by $T(H, \mu)$.

31 **Proposition 3.7.** *The tree-width of the Tseitin formulas $T(H, \mu)$ is at least as big as the tree-width of H .*

Proof. One can show that H is a minor of $I(T(H, \mu))$. \square

33 The pigeon-hole formulas PHP_n^{n+1} are defined as follows. We have variables $p_{i,j}, a_i, b_{i,j,k}$ for $i = 1, \dots, n+1$ and
 $k, j = 1, \dots, n$. $p_{i,j}$ stands for “pigeon i sits in hole j ”. a_i stands for “pigeon i sits in one of the holes”. $b_{i,j,k}$ stands
35 for “pigeon i and j sit both in hole k ”.

$$PHP_n^{n+1} = \bigwedge_{i=1}^{n+1} \bigvee_{j=1}^n p_{i,j} \rightarrow \bigvee_{k=1}^n \bigvee_{j=1, i \neq j}^{n+1} (p_{i,k} \wedge p_{j,k}).$$

1 We use the additional variables to write it in readable clausal form. We write a_i for $\bigvee_{j=1}^n p_{i,j}$, and $b_{i,j,k}$ for $(p_{i,k} \wedge p_{j,k})$.
 This gives

$$3 \quad \bigvee_{i=1}^{n+1} a_i \vee \bigvee_{k=1}^n \bigvee_{i,j=1, i \neq j}^{n+1} b_{i,j,k}.$$

We also add the clauses

$$5 \quad A_i = a_i \leftrightarrow \bigwedge_{j=1}^n \neg p_{i,j} \quad \text{and} \quad B_{i,j,k} = b_{i,j,k} \leftrightarrow (p_{i,k} \wedge p_{j,k}).$$

Proposition 3.8. *The tree-width of the pigeon-hole formulas PHP_n^{n+1} is at least n .*

7 **Proof.** The grids $\text{Grid}_{n,n}$ are minors of $G_{\text{PHP}_n^{n+1}}$. \square

3.4. H -sums of graphs

9 Given a k -tree decomposition of a graph G with tree \mathcal{T} and sets of vertices $A_t, t \in \mathcal{T}$, we denote by H_t the induced subgraph of G with vertex set A_t . Given the k -tree decomposition and all the induced subgraphs H_t , we can reconstruct the original graph G using successive (almost disjoint) unions. To make this precise we define the H -sum of two graphs.

11 Given two graphs G_1, G_2 with distinguished induced subgraphs H_1, H_2 which are isomorphic to H with isomorphisms h_1, h_2 , the H -sum of G_1 and G_2 is an almost disjoint union of the two graphs where the intersection contains exactly H as induced subgraph (using the isomorphisms h_1 and h_2 to fix it).⁶ In other words:

15 **Definition 3.9.** (i) For $i = 1, 2$ let $G_i = \langle V(G_i), E(G_i) \rangle$ and $V(G_1) \cap V(G_2) = V(H)$ and $E(H) = E(G_1) \cap V(H)^2 = E(G_2) \cap V(H)^2$. Then $G = G_1 \oplus_H G_2$ is given by $V(G) = V(G_1) \cup V(G_2)$ and $E(G) = E(G_1) \cup E(G_2)$.

17 (ii) H -sums of edge and vertex coloured graphs are defined similarly.

19 In the reconstruction process of G from \mathcal{T} and the H_t 's we have to perform a sequence of H -sums where H is always an induced subgraph of the H_t 's.

4. A splitting formula for H -sums of clause graphs

21 In this section all clauses are non-generalized. Let Σ be a set of clauses over a variable set V , and let $W \subseteq V$ and $z : W \rightarrow \{0, 1\}$ be a partial assignment. We denote by $\Sigma^{(z)}$ the set of clauses obtained from Σ by performing the substitution

$$s(v) = \begin{cases} \text{true} & \text{if } z(v) = 1, \\ \text{false} & \text{if } z(v) = 0. \end{cases}$$

25 Similarly, we denote by $\text{csat}_z(\Sigma)$ the number of assignments z' with $z'|_W = z$ which make Σ true.

As any k -tree decomposition of $I(\Sigma)$ gives also a k -tree decomposition of $I(\Sigma^{(z)})$, clearly we have

27 **Lemma 4.1.** (i) $\text{csat}(\Sigma^{(z)}) = \text{csat}_z(\Sigma)$.

(ii) $\text{tw}(\Sigma^{(z)}) \leq \text{tw}(\Sigma)$.

29 The following is a straightforward consequence of our notation.

Lemma 4.2. *With the notation from above we have*

$$31 \quad \text{csat}(\Sigma) = \sum_{z:W \rightarrow \{0,1\}} \text{csat}(\Sigma^{(z)}) = \sum_{z:W \rightarrow \{0,1\}} \text{csat}_z(\Sigma).$$

⁶ Strictly speaking we should write $G_1 \oplus_{H, h_1, h_2} G_2$, but we shall drop the isomorphisms when there is no risk of confusion.

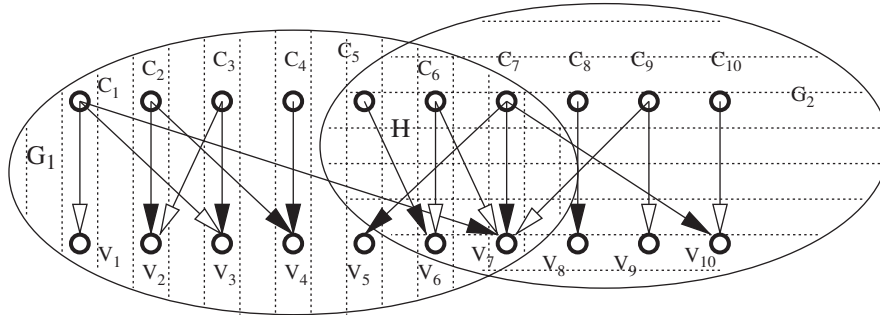


Fig. 2. H -sum of G_1 and G_2 where H contains both clauses and variables.

1 We now derive our splitting formula for the function csat for H -sums.

4.1. H -sums of incidence graphs $I(\Sigma)$

3 From now on, let Σ be a set of clauses, such that the corresponding incidence graph $G = I(\Sigma)$ is the H -sum $G_1 \oplus_H G_2$. We denote by Σ_i the set of clauses with $I(\Sigma_i) = G_i$, cf. Fig. 2.

5 We distinguish two extreme cases.

4.1.1. H contains only variables

If H contains only variables $W \subseteq V$, we can divide the clauses of Σ into four sets:

- 7 (i) Σ_i^{V-W} , $i = 1, 2$ which do not contain variables from W and such that the clauses are vertices in G_i and
- 9 (ii) Σ_i^W , $i = 1, 2$ which do contain variables from W and such that the clauses are vertices in G_i .

Clearly, $\Sigma_i^{V-W} \cup \Sigma_i^W = \Sigma_i$.

11 Using Lemma 4.2 we get immediately:

Lemma 4.3. *With the notation from above we have*

$$\begin{aligned} \text{csat}(\Sigma_1 \oplus_H \Sigma_2) &= \text{csat}(\Sigma) = \sum_{z:W \rightarrow \{0,1\}} \text{csat}_z(\Sigma^{(z)}) \\ &= \sum_{z:W \rightarrow \{0,1\}} \text{csat}_z(\Sigma_1^{(z)}) \cdot \text{csat}_z(\Sigma_2^{(z)}). \end{aligned}$$

4.1.2. H contains only clauses

15 Let $\Delta = \{D_1, \dots, D_m\}$ be the clauses in H . We write each of those as $D_i^1 \vee D_i^2$ with D_i^j containing only variables from G_j . Again, for $i = 1, 2$, let Σ_i be the set of clauses with $I(\Sigma_i) = G_i$.

17 **Lemma 4.4.** *Let $m = 1$. Then*

$$\text{csat}(\Sigma) = \text{csat}(\Sigma_1 - \{D_1^1\}) \cdot \text{csat}(\Sigma_2) + \text{csat}(\Sigma_1) \cdot \text{csat}(\Sigma_2 - \{D_1^2\}) - \text{csat}(\Sigma_1) \cdot \text{csat}(\Sigma_2).$$

19 **Proof.** Straightforward from the inclusion and exclusion principle. \square

The case $m \geq 2$ is based on the inclusion/exclusion principle. We need some notation. Let $[m] = \{1, \dots, m\}$. For $X \subseteq [m]$ and $i = 1, 2$ denote by $S_i(X) = \{D_i^j : j \in X\}$.

Lemma 4.5. *The tree-width of $\Sigma_i - S_i(X)$ is not bigger than the tree-width of Σ_i .*

1 One now proves by induction, using Lemma 4.4 as the basis:

Lemma 4.6. *With the notation from above and*

$$3 \quad B_i(X_i) = \text{csat}((\Sigma_i - S_i(X_i)))$$

we have

$$5 \quad \text{csat}(\Sigma) = \text{csat}(\Sigma_1 \oplus_{\Delta} \Sigma_2) = \sum_{k=1}^m (-1)^k \sum_{|X_1 \cap X_2|=k} B_1(X_1) \cdot B_2(X_2).$$

4.1.3. *The mixed case*

7 For the mixed case we assume that H is a signed bipartite graph with W as its variable nodes and Δ as its clause nodes.

9 **Theorem 4.7.** *With the notation from above, let $\Sigma = \Sigma_1 \oplus_{W, \Delta} \Sigma_2$ and*

$$B_i(X_i)^{(z)} = \text{csat}((\Sigma_i - S_i(X_i))^{(z)}).$$

11 *Then*

$$\text{csat}(\Sigma) = \text{csat}(\Sigma_1 \oplus_{W, \Delta} \Sigma_2) = \sum_{z: W \rightarrow \{0,1\}} \sum_{k=1}^m (-1)^k \sum_{|X_1 \cap X_2|=k} B_1(X_1)^{(z)} \cdot B_2(X_2)^{(z)}.$$

13 *Hence, computing Σ needs at most $2^{|W|} \cdot 4^m \leq 4^{tw(I(\Sigma))}$ additions and an equal number of multiplications.*

Proof. Apply the inclusion/exclusion principle to $B_i(X_i)^{(z)}$. \square

15 5. Proofs of Theorems 1.3 and 1.5

17 5.1. Proof of Theorem 1.3

19 **Theorem 1.3.** *Given a k -tree decomposition of $I(\Sigma)$, $\text{csat}(\Sigma)$ can be computed, and hence also SAT can be solved, if restricted to Σ with $I(\Sigma)$ of tree-width at most k , using $4^k \cdot n$ arithmetic operations (or in time $4^k(n + n^2 \cdot \log_2(n))$ if bit cost is applied).*

21 **Proof.** We use a dynamic programming approach. We start from the leaves. Let n be the number of nodes of G_{Σ} . Using the k -tree decomposition of G_{Σ} and the induced subgraphs G_t we know how to reconstruct G_{Σ} , starting with small graphs (of size at most $k + 1$) and then using H -sums where H is of size at most k . In each step where an H -sum is performed we use Theorem 4.7.

23 For this we have to compute $2^{|W|} \cdot 4^{|\Delta|} \leq 4^k$ many times products of $\text{csat}((\Sigma_i - S_i(X_i))^{(z)})$, where $(\Sigma_i - S_i(X_i))^{(z)}$ has again tree-width at most k . This uses at most $4^k \cdot n$ additions and multiplications over \mathbb{Z} . As the number of assignments is bound by 2^n the bit size of the numbers involved is at most n . Multiplication of n -bit numbers uses no more than $n \cdot \log_2(n)$ bit-operations. Hence we get an algorithm which runs in time $4^k(n + n^2 \cdot \log_2(n))$ on a Turing machine.⁷ \square

31 5.2. Proof of Proposition 1.4

Proposition 1.4. *For every instance Σ for GENSAT(S)*

- 33 (i) $tw(P(\Sigma^{\text{exp}})) = tw(P(\Sigma))$.
(ii) *If the arities $\rho(i)$ of the clauses in S are bounded by m , $tw(I(\Sigma^{\text{exp}})) \leq tw(I(\Sigma)) \cdot m$.*

⁷ A closer computation actually gives $3^{k+1}(n + n^2 \cdot \log_2(n))$.

1 **Proof.** For an S -formula Σ we define a set of non-generalized clauses Σ^{exp} as follows: Let $R_i \in S$ and $r_i(\bar{v})$ be a
 3 corresponding generalized clause. Denote by $\bar{r}_i(\bar{v})$ the formula in conjunctive normal form representing R_i with the
 appropriate variables. Then

$$\Sigma^{\text{exp}} = \{\bar{r}_i(\bar{v}) : r_i(\bar{v}) \in \Sigma\}.$$

5 To prove (i), we observe that $P(\Sigma^{\text{exp}})$ is the same graph $P(\Sigma)$. To prove (ii), we show how given a k -tree for $I(\Sigma)$ we
 can construct an mk -tree for $I(\Sigma^{\text{exp}})$. We go over the k -tree of $I(\Sigma)$, and in the first stage in every set A_t of the tree we
 7 replace every clause vertex with all its incident variable vertices.

At this stage, the new tree clearly has a bound of mk on its set sizes, and still satisfies the connectivity condition
 9 for every variable vertex (the tree does not contain any clause vertex at this stage). It is also easy to see that for every
 clause of Σ , and hence of Σ^{exp} , there is a set A_t of the tree containing all of its incident variables (just take any set that
 11 in the original tree contained the appropriate clause vertex).

We finish the construction by adding a new leaf for every clause of Σ^{exp} with a set that contains the appropriate
 13 clause vertex and all incident variable vertices, connecting this leaf to the appropriate A_t that contains all variable
 vertices. \square

15 5.3. Proof of Theorem 1.5

17 **Theorem 1.5.** Given a k_1 -tree decomposition of $P(\Sigma)$, a k_2 -tree decomposition of $I(\Sigma)$, let $m = \max_i\{\rho(i)\}$ (if it
 exists), and $k_3 = \max_i\{\rho(i), k_2\}$. Then $\sharp\text{GENSAT}(S)$ can be computed

- 19 (i) with $4^{k_1} \cdot n^2$ arithmetic operations, provided the size of each clause is bounded by $\log_2(n)$;
 21 (ii) with $4^{k_1+m} \cdot n$ arithmetic operations, provided the size of each clause is bounded by $m \in \mathbb{N}$;
 (iii) with $4^{k_3 \cdot m} \cdot n$ arithmetic operations, provided the size of each clause is bounded by $m \in \mathbb{N}$.

Proof. Instead of solving $\sharp\text{GENSAT}(S)$ with input Σ we reduce it to computing $\text{csat}(\Sigma^{\text{exp}})$. According to Proposition
 23 1.4 the reduction does not increase the tree-width of the primal graph. It also increases the tree-width of the incidence
 graph by at most m , provided that every $\rho(i)$ is bounded by m . Hence we only have to make sure that the size of Σ^{exp}
 25 is bounded. But in Σ^{exp} each clause C of Σ with $\rho(i)$ many variables is replaced by at most $2^{\rho(i)}$ many clauses of size
 at most $\rho(i)$.

27 The remaining computations for the estimates in (i)–(iv) are left to the reader. \square

6. The case of bounded clique-width

29 6.1. Background on clique-width

The notion of clique-width was introduced in [21] and studied more systematically in [24,20,18,31,25,37]. In
 31 the last 10 years, the study of graphs of bounded clique-width became very popular, cf. the work of A. Brand-
 staedt, B. Courcelle, V.V. Lozin, P. Seymour, J. Spinrad, and their many collaborators. Courcelle and Olariu in
 33 [25] showed that clique-width of graphs of tree-width k , is at most $2^{k+1} + 1$. Therefore, any class of graphs of
 bounded tree-width is automatically of bounded clique-width. Moreover, Courcelle et al. in [21] provided a com-
 35 plicated proof that any given context-free graph grammar based on vertex-replacement (confluent *NCE*, or context-
 free VR grammar) generates graphs of bounded clique-width. Although an upper bound for the clique-width could
 37 be derived from their proof, it is not straightforward. In general, finding an explicit bound for the clique-width
 is a more complicated task than finding a bound for the tree-width. For explicit computations of clique-width,
 39 cf. [37,36].

Courcelle and Olariu in [25] study two versions of clique-width, for undirected and for directed graphs. We give here
 41 a version for directed or signed graphs where additionally the bipartite character of the graphs is taken into account.
 We identify a SAT formula Σ with the bipartite graph $SI(\Sigma)$ that has edges ‘signed’ with ‘+’ and ‘−’ according
 43 to which variables appear in a clause and whether they are negated. If we drop the signing of the edges, we just
 get $I(\Sigma)$.

1 **Definition 6.1.** The set of SAT formulas of clique width up to k is defined as the set of formulas that can be obtained by
 the following operations over such graphs whose vertices are coloured by $\{1, \dots, k\}$, starting with singletons (formulas
 3 consisting of a single “clause” or “variable” vertex with some colour from $\{1, \dots, k\}$ and no edges).

- (i) Disjoint union.
 5 (ii) Recolouring: For a vertex-coloured edge-signed bipartite graph I , we define $\rho_{i,j}(I)$ to be the graph that results by
 recolouring with j all vertices that were previously coloured with i .
 7 (iii) Positive edge creation: For a vertex-coloured edge-signed bipartite graph I , we define $\eta_{i,j}^+(I)$ to be the graph that
 results from connecting all clause-vertices coloured with i to all variable-vertices coloured with j , with edges
 9 signed by ‘+’. We do not add edges between variable-vertices coloured i and clause-vertices coloured j , or any
 other vertices.
 11 (iv) Negative edge creation: Similarly to the above, we define $\eta_{i,j}^-(I)$ to be the graph resulting from connecting all
 clause-vertices coloured with i to all variable-vertices coloured with j , with edges signed by ‘-’.
 13 (v) In the case of unsigned edges, and without distinguishing clause-vertices and variable-vertices, there is just one
 operation $\eta_{i,j}$ for each $i \neq j$. This corresponds to the original definition in [25].
 15 (vi) The clique-width of a (signed, bipartite) graph is the minimum k such that it has clique-width at most k . We denote
 by $scw(SI(\Sigma))$ the signed bipartite clique-width of $SI(\Sigma)$ and by $cw(I(\Sigma))$ the unsigned clique-width of $I(\Sigma)$.

17 A parse tree der_{SI} for the signed clique-width of a formula Σ is just the rooted tree whose leaves hold singleton
 graphs, whose internal vertices are coloured with the operations of the definitions above (so a vertex corresponding to
 19 a disjoint union has two children, and vertices corresponding to other operations have one child), and whose root holds
 the graph $SI(\Sigma)$ (with any vertex colouring). A parse tree der_I for the clique-width of a formula Σ is defined similarly
 21 for the case of the unsigned graph $I(\Sigma)$.

Every graph G of size n has clique-width $cw(G)$ at most n . The simplest class of graphs of unbounded tree-width
 23 but of clique-width at most 2 are the cliques. To see this assume we have two colours red (1) and blue (2). We start with
 a red singleton and a blue singleton and connect using $\eta_{1,2}$, then we recolour all points red, add a new blue singleton
 25 and connect again using $\eta_{1,2}$, and so forth.

Given a graph G and $k \in \mathbb{N}$, determining whether G has clique-width k is in **NP**. A polynomial time algorithm was
 27 presented for $k \leq 3$ in [17]. It remains open whether for some fixed $k \geq 4$ the problem is **NP**-complete. The recognition
 problem for the analogue of clique-width for relational structures, cf. [12], has not been studied so far even for $k = 2$.
 29 However, once a parse tree is known the number of satisfying assignments can be efficiently calculated.

However, for our purposes, a recent result of Oum and Seymour [51] suffices to apply Theorem 1.8. They have shown
 31 that testing a graph for clique-width k is fixed parameter tractable, and an approximate parse tree can be produced in
 polynomial time in n .

33 **Theorem 6.2** (Oum and Seymour). *There is a function f , such that, for given k , there is a polynomial time algorithm
 that, with input a graph G , either concludes that its clique-width is $> k$ or outputs a $f(k)$ -parse tree for G . Its running
 35 time is $O(n^9 \log n)$ and $f(k) = 2^{3k+2} - 1$.*

By straight inspection of their proof a similar theorem can be proven also for the clique-width of signed graphs.

37 **Theorem 6.3.** *There is a function g , such that, for a given k , there is a polynomial time algorithm that, with input a
 signed graph G , either concludes that its signed clique-width is larger than k or outputs a $g(k)$ -parse tree for G . Its
 39 running time is $O(n^9 \log n)$ and $g(k) = 3^{3k+O(1)} = 2^{O(k)}$.*

Using the parse tree obtained from Theorem 6.3, we can produce $g(k)$ -parse trees for signed graphs with clique-width
 41 k , which makes our results applicable.

In [47] the following is shown for undirected clique-width, but the same proof gives it also for directed clique-width.
 43 To estimate the clique width this is often useful.

Proposition 6.4. *Clique-width is preserved for induced subgraphs. More precisely, if G is a (undirected, signed,
 45 directed) graph, and H is an induced (undirected, signed, directed) subgraph of G , then we have*

$$cw(H) \leq cw(G),$$

1 respectively,

$$scw(H) \leq scw(G).$$

3 6.2. Clique-width of clause graphs

5 We noted already that for the unsigned clique-width it is shown in [25] that clique-width of graphs of tree-width k , is at most $2^{k+1} + 1$. Hence we have

Proposition 6.5. *Let Σ be a set of clauses. Then we have*

- 7 (i) $cw(P(\Sigma)) \leq 2^{tw(P(\Sigma))+1} + 1$ and
 (ii) $cw(I(\Sigma)) \leq 2^{tw(I(\Sigma))+1} + 1$.

9 However, a bound on the clique-width of $P(\Sigma)$ gives no computational advantage.

Proposition 6.6. *Let $SAT(cw2)$ be SAT restricted to sets of clauses Σ with $cw(P(\Sigma)) = 2$.*

- 11 (i) $SAT(cw2)$ is **NP**-complete.
 (ii) $\sharp SAT(cw2)$ is $\sharp P$ -complete.

13 This follows immediately from:

15 **Lemma 6.7.** *For every set of clauses Σ in n variables v_1, \dots, v_n we define a set of clauses Σ' in $n + 1$ variables v_0, v_1, \dots, v_n by*

$$\Sigma' = \Sigma \cup \{v_0\} \cup \{v_i \vee v_j \vee v_0 : i, j \geq 1, i \neq j\}.$$

17 *For an assignment z for the variables v_1, \dots, v_n we define the assignment z' for v_0, v_1, \dots, v_n by setting $z(v_0) = 1$. Then we have*

- 19 (i) z makes Σ true iff z' makes Σ' true.
 (ii) $P(\Sigma')$ is a clique, hence $cw(P(\Sigma')) = 2$.

21 Next, we compare the clique-width of the signed and the unsigned cases:

Proposition 6.8.

23
$$cw(I(\Sigma)) \leq 2 \cdot scw(SI(\Sigma)).$$

25 **Proof (Sketch).** We take a parse tree der_{SI} for $SI(\Sigma)$. By doubling the number of colours (separating clause-vertices from variable-vertices we get for each colour i two colours i_c and i_v) we can disregard the bipartite character of the graphs. For this we replace each operation $\eta_{i,j}$ by η_{i_c, j_v} . The resulting parse tree is a parse tree for $I(\Sigma)$, where all the operation $\eta_{i,j}$ have different indices i, j . \square

29 Let G be any graph (not necessarily a clause graph of Σ). The incidence graph $I(G) = (V \cup E, F)$ of a graph (V, E) is the bipartite graph with V and E as vertex sets, and $(v, e) \in F$ iff v is a vertex of e . Clique-width and tree-width behave quite differently, when passing from G to $I(G)$.

31 **Proposition 6.9.** (Folklore) *For every graph, $tw(G) = tw(I(G))$.
 ([48]) $cw(K_n) = 2$, but $cw(I(K_n))$ goes to infinity with n .*

33 A converse inequality to the one in Proposition 6.8 does not hold.

Proposition 6.10. For every m there is a set of clauses Σ_m such that

- (i) $I(\Sigma_m) = K_{m,n}$, the complete bipartite graph on m and n elements with $n = \binom{m}{2}$. Hence $cw(I(\Sigma_m)) = 2$;
- (ii) The clique width $scw(SI(\Sigma_m))$ is a function of m which tends to infinity with m .

Proof (Sketch). Let the variables be v_1, \dots, v_m . For each $i \neq j \leq m$ let $C_{i,j}$ be the clause containing all the variables, but where exactly v_i and v_j occur negatively. Σ_m is the set of such clauses. Clearly, $I(\Sigma_m) = K_{m,n}$, the complete bipartite graph on m and n elements with $n = \binom{m}{2}$. So (i) is established. To see (ii), assume $der_{SI}(m)$ is a parse tree for (Σ_m) . We omit each η^+ in der_{SI} to obtain a parse tree $der_I(m)$. But $der_I(m)$ is a parse tree for $I(K_m)$, which is unbounded by Proposition 6.9. Note that here we use Proposition 6.4. \square

6.3. Clique-width of pigeon-hole and Tseitin formulas

We return to the examples of Section 3.3. First we quote from [37]

Proposition 6.11. The clique-width of the grid graphs $Grid_{n,n}$ is at least n .

From this, together with Proposition 6.4, the following is not difficult to show.

Proposition 6.12. The undirected, and hence the directed clique-width of the pigeon-hole formulas and the Tseitin formulas is unbounded.

6.4. Main result for bounded clique-width

We restate from the Introduction

Theorem 1.8. Given a set of clauses Σ and a signed parse tree $der_{SI}(\Sigma)$ for clique-width of up to k , it is possible to calculate $csat(\Sigma)$, with a number of algebraic operations that is linear in the size of the parse tree $der_{SI}(\Sigma)$, and exponential in k .

Remark 6.13. (i) The corresponding theorem for unsigned clique-width seems to be true as well, but the proof may be more involved and we did not check it in detail.

(ii) Although bounded tree-width of a class of graphs implies bounded clique-width of the same class, cf. Proposition 6.5, the clique-width grows exponentially. Therefore, Theorem 1.8 does not imply Theorem 1.3, even if the unsigned version of Theorem 1.8 is true.

The proof is given in Section 6.5. We leave it to the reader to formulate and prove the corresponding theorem for GENSAT.

Before we continue, we define some possible transformations of formulas corresponding to vertex-coloured edge-signed bipartite graphs.

Definition 6.14. Given subsets A, B, C of $\{1, \dots, k\}$ (not necessarily disjoint), and a formula Σ whose signed graph $SI(\Sigma)$ is vertex-coloured with $\{1, \dots, k\}$, we define $\Sigma^{(A,B,C)}$ as the formula resulting from Σ by the following operations:

- (i) Every clause in Σ whose vertex is coloured with a member of A is removed (but we do nothing with variables whose vertices are coloured with members of A).
- (ii) For $i \in \{1, \dots, k\}$, denote by X_i the set of variables whose vertices are coloured with i . For every $i \in B$ we add a clause consisting of the disjunction of all the variables in X_i .
- (iii) For every $i \in C$ we add a clause consisting of the disjunction of all the negations of the variables in X_i .

Note that in particular $\Sigma = \Sigma^{(\emptyset, \emptyset, \emptyset)}$.

1 6.5. Reduction lemmas

We assume w.l.o.g. that in the parse tree $der_{SI}(\Sigma)$ of Σ , all unions are made between graphs that use disjoint subsets of the colour set in their vertex colouring. The reason for this is that given a parse tree that does not satisfy this condition and uses k vertex colours in all, one can easily construct a parse tree with $2k$ vertex colours for which this additional condition holds.

To prove Theorem 1.8, we calculate for every node v of the parse tree $der_{SI}(\Sigma)$ not only the value $csat(\Sigma_v)$ for the formula Σ_v constructed by the operation of that node, but we also calculate $csat(\Sigma_v^{(A,B,C)})$ for every $A, B, C \subseteq \{1, \dots, k\}$, which are used in the reductions through which we obtain the final $csat(\Sigma)$. We use the following reduction lemmas.

Lemma 6.15. *If the operation in node v is a disjoint union of its children u and w , then*

$$11 \quad csat(\Sigma_v^{(A,B,C)}) = csat(\Sigma_u^{(A,B,C)}) \cdot csat(\Sigma_w^{(A,B,C)})$$

for every A, B, C .

Proof. We assumed above that in all disjoint unions, the colour sets used by the two children are also disjoint, and under this assumption it is not hard to see that the above holds. \square

Lemma 6.16. *If the operation in v is $\rho_{i,j}(\Sigma_w)$ where w is the child of v , for every A, B, C it is possible to calculate $csat(\Sigma_v^{(A,B,C)})$ from the values stored for w using a constant number of operations.*

Proof. If $i \in B$ or $i \in C$ then $csat(\Sigma_v^{(A,B,C)}) = 0$, because Σ_v contains no variables coloured with i and hence $\Sigma_v^{(A,B,C)}$ contains an empty (unsatisfiable) clause. From now on we assume that B and C do not contain i . If $j \in A$ we set $A' = A \cup \{i\}$, and otherwise we set $A' = A \setminus \{i\}$. We now distinguish four cases:

Case 1: B and C do not contain j .

In this case clearly

$$13 \quad csat(\Sigma_v^{(A,B,C)}) = csat(\Sigma_w^{(A',B,C)}).$$

Case 2: B contains j but C does not.

In this case we use the inclusion–exclusion principle.

We set $B_1 = B \cup \{i\} \setminus \{j\}$, $B_2 = B$, and $B_3 = B \cup \{i\}$, and obtain

$$15 \quad csat(\Sigma_v^{(A,B,C)}) = csat(\Sigma_w^{(A',B_1,C)}) + csat(\Sigma_w^{(A',B_2,C)}) - csat(\Sigma_w^{(A',B_3,C)}).$$

Case 3: C contains j but B does not.

This is analogous to the previous case. In this case we set $C_1 = C \cup \{i\} \setminus \{j\}$, $C_2 = C$, and $C_3 = C \cup \{i\}$, and obtain

$$17 \quad csat(\Sigma_v^{(A,B,C)}) = csat(\Sigma_w^{(A',B,C_1)}) + csat(\Sigma_w^{(A',B,C_2)}) - csat(\Sigma_w^{(A',B,C_3)}).$$

Case 4: Both B and C contain j .

We define $B_1, B_2, B_3, C_1, C_2, C_3$ as above and use again the inclusion–exclusion principle, but this time the resulting formula is somewhat more complex:

$$19 \quad csat(\Sigma_v^{(A,B,C)}) = csat(\Sigma_w^{(A',B_1,C_1)}) + csat(\Sigma_w^{(A',B_1,C_2)}) + csat(\Sigma_w^{(A',B_2,C_1)}) + csat(\Sigma_w^{(A',B_2,C_2)}) \\ - csat(\Sigma_w^{(A',B_3,C_1)}) - csat(\Sigma_w^{(A',B_3,C_2)}) - csat(\Sigma_w^{(A',B_1,C_3)}) - csat(\Sigma_w^{(A',B_2,C_3)}) \\ + csat(\Sigma_w^{(A',B_3,C_3)}). \quad \square$$

Lemma 6.17. *If the operation in v is $\eta_{i,j}^+(\Sigma_w)$ where w is the child of v , for every A, B, C it is possible to calculate $csat(\Sigma_v^{(A,B,C)})$ from the values stored for w using a constant number of operations.*

1 **Proof.** If $i \in A$ then clearly

$$\text{csat}(\Sigma_v^{(A,B,C)}) = \text{csat}(\Sigma_w^{(A,B,C)}),$$

3 and if $j \in B$ then clearly

$$\text{csat}(\Sigma_v^{(A,B,C)}) = \text{csat}(\Sigma_w^{(A \cup \{i\}, B, C)}).$$

5 Otherwise we note that a satisfying assignment for $\Sigma_v^{(A,B,C)}$ is an assignment that satisfies $\Sigma_w^{(A,B,C)}$ or $\Sigma_w^{(A \cup \{i\}, B \cup \{j\}, C)}$, and we use the inclusion–exclusion principle to obtain

$$7 \quad \text{csat}(\Sigma_v^{(A,B,C)}) = \text{csat}(\Sigma_w^{(A,B,C)}) + \text{csat}(\Sigma_w^{(A \cup \{i\}, B \cup \{j\}, C)}) - \text{csat}(\Sigma_w^{(A, B \cup \{j\}, C)}). \quad \square$$

9 **Lemma 6.18.** *If the operation in v is $\eta_{i,j}^-$ (Σ_w) where w is the child of v , for every A, B, C it is possible to calculate $\text{csat}(\Sigma_v^{(A,B,C)})$ from the values stored for w using a constant number of operations.*

Proof. Virtually identical to that of the previous lemma. \square

11 **Proof of Theorem 1.8.** We start with the leaves and go upward, at every node v calculating $\text{csat}(\Sigma_v^{(A,B,C)})$ for all
 13 possible A, B, C . For every node the total number of calculated values is exponential in k , and the number of operations
 to calculate each of them is constant, and so the number of operations required to reach the root is linear in the size of
 the tree and exponential in k . \square

15 7. Conclusions and further research

We have presented evidence from the literature that sets of clauses Σ with clause graphs of bounded tree-width
 17 or clique-width can be derived from real-world applications. Small tree-width and small clique-width are structural
 properties of the various clause graphs. Engineering artefacts come with built in modularization with minimal or well
 19 structured interconnectivity which imply these structural properties, cf. [4].

We have shown how to solve SAT, GENSAT, \sharp SAT and \sharp GENSAT efficiently on sets of clauses with incidence
 21 graphs of tree-width at most k . Our new algorithm has feasible constants, when k is not too large. It also allows us to
 solve SAT efficiently, but it remains to be checked whether it is more efficient than the resolution method, applied to
 23 formulas of bounded tree-width as presented in [2].

We have also shown how to use parse trees of signed clique-width efficiently to solve SAT and \sharp SAT. This widens
 25 the applicability of our results considerably, especially, since Oum and Seymour have shown that finding a suitable
 parse tree for (signed) graphs is fixed parameter tractable (in FPT).

Our methods apply also to any other problem which is reducible to SAT by polynomial time Turing reductions where
 27 the tree-width or clique-width is bounded. We have shown how to use this for various versions of GENSAT.

It would be interesting to see, for which versions of GENSAT there are splitting formulas similar to the one given in
 29 Theorem 4.7. Such splitting formulas are bound to give better constants than the ones one gets by using reductions.

The results of [23,44] give general splitting theorems and polynomial time algorithms for many other counting
 31 problems. It remains a challenge to find direct proofs for simpler splitting formulas, say, for counting perfect matchings,
 33 hamiltonian cycles or various colourings.

Acknowledgements

We thank A. Magid and Y. Magid, the first author's M.Sc. students, for checking our first proof of Theorem 4.7, A.
 Glikson for Fig. 1, H. Bodlaender and J. Naor for guidance to the literature. We also thank Traldi, whose [57] he made
 available to us before publication and which inspired us to write this paper. Finally, we thank three anonymous referees
 for very valuable suggestions.

35 References

- [1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

- 1 [2] M. Alekhnovich, A. Razborov, Satisfiability, branch-width and Tseitin tautologies, in: Proceedings of the 43rd Annual Symposium on the
 3 Foundations of Computer Science (FOCS 2002), Vancouver, British Columbia, Canada, IEEE Computer Society Press, Silver Spring, MD,
 2002, pp. 593–603.
- 5 [3] E. Amir, (de)Composition of situation calculus theories, in: Proceedings of the National Conference on Artificial Intelligence (AAAI'00),
 AAAI Press, MIT Press, Cambridge, MA, 2000, pp. 456–463.
- 7 [4] E. Amir, Dividing and conquering logic, Ph.D. Thesis, Department of Computer Science, Stanford University, 2001.
- 9 [5] E. Amir, Approximation algorithms for treewidth, 2002, submitted for publication.
- 11 [6] E. Amir, S. McIlraith, Theorem proving with structured theories, in: Proceedings of the 17th International Joint Conference on Artificial
 Intelligence (IJCAI'01), 2001, pp. 624–631.
- 13 [7] E. Amir, S. McIlraith, Partition-based logical reasoning for first-order and propositional theories, *Artificial Intelligence* 162 (1–2) (2005) 49–
 88.
- 15 [8] A. Andrzejak, Splitting formulas for Tutte polynomials, *J. Combin. Theory Ser. B* 70 (2) (1997) 346–366.
- 17 [9] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embedding in a k -tree, *SIAM. J. Algebraic Discrete Methods* 8 (1987) 277
 –284.
- 19 [10] A. Atserias, V. Dalmau, A combinatorial characterization of resolution width, in: Proceedings of the 18th IEEE Conference on Computational
 Complexity (CCC 2003), Aarhus, Denmark, 2003, pp. 239–247.
- 21 [11] P. Bjesse, J. Kukula, R. Damiano, T. Stanion, Y. Zhu, Guiding SAT diagnosis with tree decompositions, in: E. Giunchiglia, A. Tacchella (Eds.),
 Theory and Applications of Satisfiability Testing: Sixth International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5–8, 2003,
 Selected Revised Papers, Lecture Notes in Computer Science, vol. 2919, Springer, Berlin, 2004, pp. 315–329.
- 23 [12] A. Blumensath, B. Courcelle, Recognizability and hypergraph operations using local information, *Inform. and Comput.* xx(x) (200x) xx–yy.
- 25 [13] H. Bodlaender, Treewidth: algorithmic techniques and results, in: I. Privara, P. Ruzicka (Eds.), Proceedings of the 22nd International Symposium
 on the Mathematical Foundation of Computer Science, MFCS'97, Lecture Notes in Computer Science, vol. 1295, Springer, Berlin, 1997, pp.
 29–36.
- 27 [14] B. Bollobás, *Modern Graph Theory*, Springer, Berlin, 1999.
- 29 [15] V. Chvátal, E. Szemerédi, Many hard examples for resolution, *J. ACM* 35 (4) (1988) 759–768.
- 31 [16] P. Cohen, P. Schrag, R. Jones, E. Pease, A. Lin, A. Starr, B. Gunning, M. Burke, The DARPA high-performance knowledge base project, *AI
 Magazine* 19 (4) (1998) 25–49.
- 33 [17] D.G. Corneil, M. Habib, J.-M. Lanlignel, B. Reed, U. Rotics, Polynomial time recognition of clique-width ≤ 3 graphs, in: Proceedings of
 LATIN'2000, Lecture Notes in Computer Science, vol. 1776, Springer, Berlin, 2000, pp. 126–134.
- 35 [18] B. Courcelle, Monadic second-order logic of graphs VII: graphs as relational structures, *Theoret. Comput. Sci.* 101 (1992) 3–33.
- 37 [19] B. Courcelle, The monadic second order logic of graphs, XIV: uniformly sparse graphs and edge set quantification, *Theoret. Comput. Sci.* 299
 (1) (2003) 1–36.
- 39 [20] B. Courcelle, J. Engelfriet, A logical characterization of the sets of hypergraphs defined by hyperedge replacement grammars, *Math. Systems
 Theory* 28 (1995) 515–552.
- 41 [21] B. Courcelle, J. Engelfriet, G. Rozenberg, Handle-rewriting hypergraph grammars, *J. Comput. System Sci.* 46 (1993) 218–270.
- 43 [22] B. Courcelle, J.A. Makowsky, Fusion on relational structures and the verification of monadic second order properties, *Math. Structures Comput.
 Sci.* 12 (2) (2002) 203–235.
- 45 [23] B. Courcelle, J.A. Makowsky, U. Rotics, On the fixed parameter complexity of graph enumeration problems definable in monadic second order
 logic, *Discrete Appl. Math.* 108 (1–2) (2001) 23–52.
- 47 [24] B. Courcelle, M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, *Theoret. Comput. Sci.* 109 (1993) 49–82.
- 49 [25] B. Courcelle, S. Olariu, Upper bounds to the clique-width of graphs, *Discrete Appl. Math.* 101 (2000) 77–114.
- 51 [26] N. Creignou, M. Hermann, Complexity of generalized satisfiability counting problems, *Inform. and Comput.* 125 (1996) 1–12.
- 53 [27] N. Creignou, S. Khanna, M. Sudhan, Complexity Classifications of Boolean Constraint Satisfaction Problems, *SIAM Monographs on Discrete
 Mathematics and Applications*, SIAM, Philadelphia, PA, 2001.
- 55 [28] R. Dechter, J. Pearl, The clustering for constraint networks, *Artificial Intelligence* 38 (1999) 353–366.
- 57 [29] R. Diestel, *Graph Theory*, Graduate Texts in Mathematics, Springer, Berlin, 1996.
- 59 [30] R.G. Downey, M.F. Fellows, *Parametrized Complexity*, Springer, Berlin, 1999.
- 61 [31] J. Engelfriet, V. van Oostrom, Logical description of context-free graph-languages, *J. Comput. System Sci.* 55 (1997) 489–503.
- [32] T. Feder, M. Vardi, The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group
 theory, *SIAM J. Comput.* 28 (1999) 57–104.
- [33] M. Frick, M. Grohe, The complexity of first-order and monadic second-order logic revisited, *Ann. Pure Appl. Logic* 130 (1) (2004) 3–31.
- [34] Z. Galil, On the complexity of regular resolution and the Davis–Putnam procedure, *Theoret. Comput. Sci.* 4 (1977) 23–46.
- [35] M.G. Garey, D.S. Johnson, *Computers and Intractability*, Mathematical Series, W.H. Freeman and Company, New York, 1979.
- [36] A. Glikson, J.A. Makowsky, NCE graph grammars and clique-width, in: H.L. Bodlaender (Ed.), Proceedings of the 29th International Workshop
 on Graph-Theoretic Concepts in Computer Science (WG 2003), Elspeet, The Netherlands, Lecture Notes in Computer Science, vol. 2880,
 Springer, Berlin, 2003, pp. 237–248.
- [37] M.C. Golumbic, U. Rotics, On the clique-width of some perfect graph classes, *Internat. J. Foundations Comput. Sci.* 11 (2000) 423–443.
- [38] G. Gottlob, R. Pichler, Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width, 28th International Colloquium on
 Automata, Languages and Programming, ICALP'01, Lecture Notes in Computer Science, vol. 2077, Springer, Berlin, 2001, pp. 708–719.
- [39] A. Haken, The intractability of resolution, *Theoret. Comput. Sci.* 39 (1985) 297–308.
- [40] A. Itai, J.A. Makowsky, Unification as a complexity measure for logic programming, *J. Logic Programming* 4 (2) (1987) 105–117.
- [41] D.S. Johnson, M. Yannakakis, C.H. Papadimitriou, On generating all maximal independent sets, *Inform. Process. Lett.* 27 (3) (1988) 119–123.

Q2

- 1 [42] D.B. Lenat, Cyc: a large-scale investment in knowledge infrastructure, *Comm. ACM* 38 (11) (1995) 33–38.
- 3 [43] J.A. Makowsky, Colored Tutte polynomials and Kauffman brackets on graphs of bounded tree width, in: *Proceedings of the 12th Symposium on Discrete Algorithms*, SIAM, Philadelphia, PA, 2001, pp. 487–495.
- 5 [44] J.A. Makowsky, Algorithmic uses of the Feferman–Vaught theorem, *Ann. Pure Appl. Logic* 126 (2004) 1–3.
- 7 [45] J.A. Makowsky, Colored Tutte polynomials and Kauffman brackets on graphs of bounded tree width, *Discrete Appl. Math.* 145 (2) (2005) 276–290.
- 9 [46] J.A. Makowsky, J.P. Mariño, Farrell polynomials on graphs of bounded treewidth, *Adv. in Appl. Math.* 30 (2003) 160–176.
- 11 [47] J.A. Makowsky, J.P. Mariño, Tree-width and the monadic quantifier hierarchy, *Theoret. Comput. Sci.* 303 (2003) 157–170.
- 13 [48] J.A. Makowsky, U. Rotics, On the cliquewidth of graphs with few P_4 's, *Internat. J. Foundations Comput. Sci.* 10 (1999) 329–348.
- 15 [49] S. Negami, Polynomial invariants of graphs, *Trans. Amer. Math. Soc.* 299 (1987) 601–622.
- 17 [50] S.D. Noble, Evaluating the Tutte polynomial for graphs of bounded tree-width, *Combin. Probab. Comput.* 7 (1998) 307–321.
- 19 [51] S. Oum, P. Seymour, Approximating clique-width and branch-width, *J. Combin. Theory Ser. B* xx (2006) xx–yy in press, electronically available since January 5, 2006.
- 21 [52] J.G. Oxley, D.J.A. Welsh, Tutte polynomials computable in polynomial time, *Discrete Math.* 109 (1992) 185–192.
- 23 [53] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
- 25 [54] T.J. Schaefer, The complexity of satisfiability problems, in: *Proceedings 10th Symposium on Theory of Computing (STOC'78)*, San Diego, California, USA, 1978, pp. 216–226.
- 27 [55] S. Szeider, On fixed-parameter tractable parameterizations of SAT, in: *Sixth International Conference on Theory and Applications of Satisfiability Testing, (SAT'03)* S. Margherita Ligure, Italy, *Lecture Notes in Computer Science*, Springer, Berlin, 2003, pp. 188–202.
- [56] M. Thorup, Structured programs have small tree-width and good register allocation, *Inform. and Comput.* 142 (1998) 159–181.
- [57] L. Traldi, A note on the colored Tutte polynomial of a graph of bounded tree width, Preprint, January 2003.
- [58] G.S. Tseitin, On the complexity of derivation in propositional calculus, in: A.O. Slisenko (Ed.), *Structures in Constructive Mathematics and Mathematical Logic, Part II*, 1968, pp. 115–125 (Translation from Russian).
- [59] L.G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.* 8 (3) (1979) 410–421.
- [60] Y. Zhu, D. Wang, E. Clarke, J. Kukula, Using cutwidth to improve symbolic simulation and boolean satisfiability, in: *Proceedings of the Sixth IEEE International High-Level Design Validation and Test Workshop (HLDVT'01)*, IEEE Computer Society, Silver Spring, MD, 2001, pp. 165–170.